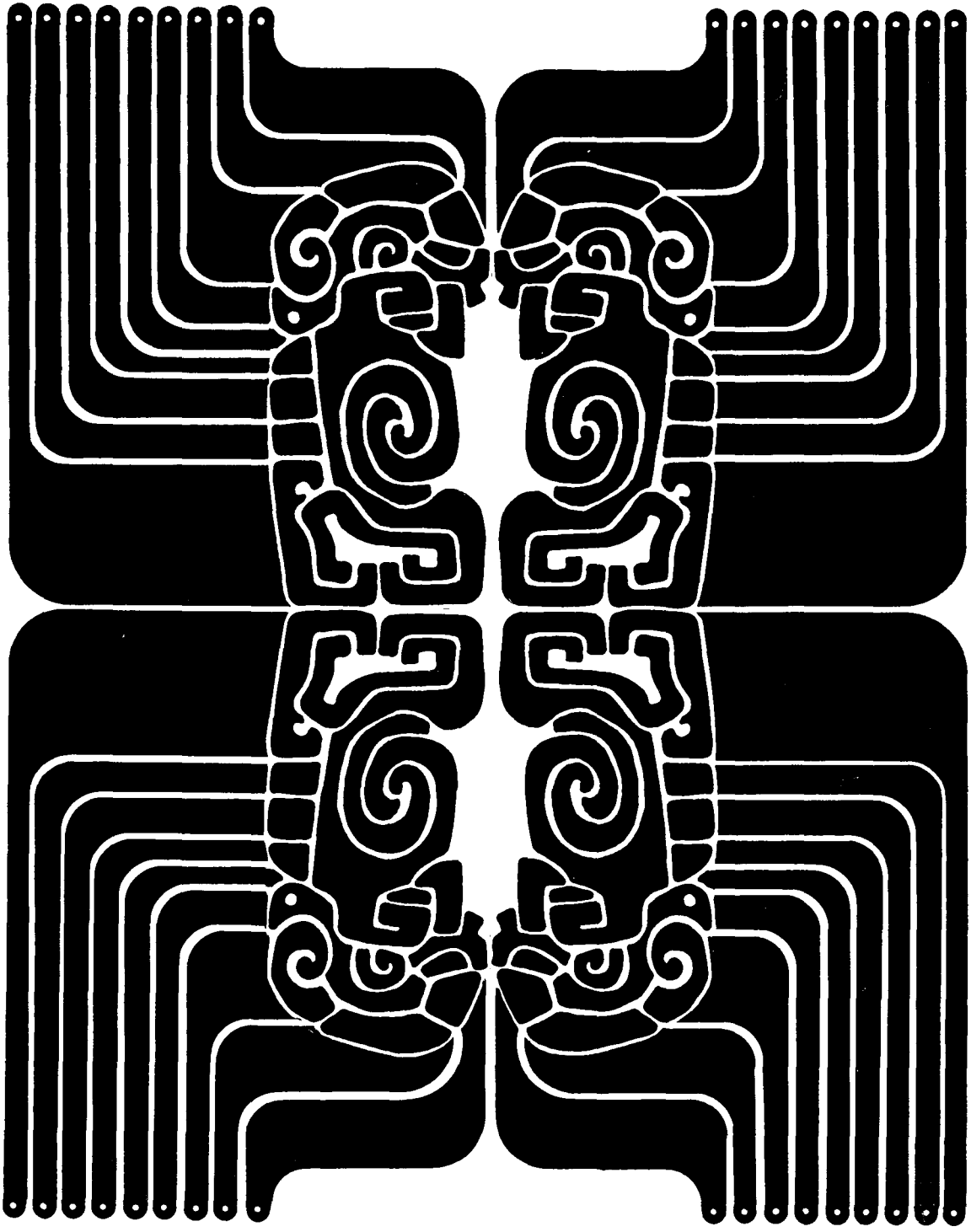


Proceedings of the Symposium on Recent Developments in Computing, Processor and Software Research for High-Energy Physics

Guanajuato, Mexico, May 8-11, 1984

Editors
Rene Donaldson
Michael N. Kreisler

Sponsored by CoNaCyt (Mexico), Fermilab, National Science Foundation (USA), Secretaría de Educación Pública, Universidad de Guanajuato, Universidad de México, University of Massachusetts at Amherst, the U.S. Department of Energy

FOREWORD

The Symposium on Recent Developments in Computing, Processor, and Software Research for High Energy Physics was held in early May 1984 in Guanajuato, Mexico. The Symposium brought together many of the experts in the field who have been addressing the problems of handling huge data samples and gargantuan computational tasks.

In these proceedings, we have tried to capture not only the details of the technical papers, but also the intensity of the give-and-take in the questions and answers. The Symposium was extremely exciting as the many different groups hotly debated the relative merits of various proposed solutions. The active participation of computer industry representatives gave the Symposium an added flair. We hope we have been successful in capturing that intensity.

In addition to thanking the sponsors of the Symposium, a special note of appreciation must go to Governor Velazco Ibarra, the Govenor of the State of Guanajuato. In addition to being the gracious host of our international gathering, he is to be thanked once again for a spectacular conference banquet and the callejoneada which followed.

The Symposium was organized by C. Avilez, Universidad Nacional Autonoma de México; A. Garcia, Universidad de Guanajuato; M. Kreisler, University of Massachusetts at Amherst; and T. Nash, Fermilab. The Symposium Secretariat was R. Donaldson, Fermilab, and I. Menocal, Universidad Nacional Autonoma de México.

We would also like to thank several people who made the editing of these proceedings possible. Angela Gonzales of Fermilab has done a stellar job on the artwork; Susan Winchester of Fermilab and Nellie Bristol and Judy Ksieniewicz of the University of Massachusetts have suffered under many revisions. The photograph of the Teatro Juárez on page 456 was taken by Joaquín Escalona, Universidad Nacional Autonoma de México. Thanks to all.

R. Donaldson
M. Kreisler
August 1984

PROCEEDINGS OF THE SYMPOSIUM ON RECENT DEVELOPMENTS
IN COMPUTING, PROCESSOR, AND SOFTWARE RESEARCH
FOR HIGH-ENERGY PHYSICS

TABLE OF CONTENTS

Lattice Gauge and General Theoretical Processors and Computing

Multiprocessor Projects

e^+e^- Colliders On and Offline Reconstruction Processors

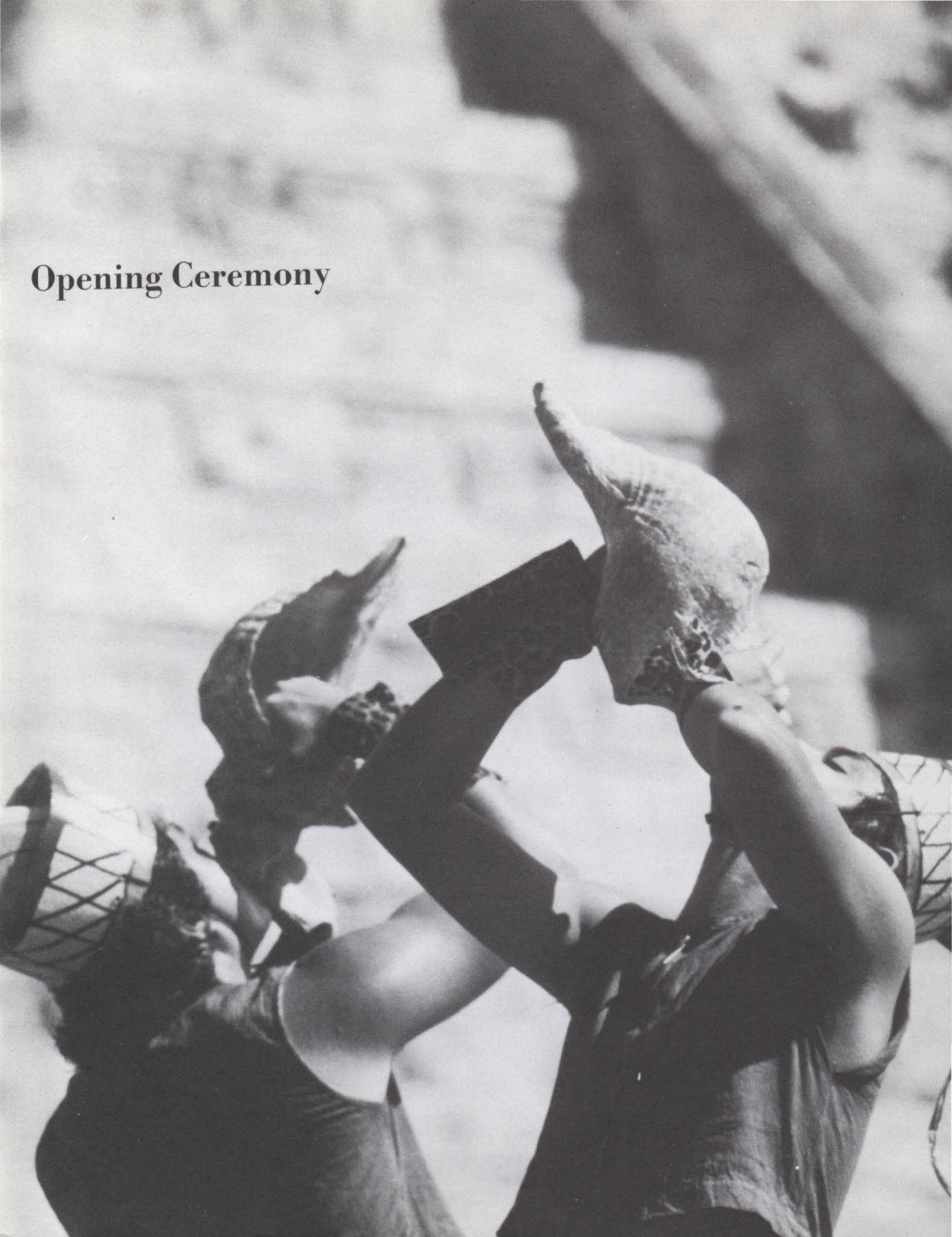
State of the Art in University Computer Science and Industry

Software Research

$\bar{p}p$ Colliders On and Offline Reconstruction Processors

Post-Deadline

Opening Ceremony



[Opening ceremony, La Fiesta de la Primavera, Chichén Itzá, Yucatán, México. Photograph by Anthony R. Donaldson.]

DISCURSO PRONUNUADO

Nestor Raul Luna Hernández
Rector, University of Guanajuato
Guanajuato, Gto., Mexico

[Editors' Note: We present here Rector Luna's talk in Spanish as it was given. The Symposium was officially opened by Dr. Jorge Flores, the Undersecretary of Education of Mexico.]

Si el saber teórico permite llegar al conocimiento en el campo de la ciencia pura, ésta a su vez deriva a lo que denominado actualmente tecnología, es el saber práctico proyectado hacia la construcción y el progreso.

Mucho se objecta la práctica de la epísteme por la inmitación que impone de la realidad hasta llegar a su exclusión y son por ello afectados los científicos que dedicados a la teoría, de ella parten para forjar otra realidad.

La realidad que de la ciencia deriva no puede ser si no producto del saber puro, más no por esto debe permanecer en el estrato de lo ideal sino plasmarse en su consecuente concreto y aprovechable por la experiencia sensible.

Este simposio sobre desarrollos recientes de procesadores, computación e investigación en el campo de la física de altas energías, concede la oportunidad de confirmarlo. Su ámbito, perteneciente al más puro y elevado saber, avanza en importancia para el desarrollo de la tecnología indispensable a la evolución de la realidad futura.

Su concresión se aprecia en técnicas desarrolladas y aplicadas a la solución de problemas en diversas áreas de la ciencia y manejo de datos esenciales para la informática.

El porvenir de la humanidad precisa cada vez más de la ciencia y sus proyecciones, el deber del científico es conocerla y transformarla para beneficio de un universo día a día más complicado, alejado de la naturaleza y dependiente de la tecnología.

En estas reuniones científicas, que permiten el intercambio de experiencias en problemas de investigadores se afirmarán las relaciones entre expertos del campo de la ciencia pura y se establecerán contactos entre especialistas que expresarán el saber más alto aplicado en la solución de una amplia gama de complejas situaciones, cuya clarificación deje ver el avance en el uso creciente de estos recursos.

El país requiere de tecnología propia y la esta produciendo, solo necesita que sea dada a conocer y evaluada debidamente.

Identificar necesidades técnologicas y dar difusion a los avances de su conocimiento deben ser preocupaciones fundamentales de quienes se ocupan de servir a la sociedad a través del más elevado conocimiento. Bien venidos señores congresistas. Muchas gracias.

INTRODUCTORY REMARKS

Michael N. Kreisler
University of Massachusetts, Amherst, Massachusetts

Governor Velazco Ibarra, Subsecretary Flores, Dr. Jaime Tacher, the representative of the Director General of CoNaCyt, Rector Luna, other honored guests, and my fellow scientists: On behalf of the Organizing Committee, I take great pleasure in welcoming you to Guanajuato and to the Symposium on Recent Developments in Computing, Processor, and Software Research for High Energy Physics.

Let me take this opportunity at the beginning of the Symposium both to thank and to congratulate our Guanajuato hosts for their cooperation and extremely hard work in preparing for and holding this international gathering. Without the diligent efforts of the Universidad de Guanajuato, the Symposium would not have been possible. Thank you, Rector Luna.

In addition to the enthusiastic support of our colleagues in Guanajuato and the Universidad de Guanajuato, we have been fortunate enough to recieve the support of the Governor of Guanajuato, the Subsecretary of Education, and the Director General of CoNaCyt. Our hosts have done and are doing an excellent job--one that speaks extremely well as an indication to the international scientific community of the wisdom of holding future conferences in Mexico.

They've done their job very well--we now have to get on with ours.

As most of us realize, the problem that has caused us to gather in this charming city is extremely pressing. Despite the rapid growth in computers and related technology, we are painfully aware that there are many crucial questions which cannot be addressed with either current technology or with that technology one could reasonable expect to exist in the commercial sector in a few years. Those problems involve either the analysis of huge complex data banks or laborious multidimensional calculations as in gauge theories or weather simulation. Most laboratories and universities making projections of computer needs for the near term future recognize that demand for access to even the most advanced current computers will outrun the financial ability of those institutions in a few years--at some, the current facilities are already inadequate.

One could question whether such a demand for computation is necessary--are the problems sufficiently important? The demand is not limited to the field of pure research in high-energy physics but rather represents a broad, growing awareness by all

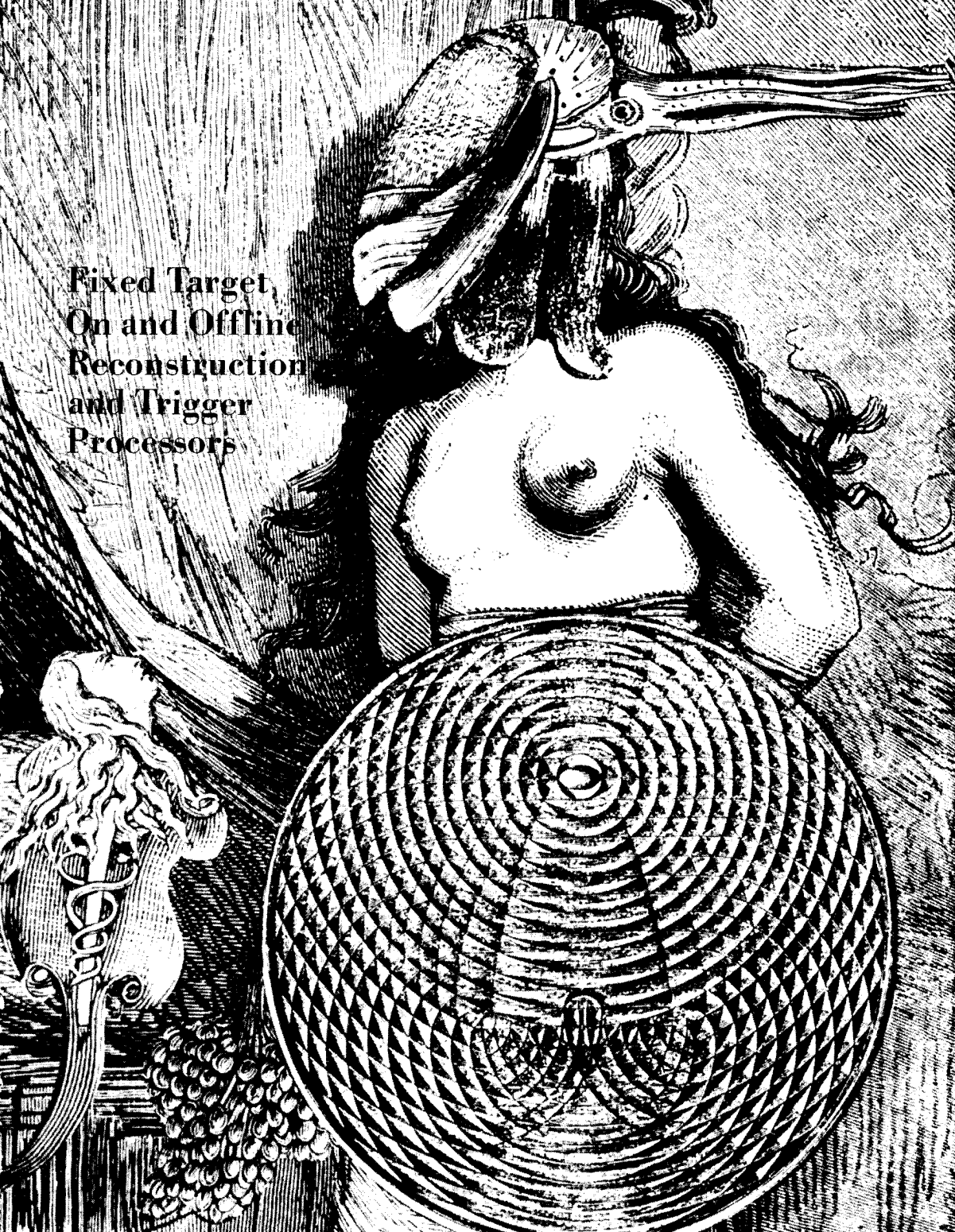
sectors of the economy--led, as usual, by extremely intelligent scientists--that one con obtain crucial answers to vitally important questions if one only had sufficient computation power.

Some of us like to view the importance of the questions in another related manner. In searching for the answers to the riddles of the structures and secrets of matter, we have been use to searching through a small handful of hay or straw looking for a needle of truth. Such searches have been extremely difficult, time consuming, and, of course, occasionally very rewarding. Those of us here realize that not only could a breakthrough in technology allow us to sift through those handfuls of hay much, much quicker, but we even might be able to begin attacking the large haystack against which we have been leaning. There are even visionaries among us who have taken the time to walk up a nearby hill and notice that the countryside is dotted with thousands of haystacks.

Our collective goal is to be able to make those breakthroughs yielding either new technological approaches and/or more efficient uses of existing resources. We would then be able to explore the new scientific problems lying so temptingly just outside our grasp. Obviously we are driven by the problems in high-energy physics. Our solutions will have a very wide impact throughout the technological community.

During the Symposium, we'll hear from spokesmen from many of the approaches which are being developed and tried. We look forward to learning lots of new things and to participating in lively debates about the virtues of competing ideas. Coupling such important, exciting technical problems with the charm of Mexico in general and Guanajuato in particular seems to promise a busy, enjoyable time for us all.

Again--welcome to Guanajuato.



Fixed Target On and Offline Reconstruction and Trigger Processors

[Collage by Max Ernst, 1891-1976, from "Une Semaine de Bonté," first published in 1934.]

A REVIEW OF TRIGGER AND ON–LINE PROCESSORS AT SLAC*

A. J. LANKFORD

Stanford Linear Accelerator Center
Stanford University, Stanford, California, 94305

1. INTRODUCTION

The role of trigger and on–line processors in reducing data rates to manageable proportions in e^+e^- physics experiments is defined not by high physics or background rates, but by the large event sizes of the general-purpose detectors employed. The rate of e^+e^- annihilation is low, and backgrounds are not high; yet the number of physics processes which can be studied is vast and varied.

This paper begins in Section 2 by briefly describing the role of trigger processors in the e^+e^- context. The usual flow of the trigger decision process is illustrated with selected examples of SLAC trigger processing. The examples discussed are the energy trigger of the ASP detector and the charged particle triggers of the Mark III and Mark II detectors. Section 3 sketches the features of triggering at the SLC and the trigger processing plans of the two SLC detectors: The Mark II and the SLD. In Section 4, the most common on-line processors at SLAC, the BADC, the SLAC Scanner Processor, the SLAC FASTBUS Controller, and the VAX CAMAC Channel, are discussed. Uses of the 168/E, 3081/E, and FASTBUS VAX processors are mentioned. The manner in which these processors are interfaced and the function they serve on line are described. Finally, Section 5 outlines the accelerator control system for the SLC. This paper is a survey in nature, and hence, relies heavily upon references to previous publications for detailed description of work mentioned here. In addition, apologies are deserved by all the experimenters whose work is overlooked or improperly acknowledged by this overview.

---

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.

2. TRIGGER PROCESSING FOR e^+e^- ANNIHILATIONS

e^+e^- collisions are provided by bunched beams of electrons and positrons which cross with frequency set by the physical scale of the accelerator. Trigger decisions can be made without deadtime during the period between crossings. This period is shown for the accelerators SPEAR, PEP, and SLC in Table I. The time between crossings at SPEAR and at PEP is sufficiently short that trigger decisions are normally made by multi-level hardware processors. The first-level trigger, made between crossings, reduces the beam-crossing frequency (1.3 MHz at SPEAR and 420 KHz at PEP) to a rate set by its allowed deadtime fraction. First-level trigger rates are typically about a kilo-Hertz. The second-level trigger rate is set by the readout time until an event is buffered. Rates are typically 2 - 5 Hz and deadtimes less than 10% . Some experiments also employ third-level triggers for long decision times on small numbers of events. The MAC experiment at PEP, for instance, reduces its trigger rate by about a factor of two with a software trigger decision requiring 10 to 20 msec on its VAX.

Table I. Beam crossing and interaction rates at SLAC e^+e^- accelerators

Accelerator	Period between beam crossings	Rate ($\sigma_{\rm pt}$)	Rate ($\sigma_{\rm tot}$)
SPEAR	780 *nsec*	$\sim 2 \times 10^{-2}$ *Hz*	$\lesssim 1$ *Hz*
PEP	2.3 μ*sec*	$\sim 2 \times 10^{-3}$ *Hz*	~ 0.01 *Hz*
SLC	5.5 *msec*	$\sim 7 \times 10^{-5}$ *Hz*	$\lesssim 0.25$ *Hz*

The rate of e^+e^- annihilations is parameterized as:

$$\text{rate} = R_{tot} \times \sigma_{pt} \times \mathcal{L} \ , \qquad \text{where} \quad \sigma_{pt} = \frac{87nb}{(2\ E_{\rm beam})^2} \tag{1}$$

is the point-like cross-section for $e^+e^- \rightarrow \mu^+\mu^-$. R_{tot} is the ratio of the total annihilation cross-section to σ_{pt}. R_{tot} is in the range 4 to 8 in continuum regions, and is itself an indication of the onset of new physics processes. On resonances, R_{tot} can be much greater. R_{tot} is ~ 2500 and ~ 25 on the $\psi(3100)$ and $\Upsilon(9460)$ respectively, and is expected to be about 4000 on the Z° resonance. $\mathcal{L}$ is the luminosity, and is characteristically within about a factor of two of $10^{31}\ cm^{-2}\ sec^{-1}$. Typical physics rates are shown in Table I for SPEAR, PEP, and SLC. The total physics rates are quite manageable; consequently, most detectors for e^+e^- physics are general purpose in nature and try to record all physics events. The detectors generally consist of cylindrical drift chambers in solenoidal magnet fields surrounded by calorimetry. Typical event sizes range from

a couple of kBytes to about a hundred kBytes. Similar detector geometries result in similar trigger problems and generic solutions.

The general purpose nature of e^+e^- experiments require that triggers include a broad range of physics topologies, from multiparticle hadronic events to two-track leptonic events, single electron events, two-photon or one-photon final states, and decays of possible long-lived particles. These events generally fall into two broad categories: events with two or more charged tracks and events with one or more energetic showers. Trigger designs generally consist of parallel logic to identify these event types. Charged particle triggers define a track as a set of tracking chamber hits in a trigger road. The numbers of planes of tracking available to and required by the trigger processor vary among experiments, as do the precision with which roads are defined and the momentum range covered. Neutral energy triggers discriminate on local or global sums of energy from calorimeter channels. The thresholds accessible and chosen depend on the type of calorimeter and the way in which the sums are formed. Information from charged particle and neutral energy triggers may be combined in other triggers. Additional parallelism at all decision levels provides redundancy helpful for determining trigger efficiency.

The backgrounds to the physics triggers arise from cosmic rays, beam-gas collisions, beam-pipe collisions of off-energy electrons, synchrotron radiation, and electronic pick-up. The sources of background can generally be reduced to quite manageable magnitudes by careful masking and shielding. Residual background is reduced by limiting acceptance in the radial and longitudinal position of and in the time of the interaction. Ability to reject random hits can also be important. Most trigger processors require that the projection of candidate tracks in the plane perpendicular to the beam axis ($r-\phi$ plane) pass through a fiducial area surrounding the beam-crossing point ($r = 0, z = 0$). In addition, the track must be in time with the beam-crossing. Few experiments are able to project track candidates longitudinally along the beam-direction (z); so experiments commonly have inner trigger chambers or vertex detectors which restrict the acceptance in this dimension. The TPC detector[1] at PEP, however, uses its drift time measurement along the beam direction to project tracks toward the beam-crossing point in $r - z$ planes instead of the $r - \phi$ plane.

The following sections describe examples of SLAC trigger processors which illustrate solutions to the problems of trigger efficiency and background rejection. The examples discussed are the energy trigger of the ASP detector and the charged particle triggers of the Mark III and Mark II detectors.

2.1 THE ASP ENERGY TRIGGER

The trigger logic of the ASP Detector[2] provides an example of an energy trigger accomplished between PEP beam crossings so as to have no deadtime. This new detector, designed for the detection of anomalous single photons, consists of four walls of lead-glass shower counters, called quadrants, which surround the interaction point. Each quadrant consists of five layers of glass separated by proportional chambers. The ASP trigger, with several thresholds and programmable logic, provides simple and general logic for triggering on both localized and overall energy deposit.

The flow of the trigger decision is shown schematically in Fig. 1. The 632 photomultiplier signals from the lead glass are each split to a digitizing system and to the trigger. The trigger signals are summed to eighty sums of eight and then summed again to twenty sums, each corresponding to a layer. These twenty layer sums (five per quadrant) each go to integration circuits and then are discriminated to define hit layers.

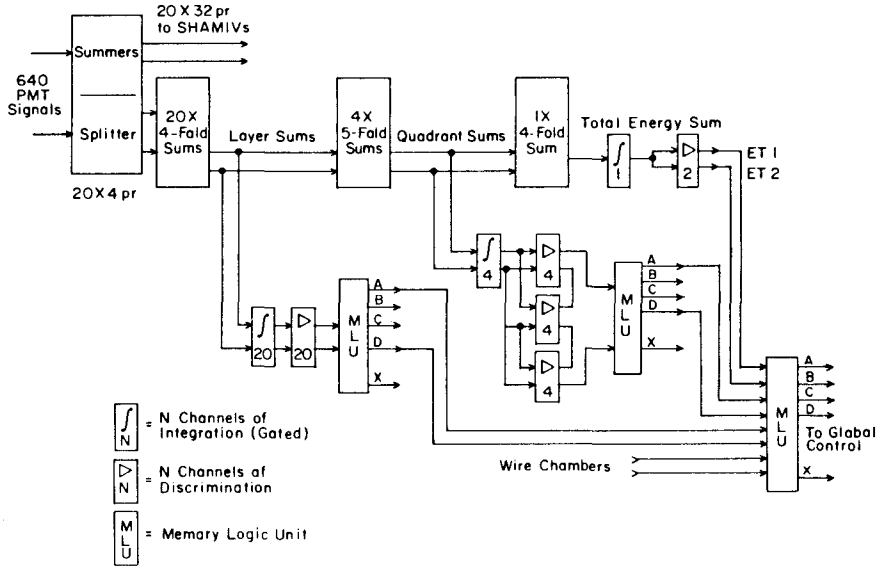


Fig. 1. ASP Energy Trigger block diagram.

The hit layers address a memory look-up which in turn defines allowed combinations of hits; for example, the first two layers of a quadrant but not the last two. The layer sums are also summed in turn into four quadrant sums, which are integrated and each discriminated against three levels defining three energy thresholds for deposit within a quadrant. The resulting twelve signals address a memory look-up that counts and defines combinations of quadrant hits. The quadrant sums are also summed to form a total energy sum, which is also integrated and discriminated against two thresholds. The resulting two total energy sum signals, four-bit combinations of quadrant sums, and four-bit combinations of layer sums, along with signals from PWC's and low-angle shower counters, address a final memory look-up which forms a four-bit output used by the global control module, which issues the trigger interrupt to the data acquisition computer and controls the overall system timing.

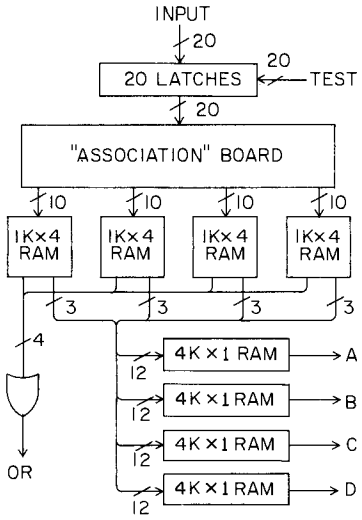


Fig. 2. Memory Logic Unit (MLU) block diagram.

The memory look-up is performed by Memory Logic Units (MLU's) with twenty inputs which address RAM's to provide five outputs, as shown in Fig. 2. To reduce the size of the RAM needed, the latched information from the twenty inputs is passed to an "association" board with forty outputs. This association board is simply a plug-in piggyback card which allows the twenty inputs to be patched in arbitrary fashion to four sets of ten outputs which address four $1K \times 4$-bit RAM's on the main board. One output from each of these RAM's is ORed with a bit from each of the others to

produce an OR output from the MLU. The remaining three bits from all four RAM's are combined to address each of four $4K \times 1$-bit RAM's in parallel. These RAM's provide four additional MLU outputs. The RAM's are all read/write, and test inputs to the MLU exist for diagnostics.

2.2 The Mark III Charged Particle Trigger

The Mark III charged particle trigger is another example of a fast trigger decision that could be made between beam crossings. In fact, since the Mark III drift chamber has maximum drift times longer than the available decision time at SPEAR, the Mark III trigger has multilevels. The first-level decision is made between crossings, the two second-level decisions are made before the second subsequent crossing, and the third-level is made in the following 100 μsec. Only levels 1 and 2b are described here. All levels of the charged particle trigger are described in Ref. 3.

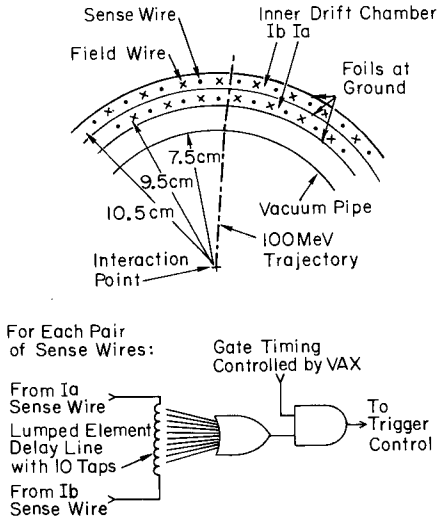


Fig. 3. Mark III Inner Drift Chamber, showing delay line logic for Level 1 trigger.

The Level 1 trigger utilizes the inner drift chamber of the Mark III. This one-meter-long chamber, located 10 cm from the interaction point, has two overlapping layers with 1-cm drift, as shown in Fig. 3. The Level 1 decision is based on the fact that the sum of the drift times in the two overlapping layers is a constant for tracks originating at the interaction point. It uses a chronotron composed of a lumped

element delay line with 10 taps to define a hit. Level 1 reduces the rate to about 300 Hz on the ψ resonance, by restricting the longitudinal acceptance to the length of the chamber and by restricting the radial acceptance and time acceptance using the chronotron. No scintillation counters are used; however, time-of-flight scintillators can also be incorporated if desired. A similar technique is used by the TPC experiment[4] as a first-level trigger.

The Level 2b trigger is based upon fast circle finding using programmed logic arrays. Three drift chamber layers, with radii at 10, 40, and 65 cm, are used as shown in Fig. 4. Since each of the outer layers consists of three sense wires among which only two hits are required, the track-finding efficiency remains at about 95% . The logic searches in parallel for tracks through any of the eighty cells in the outer layer. Through any one outer cell and the interaction point, there exist less than sixteen possible trajectories with momentum greater than 50 MeV. A PAL is associated with each outer cell. Its inputs are that cell, the cells which lie on possible trajectories through the other two layers, and control lines to select a momentum cutoff. The PAL is programmed as an OR of up to 16 AND's such that satisfying any possible trajectory identifies a track. Demanding two or more tracks reduces the trigger rate on the ψ resonance to 3.5 Hz, composed roughly equally of physics, cosmics, and beam-gas. Level 2b decision time requires 25 nsec after the maximum drift time. Searching 80 sets of conditions in parallel provides this speed. Having only 80 sets of conditions by using only three layers to define a trajectory allows the parallel search for tracks. Although this approach has worked effectively, it is potentially susceptible to inefficiency or noise should a chamber layer not operate correctly.

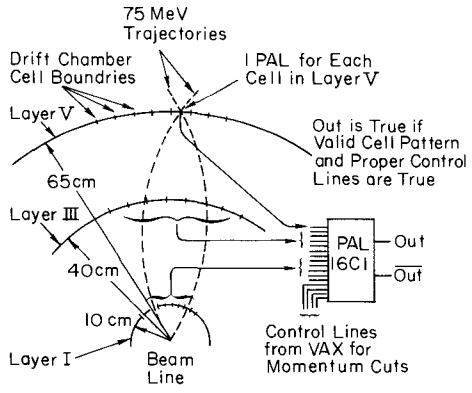


Fig. 4. Mark III Drift Chamber Layers I, III, and V, showing circle combinations and PAL logic for Level 2b trigger.

2.3 THE MARK II CHARGED PARTICLE TRIGGER

The Mark II charged particle secondary trigger, described in detail in Ref. 5, is an example of a second-level trigger processor which utilizes more complete drift chamber information in a flexible manner. The HRS Curvature Processor[6] is essentially identical, with some minor extensions. The DELCO secondary trigger[7] is also somewhat similar except that only one road pattern is defined. The Mark II processor reads out the cells serially in each of twelve drift chamber layers. This serial readout, which is via shift registers, translates the polar angle of a hit wire into a time t. By shifting twelve layers in parallel at a constant angular velocity, a straight track at an angle ϕ_i gives a coincidence among layers at readout time t_i. Curved tracks are found by appropriately varying the relative delay of the readout of the various layers. This process effectively rotates a set of curved masks through azimuth (see Fig. 5), searching for tracks which match a mask.

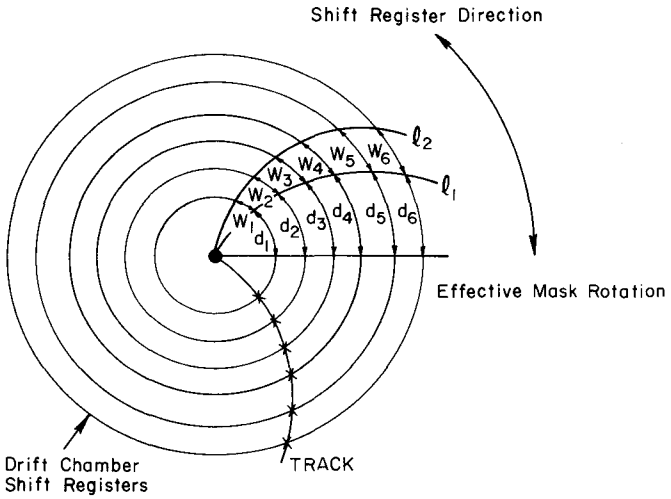


Fig. 5. Mark II Charged Track Finding Principle.

The trigger processor is shown schematically in Fig. 6. Hits in the drift chamber cells are recorded by time-to-amplitude or time-to-digital converters and registered in shift registers in those modules. The shift registers of twelve layers of cells are simultaneously shifted under the control of the Master Clock through a Test and Pickoff module, which places the shift register output onto the auxiliary bus of three CAMAC crates. In these crates, 24 Curvature Modules, operating in parallel, search the data

on the bus for tracks in 24 different curvature ranges. A complete set of curvature "masks" is shown in Fig. 7. Identified tracks of three types are signalled to three Track Counter modules which count tracks and record their angles and curvatures. At the end of the process, the track counts are sent to the master interrupt controller where the trigger decision is made.

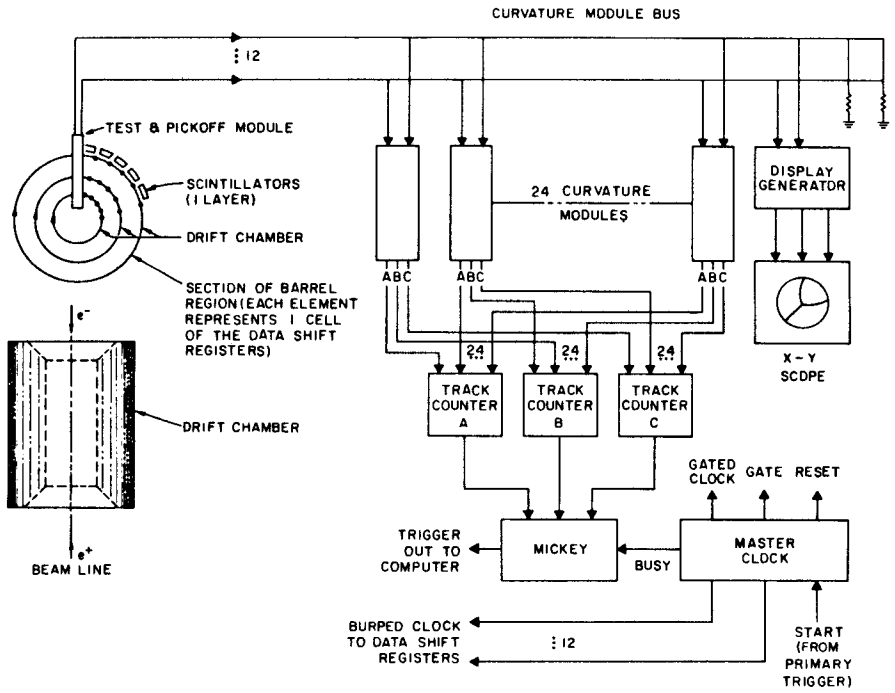


Fig. 6. Mark II Charged Particle Trigger Processor block diagram.

The Test and Pickoff module shifts circularly through 490° in order to find tracks of all curvatures and of both signs at the $\phi = 0$ boundary. It also allows the injection of patterns to test the integrity of the shift registers and of the rest of the logic. The Curvature Modules contain programmable logic to define a momentum bite (curvature) and a road width. Hits in the twelve layers address a $4K \times 2$-bit RAM which is programmed to identify three exclusive track types (seven types in the HRS logic). The programmed parameters are chosen for optimal efficiency using off-line simulation

of the hardware interfaced to a physics Monte Carlo. Efficiency is measured using real tracks. The Master Clock module provides twelve separate "burped" clocks for the twelve layers. All clocks are based on the same frequency of 10 MHz; however, clock transitions are periodically skipped for each layer in order to shift all layers at a constant angular velocity. Twelve such clocks are necessary because each drift chamber layer has a different number of cells. The Master Clock is programmable to allow choice of the twelve layers used in the trigger. The Track Counter modules contain logic to avoid double counting of tracks from the same or different Curvature Modules. They also permit CAMAC readout of information about tracks found. This information is used for diagnostics and to aid track reconstruction. In addition, the trigger processor includes a Display Generator, which serves as an invaluable diagnostic tool, and a Colinear Track Finder, which identifies back-to-back track combinations such as small-angle Bhabha events. The master interrupt controller MICKEY manages the multi-level decision process, including gates, resets, and interrupts. It also has memory look-up to trigger on programmable combinations of signals from various trigger processors, including the count of each track type, a count of hit calorimeter modules, and an overall energy threshold. A complete package of diagnostics perform routine tests of the entire processor and identify failing modules.

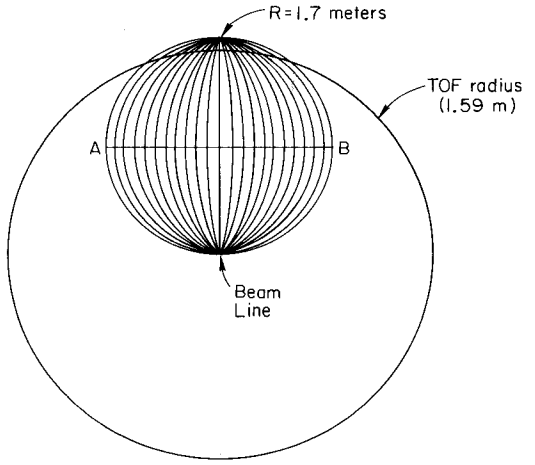


Fig. 7. Mark II Curvature Masks for a given azimuth

The charged-particle trigger decision using this processor requires 35 μsec. A compromise between serial and parallel processing is achieved which economizes on the number of connections and on speed. Much drift chamber information is available to the processor, permitting tight roads pointing to the origin which are effective at

rejecting background but efficient for finding tracks. Programmability has enabled the processor to function effectively at SPEAR and at PEP with various inner drift chambers and despite the inefficiencies of an aging central drift chamber. This processor will also be used with a new drift chamber at PEP and at SLC.

3. TRIGGERING AT THE SLC

The unique features of the SLC with respect to trigger processing are the long, 5.5-*msec* interval between beam crossings, the high luminosity without high beam current, and the high event rate provided by the Z^0 resonance. The long interval between beam crossings allows complex, hierarchal trigger processing, including software processing. Such triggers would allow maximum flexibility through programmability, use existing data paths, and allow uniform treatment of all detector subsystems. The lower beam-crossing rate also limits background. Beam-gas rates, with similar vacuums and numbers of electrons per bunch at PEP and SLC, are reduced by more than 10^3 at SLC by the lower crossing rate. Cosmic rates are similarly reduced. Synchrotron radiation, beamstrahlung, and radiation from the beam dump can all be masked, except very near the beam-pipe; however, synchrotron radiation could be a surprise. It could slow trigger processing times and data acquisition times, as well as increasing the background rate. Ability to recognize and reject synchrotron radiation hits could prove to be important.

3.1 THE MARK II TRIGGER AT SLC

Trigger processing for the Mark II detector at the SLC is shaped also by the need to trigger the upgraded detector during checkout at PEP. Consequently, the existing trigger processor (see Section 2.3) will continue to be used. Information from the new central drift chamber will be processed first by trigger cards which interface FASTBUS TDC modules[8] to the existing trigger. These trigger cards connect and are addressed through the FASTBUS auxiliary connector. They latch hits on and do majority logic on the six wires in each drift chamber cell. The results of the majority logic, performed by addressing programmable RAM for all 972 cells in parallel, are then shifted through the existing logic. The ability to do majority logic at the cell level will allow a tighter majority requirement at the road level, which could help reject random background hits. Other new detector elements will be incorporated into the trigger in the fashion of the elements which they replace.

The Mark II at SLC will also be prepared to do software trigger processing if required by severe backgrounds. Synchrotron radiation hits could be rejected, based on pulse height, before shifting drift chamber hits through the existing logic. Beam-gas tracks could be rejected by determining the event vertex position by fitting tracks

found by the hardware trigger processor. Such processing would be performed by the SLAC Scanner Processors (SSP's; see Section 4.2) and 3081/E processors (see Section 4.4) which are part of the data acquisition system. The 3081/E's will have available to them the entire event record for trigger considerations.

3.2 THE SLD TRIGGER

Trigger processing for the SLD detector[10] will use data paths similar to the event acquisition paths. Input data for the trigger will be derived in the data acquisition modules. Chamber data, acquired in Waveform Sampling Modules, will be reduced to one bit per wire. Calorimeter data, acquired in Calorimetry Data Modules, will be the digitized energies. This trigger input data will be read out by the same crate-level SLAC Scanner Processors (SSP's; see Section 4.2) used in event acquisition. At the crate level the trigger data will be tested for evidence of tracks and further compressed. For instance, five out of eight drift chamber wires hit in a cell will define a hit cell, and one bit per cell will indicate whether a cell was hit. For calorimeters, the digitizations within a tower will be checked for consistency and compressed to one bit per tower and to a total energy per tower. The crate-level SSP's will send the compressed trigger data to dedicated Trigger Processors. All Trigger Processors and their programs will be identical. The program will perform pattern recognition by table look-up and will result in simple numeric descriptions of recognized patterns. Pattern recognition will be done by separate Trigger Processors for each of the three stereo views of the drift chamber and for each of four calorimeter units. SSP's may serve as these processors. Finally, the recognized patterns will be read by a Trigger Master which makes the final trigger decision. The Trigger Master will also be the event acquisition master, controlling detector resets, waits, etc., data flow, and the running environment. It may be an SSP or a FASTBUS VAX (see Section 4.4). The trigger decision will be complete in time to reset the detector before the next beam crossing.

4. ON–LINE PROCESSING AT SLAC

Large, general-purpose e^+e^- detectors require on-line processors for data acquisition, calibration, monitor, and diagnostic tasks. In order to reduce readout times and to simplify downstream processing, event and calibration data is typically compressed by the data acquisition modules or by crate-level processors such as the BADC and the SSP. Some monitoring and control may be provided by microprocessors, such as the SFC. Thorough monitoring and diagnostics, which frequently involve studying sampled events, sometimes after complete event reconstruction, is usually done by host computers, which at SLAC are VAX's. The host generally writes the data to tape for transfer to off-line processing; however, the MAC experiment at PEP transfers data directly

from disk on their VAX to disk on the SLAC IBM facility via a Long Line Adapter. For the complex SLC detectors, processors such as the 3081/E or the FASTBUS VAX will supplement the hosts at the system level in monitor and diagnostic tasks. Microcomputers will manage the flow of data among the on-line processors. The on-line processing systems for the Mark II and SLD detectors at SLC are described in Refs. 9 and 10, respectively. The following sections describe some of the on-line processors commonly used at SLAC.

4.1 THE BADC

The BADC[11] is a microprocessor-based semi-autonomous controller for CAMAC. It was designed to control readout of a CAMAC crate of data acquisition modules, such as sample-and-hold and time-to-amplitude converters. It controls the multiplexing of data onto an analog bus, digitizes the analog data, compresses and processes the data, and buffers results until CAMAC readout. The algorithm most often used for processing of event data discards data below some threshold, corrects data by a quadratic polynomial, and relabels data by function, such as by drift chamber layer number and wire number. Threshold and correction constants for each channel and labeling information are stored in local RAM. This algorithm requires about 3 μsec per channel for data below threshold and 10 μsec per channel for data above threshold. Another algorithm streamlines calibration procedures by calculating an updated mean and variance for each channel after each event and thereby reducing the amount of CAMAC readout and of host computation. Certain diagnostic algorithms are also implemented.

The BADC economizes on cost, by amortizing the cost of digitizing hardware over hundreds of channels and by allowing higher channel densities. It economizes on readout time, by allowing several crates of electronics to be sparse scanned in parallel followed by block transfer from a small number of BADC's. It economizes on host processing time by correcting and relabeling data at high execution speeds and in parallel, and it simplifies host program structure by handling many thousands of constants.

The architecture of the BADC is described in Ref. 11. Its features are only highlighted here in order to illuminate how it fits into a data acquisition system. The BADC is a triple-width CAMAC module with CPU, RAM, and ADC boards. The ALU is four AMD 2901 four-bit slices controlled by an AMD 2909 microprogram sequencer. Program memory is two 256-word pages of 48-bit PROM. In practice, the second page has never been used in an application. Memory is either $4K$ or $12K$ 16-bit RAM with 220-$nsec$ read access. The RAM is used to contain control tables, correction constants, and data buffer. Three clock cycles are defined; short (200 $nsec$) for most operations, long (360 $nsec$) for conditional branches, and pause for operations such as CAMAC I/O

which require acknowledgement from another execution unit of the BADC. Interrupts are not implemented. CAMAC commands to the BADC, as well as front-panel signals, force branches to predefined locations in PROM.

The BADC addresses modules within a crate by transmitting an encoded station number N to a SLAC type-U crate controller. The BADC directly accesses the F, A, S1, and S2 lines. Handling of contention on the CAMAC lines is limited. If contention occurs during BADC execution, an error breakpoint is set. A software timer allows a CAMAC cycle which starts BADC execution to complete before the BADC commences CAMAC operation. Control of CAMAC scans is provided by the ADC board which receives the analog data into a three-step pipeline, consisting of address data module, sample-and-hold, and digitize. All CAMAC data is routed through the RAM board, where the D and Y busses of the ALU are interfaced by buffers to the CAMAC W and R Lines. The output buffer from Y to R is gated by a READ from CAMAC. The input buffer from W to D is loaded by a WRITE from CAMAC and gated onto the D bus by microcode which moves data from the buffer, onto the D bus, through the ALU, onto the Y bus, and into RAM in a single CAMAC cycle. Interconnection of the RAM and ADC boards to the CPU and CAMAC are shown in Fig. 8. The modular construction of the BADC has allowed easy modification of the basic design, for instance for $12K$ RAM boards and for replacement of ADC boards by interfaces for alternate hardware.

Microcode for the BADC is now generated using a machine-independent meta-assembler written in FORTRAN named MAMIC. MAMIC consists of two passes: the first defining the machine and instructions in terms of microcode fields and operations, and the second assembling the user code. It was also used for the SLAC Scanner Processor microcode PROM's and microcode for various other devices such as the VAX CAMAC Channel. In the case of the BADC, the generated microcode can be tested using a debugging RAM in a separate CAMAC module before burning PROM's.

Since its first use in 1977 with the Mark II detector at SPEAR, the BADC has been a central element in the data acquisition systems of many SLAC experiments. Seven different SLAC detectors[12] have used more than fifty of these units (which are now commercially available[13]) for a variety of detector types: MWPC, drift, lead glass, and liquid argon shower counters, drift chambers, MWPC cathode and charge division readouts, time-of-flight counters, muon detectors, and luminosity monitors. Associated data acquisition modules developed for use with BADC's include: four types of sample-and-hold circuits, single-hit and multi-hit time-to-analog converters, and a time-of-flight module containing time and pulse-height measurements. In addition, BADC units have been adapted to alternate hardware configurations, including readout of a second crate, a remote crate, and separate hardware. In the Mark II experiment, BADC's have performed the readout of every detector component except the proportional tubes of the muon system. At SLC, the Mark II will use twenty-two BADC's; however, SLAC Scanner Processors will be used for readout of FASTBUS electronics for the new drift chambers.

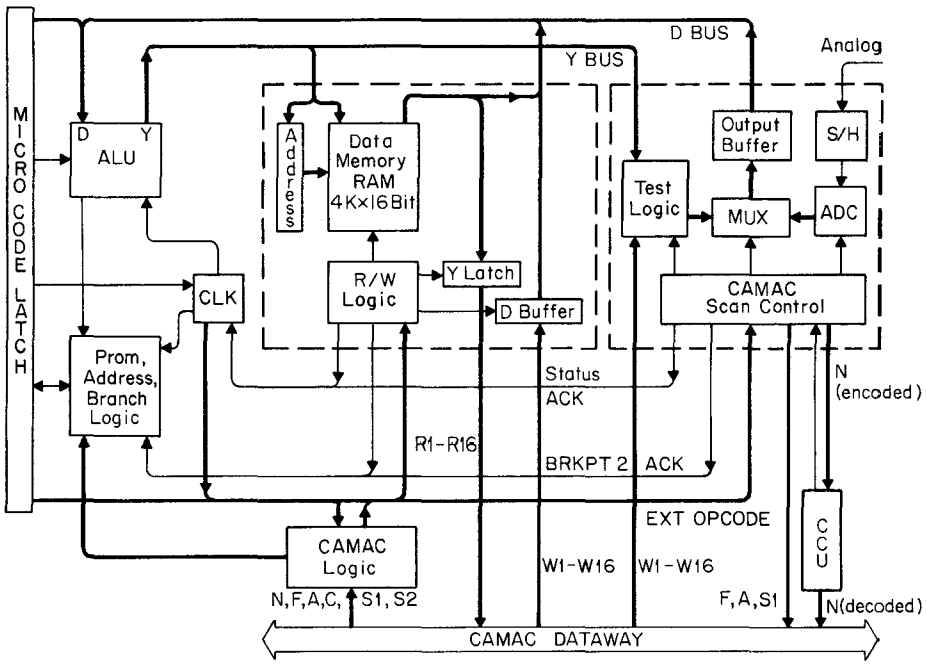


Fig. 8. BADC block diagram.

4.2 The SLAC Scanner Processor

The SLAC Scanner Processor (or SSP), which is described more completely in Ref. 14, is a general-purpose, high-speed, programmable FASTBUS module. It has been designed for the Mark II Upgrade to provide crate-level processing of data from FASTBUS modules similar to that provided in CAMAC by BADC's. The SSP, however, provides a more general, and, hence, more powerful means of moving and processing data in a FASTBUS system. Consequently, SSP's can also be used for various system-level tasks.

From the FASTBUS point of view the SSP can be attached as either a master or a slave to either a crate segment or a cable segment, while being physically connected to both. As a slave, SSP memories can be read and written. The program memory of the SSP maps into CSR space, and data memory maps both into data space and CSR space. Host control of SSP operation as a master is exercised through CSR# 0.

As a slave, the SSP operates without CPU involvement. DS-to-DK response times are about 100 *nsec*.

As a master, the SSP implements a large number of FASTBUS operations. FASTBUS primitives, such as primary address cycle, are implemented as single SSP instructions which include a pointer to an argument list. Other FASTBUS operations, such a read data block, are implemented as single SSP instructions which execute a sequence of FASTBUS primitives at the microcode level. Finally, other FASTBUS operations, such as write random data with pattern select, are implemented as macros of SSP instructions at Assembler level. Instructions which terminate a block transfer normally upon an end-of-block response (SS=2) from a slave and instructions which terminate block transfers upon exhausting a word count are both implemented. Block transfers involve a DK-to-DS delay of about 100 *nsec*. All IBM integer and many byte instructions are implemented for control and data processing.

The SSP consists of two FASTBUS boards, control and processor. The control board contains the interface and I/O logic, which are symmetric with respect to the crate and cable segments, instruction logic, address calculation logic, branch logic, and word count logic. The instruction logic consists of $4K$ words of program memory, PROM address logic, and microcode PROM's in a three-level instruction pipeline. Instructions have IBM format. They are decoded in the microcode PROM's to 56 bits which control the instruction logic and I/O logic and to 48 bits which control the processor. As many as 256 instructions can be defined in the microcode proms. The processor board contains a 32-bit CPU (eight AMD 2901C bit-slices), dedicated input and output shifters to expedite multiple shifts, $32K \times 32$-bit words of data memory, two registers, and condition code logic. The data memory is byte-addressed. It is composed of $16K$ static RAM's and will be easily expandable when $64K$ RAM's become available.

The SSP is designed to be easily programmed. Source code can be written in IBM Assembler or in FORTRAN. Code is then assembled or compiled, translated into separate program and data, optimized, and linked into an image by a translator/linker program on a host computer. This program produces a single file comprised of program and data memory images and of a header which contains information on program size, locations of COMMON blocks, etc. The program and data memories are down-loaded to the SSP over FASTBUS. The initial program status word is loaded into data memory location 0, and execution is started via CSR# 0 or by a hardware signal. Completion of execution can be signalled by a service request or by a front panel output. Execution can be interrupted between instructions if the SSP is so enabled. Instructions can be executed in single-step mode, and a cross-debugger can be implemented via FASTBUS.

The initial application of the SSP is in the drift chamber readout system for the Mark II, where SSP's will scan, process, and buffer data from commercial TDC modules[8] and from SLAC-designed FADC modules. At the crate level, these SSP's

will also process calibration data and perform crate initialization, data acquisition module testing, self-testing, and crate segment verification. In addition, SSP's will serve at the system level as cable segment masters and buffers for the cable segments linking the TDC and FADC crates and as data block movers and buffers to and from on-line 3081/E processors. More than 25 SSP's will be used by the Mark II. The SLD detector also plans to use SSP's at the crate level for all detector subsystems ($\sim$ 80 units) to readout and process data from Waveform Sampling Modules and Calorimeter Digitization Modules. The crate-level SSP's also preprocess data for the event trigger, and additional SSP's act as Trigger Processors for each subsystem and as a Trigger Master to complete the trigger decision.

4.3 THE SLAC FASTBUS CONTROLLER

The SLAC FASTBUS Controller (or SFC), which is described more completely in Ref. 15, is a single-card FASTBUS microcomputer suitable for real-time monitoring and control applications. In fact, an SFC can transfer blocks of data at rates of about 5 μsec per 32-bit word (800 KB/sec), which is faster than effective UNIBUS rates (500 KB/sec) and within a factor of about two of fastest CAMAC speeds. It can also implement all possible FASTBUS operations as both a master and a slave and can serve as a host for a FASTBUS system. Moreover, it can be programmed in higher-level languages such as FORTRAN.

The SFC consists of an interface between FASTBUS and MULTIBUS and of any MULTIBUS single-board computer, such as Motorola 68000, Intel 8086, or National 16032, which mounts in the FASTBUS module. The interface connects the 8- or 16-bit MULTIBUS data bus to the FASTBUS AD lines and maps an 8-bit MULTIBUS I/O address onto FASTBUS AS, DS, RD, MS lines. As a master, the SFC executes a FASTBUS cycle by performing a read/write to MULTIBUS I/O space. Processors, such as the 68000 and 16032, which can move a 32-bit longword on the data bus with one instruction are able to execute a FASTBUS cycle, either address or data, in a single instruction. The interface hardware is designed to reduce software overhead by arbitrating for the FASTBUS and by issuing the strobes AS and DS and waiting for acknowledge signals AK and DK while managing a timeout counter, checking parity, and checking for non-zero SS responses. The interface signals cycle completion with the XACK* signal and indicates a FASTBUS error with BERR, thus eliminating software status checking. Interrupts from FASTBUS, such as incoming service request, mastership granted, or selected as slave, are signalled to the processor via GINTR. Three different modes of mastership are supported to enhance execution speed. All FASTBUS operations are supported.

As a slave, the SFC responds in hardware to all types of address cycles and emulates data cycles in software. When the SFC is attached as a slave, the processor is

interrupted by GINTR. Then it polls I/O space for DS, reads command bits (RD, MS), and branches to the appropriate command handler which responds to the command and causes the interface to produce DK and SS responses. During the DS-DK interval, the interface normally asserts WT to avoid timeouts. After response is complete, the processor returns to polling DS until the AS-AK lock to the SFC is removed.

As implemented for monitor and control of the Mark II cryogenic system, the SFC is equipped with an Intel iSBC 86/30[16] (8086, 8087 Numeric Data Processor, 256K RAM, and 64K EPROM). FASTBUS standard routines are implemented in ROM, along with a boot program which allows the SFC to be addressed via FASTBUS. Since this SFC uses the same SBC as used in the SLC control system (see Section 5), it can also use the sophisticated software tools developed for the SLC. Applications code can be written in FORTRAN and cross-compiled and linked on a VAX. Code can be downloaded and started via MULTIBUS or FASTBUS; however, the symbolic cross-debugger currently runs only over SLCNET-MULTIBUS. An SFC with a 68000-based SBC has also been implemented at SLAC, and several units have been delivered for implementation outside SLAC.

The SFC is capable of serving as host for a FASTBUS system since it is capable of loading its own logical address and arbitration level registers and of asserting RB and GK to preempt a segment. Additional features include self-diagnostic capability and board area available for wire wrap, for instance, of a sequencer. MULTIBUS is available on the FASTBUS auxiliary connector and to plug-in modules for some expansion and peripherals.

4.4 The 3081/E Processors and FASTBUS VAXes

The 3081/E and FASTBUS VAXes have been described in Refs. 17 and 18 and discussed at this conference.[19,20]

Implemented on line, the 3081/E is a powerful (4 to 5 VAX 11/780 equivalents) processor for applications such as data preprocessing, software triggering, and event reconstruction. It is capable of executing on line the same code used off line for event reconstruction on IBM mainframes. Mark II plans to use 3081/E processors interfaced to FASTBUS through a quasi-dual-ported FASTBUS slave interface. These processors will preprocess drift chamber data from TDC and FADC systems and fully reconstruct events for on-line processing. They will also be used for an on-line software trigger if necessary.

The FASTBUS VAX, equivalent to about 90% of a VAX 11/780 in CPU, offers the easy programmability of VMS, software compatibility with the on-line host computer, and in situ debugging. The SLD plans to use these microcomputers as a master trigger

processor and to separately tag physics and background events with a fast filtering algorithm. In addition, they will be used as stand-alone microcomputers for development work.

4.5 The 168/E Processor

The 168/E processor has been described and discussed comprehensively in Ref. 21. These processors have been used at SLAC for off-line event reconstruction by LASS[22] and DELCO and for lattice gauge theory calculations.[23] On line at SLAC they have been used by the SLAC Hybrid Facility (SHF) to provide camera triggers and for monitoring.[24] At SHF, the processors could obtain their data by listening passively to data transfers on a CAMAC backplane.[25] In addition, circuits whose results were sufficiently time critical for triggering were built into a 168/E, where they interacted with the processor as locations in data memory. Data input from PWC digitizers and space point finders were implemented as read-only memories. Camera triggers were provided by accessing memory addresses on other 168/E boards.

4.6 The VAX CAMAC Channel

The VAX CAMAC Channel (or VCC), described in Ref. 26, is a UNIBUS to CAMAC interface utilizing a bipolar microprocessor to supervise the CAMAC system and minimize the host computer's work. The high speed 16-bit CPU is composed of AMD 2900 bit-slice bipolar microprocessor components. The interfaces to a DEC UNIBUS and to CAMAC are peripherals to the CPU linked by input and output data busses, a status bus, and a microcode bus for interface control. The CAMAC interface connects, via SLAC CAMAC protocol, to a set of system crates which house Branch Drivers for parallel or serial branches or for other data links. A VCC operation is initiated by a software device driver which passes the VCC the addresses of control and data buffers. No further VAX CPU activity is required until the VCC has completed the list of data transfers implied by the control "channel program" buffer. The VCC fetches commands from VAX main memory, sets up the Branch Drivers and transfers data between the CAMAC modules and VAX main memory at rates limited by the UNIBUS. Address scanning for block transfers is handled by the Branch Drivers. Data acceptance, conditioned by X and Q responses, and data packing into 16-bit words are performed by the VCC. The VCC also serves as a channel to the VAX for hardware interrupts. A coherent software package integrates high-level programs with system driver-level programs and microcode control of the system.

5. SLC CONTROL SYSTEM

The SLC control system[27] will be an order of magnitude more complex than SLAC's other accelerator control systems. In addition to modernizing and streamlining the operation of the present linac/beam switchyard system, the SLC control system must provide machine modeling to support the extensive accelerator development required to meet the tight SLC beam requirements. In its final configuration, the SLC control system (see Fig. 9) will provide a combination of two VAX 11/780 central processors networked to 70-100 powerful microprocessor clusters which interface through CAMAC with the equipment to be monitored and controlled.

The dual-VAX complex provides a centralized human interface for the machine operators, as well as on-line execution of the large modeling programs. In addition, these computers provide an environment for fast, efficient program development and maintenance for both the VAX and the micro-processor clusters.

The distributed micro-processor clusters are located in each of the 30 linac sector alcoves and near the damping ring, electron and positron sources, and the SLC arcs and final focus. The clusters are based on the Intel Multibus architecture, which provides support for an arbitrary number of single-board computers which communicate with each other through the use of shared memory and interrupts. The micro-processor clusters contain an Intel iSBC 86/30[16] with 8087 coprocessor, 786 Kilobytes of RAM, and 8 kilobytes of EPROM, providing about 1/7 of the processing power of the VAX 11/780. The microprocessor clusters interface to a 5 MHz serial CAMAC branch through a DMA channel device.

Intelligence is also distributed into the CAMAC crates via the use of dedicated microprocessors in some of the data acquisition modules. The Smart Analog Monitor (SAM) is a Zilog Z80-based CAMAC module that continuously scans 32 analog channels, auto-calibrates, auto-ranges, digitally filters, and provides floating point voltage values in either VMS or IEEE formats. The Parallel I/O Processor CAMAC module (PIOP) is a general-purpose processor based on the Intel 8088. It interfaces to CAMAC and to a front panel port which is a differential transmitter/receiver version of the micro-processor's bus structure. This port provides a standardized method of interfacing to specific devices or processes. For instance, PIOP's monitor the phase and amplitude of the 240 linac klystrons, as well as provide general klystron monitor and control. Code for the PIOP is cross-compiled or cross-assembled on the VAX, then downloaded via CAMAC to the PIOP or "burned" into EPROM.

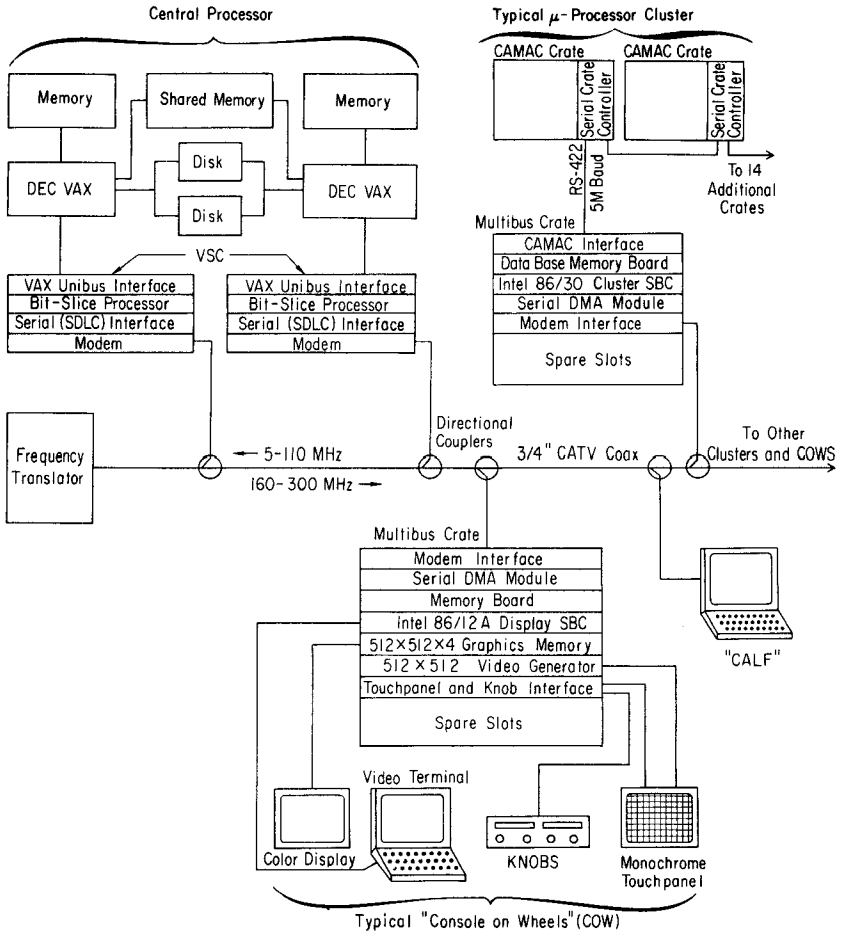


Fig. 9. SLC Control System block diagram.

There are two types of operator consoles in the system. The Console-On-Wheels (COW) consists of a micro-processor cluster, high-resolution color graphics display, touch panel, general-purpose knobs, computer terminal, video monitor, and audio intercom; yet the COW is a fully portable unit which can be connected at any point along the system's Communication Line. The Curser-Addressed Limited Facility (CALF) emulates a subset of COW functions and can be also connected at any point along the

Communications Line. The number of COW's and CALF's that can be supported is limited only by the system's processing power.

The Communications Line for the system consists of a broadband (5 - 300 MHz) cable television (CATV) system. Communications are organized in an inverted-tree topology with mid-split repeaters. Low frequencies feed from the source to the up-converters, and high frequencies feed to all receivers. Several sub-systems use the cable for communications. The micro-processor clusters and the VAX's are interconnected by a one Megabaud polled network. A bit-slice processor directs sequential polling at a rate of about 1000 polls/second and serves as a DMA channel to the VAX. The logical topology is a star network with communication only between the host and the microclusters.

A significant effort has been expended to create an efficient and user-friendly environment for the development of micro-processor software. All SLC software development is performed on the VAX. Wherever possible, which has been in about 95% of cases, FORTRAN 77 is used for applications programming of the micro-clusters. (VMS FORTRAN is used for the VAX's.) In collaboration with Intel, FORTRAN 77 and PLM 86 cross-compilers, a cross-assembler, and a cross-linker have been developed to support the 8086/8088 series of micro-processors. Further, a symbolic cross-debugger has been developed to allow the remote debugging of micro-processor programs running under iRMX. In this complex system, on-line debugging and diagnostic aids have been essential in order to trace problems efficiently in the system's operating environment.

6. SUMMARY

Trigger and on-line processors share the role of reducing data rates to manageable proportions in large, general purpose e^+e^- physics experiments. This paper has attempted to review a vast amount of work on trigger and on-line processors at SLAC.

Generic solutions to trigger processing have evolved at SPEAR and at PEP, although special-purpose hardware trigger processors have been developed for each experiment. Some of these processors, on the other hand, have a great deal of flexibility and programmability. The environment at the SLC will offer the possibility of sophisticated, software triggers.

On-line processor development at SLAC has focused on data reduction and preprocessing at the data acquisition crate level. The BADC and the SLAC Scanner Processor are examples of such processors. At the SLC, more powerful processors, such as the 3081/E and the FASTBUS VAX, will be used for more sophisticated preprocessing, software triggering, and event sampling and flagging. The SHF use of the 168/E pioneered on-line use of such powerful processors. In addition, general-purpose microprocessors, such as the SLAC FASTBUS Controller and the Parallel I/O Processor,

are assuming roles in experiment and accelerator control. At all levels, a high premium has been placed upon uniformity of approach, for both hardware and software, leading to the development of a small number of flexible yet powerful processors and sophisticated software tools.

REFERENCES

1. G. M. Ronan, et.al., "Triggering the LBL Time Projection Chamber," IEEE Trans. Nucl. Sci., Vol. NS-29, No. 1 (1982).
2. R. Hollebeek, 3rd International Conference on Instrumentation for Colliding Beam Physics, SLAC-PUB-3347.
3. J. J. Thaler, et.al., "Event Trigger for the Mark-III Detector at SPEAR," IEEE Trans. Nucl. Sci., Vol. NS-30, No. 1 (1983).
4. H. Ailiara, et.al., "Performance of a Drift Chamber System for the Time Projection Chamber Detector Facility at PEP," IEEE Trans. Nucl. Sci., Vol. NS-30, No. 1 (1983).
5. H. Brafman, et.al., "Fast Track-Finding Trigger Processor for the SLC/LBL Mark II Detector," IEEE Trans. Nucl. Sci., Vol. NS-25, No. 1 (1978).
6. Robert J. Wilson, "A Search for Scalar Electrons at PEP," Thesis, August 1983.
7. Dale Quimette, et.al., " A Versatile Secondary Trigger for a Multi-Detector System," IEEE Trans. Nucl. Sci., Vol. NS-30, No. 1 (1983).
8. LeCroy Model 1879, LeCroy Research Systems Corp., Spring Valley, New York.
9. A. J. Lankford and T. Glanzman, "Data Acquisition and FASTBUS for the Mark II Detector," IEEE Trans. Nucl. Sci., Vol. NS-31, No. 1 (1984).
10. SLD Design Report (Preliminary Edition), April 9, 1984.
11. M. Breidenbach, et.al., "Semi-Autonomous Controller for Data Acquisition, The Brilliant ADC," IEEE Trans. Nucl. Sci., Vol. NS-25, No. 1 (1978).
12. BADC's have been used by the following SLAC experiments (numbers of units shown in parentheses): LASS (6), Hybrid Facility (1), Free Quark Search (1), MAC (7), ASP (5), Mark III (15), and Mark II (20).
13. TRANSIAC Model 7001, TRANSIAC Corp., Mountain View, CA.
14. H. Brafman, et.al., "The SLAC Scanner Processor," submitted to IEEE 1984 Nuclear Science Symposium, Orlando, Florida, Oct. 31 - Nov. 2, 1984.
15. S. R. Deiss, "A Fastbus Controller Module Using a Multibus MPU," IEEE Trans. Nucl. Sci., Vol. NS-30, No. 1 (1983).
16. Intel Corp., Santa Clara, CA.

17. Paul F. Kunz, et.al., "The 3081/E Processor," published in the *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics*, Padova, Italy, March 23-25, Naz. Fis. Nucl. (1983).

18. E. J. Siskind, NYCB Real Time Computing, "Development of 32 Bit Single Board Computer Systems in Fastbus Packaging," DOE Small Business Innovation Research Program (Private Communication).

19. P. K. Kunz, et.al.,"The 3081/E Processor," published in these Proceedings.

20. E. J. Suskind, "VAX FASTBUS Module," published in these proceedings.

21. P. K. Kunz, "The LASS Hardware Processor," Nucl. Instrum. Methods 135, 435 (1976). P. K. Kunz, et.al., "The LASS Hardware Processor," 11th Annual Microprogramming Workshop, Pacific Grove, CA, Nov. 19-22, (1978). SIGMICRO Newsletter 9, 25 (1978). P. K. Kunz, "Use of Emulating Processors in High Energy Physics," *Proceedings of the International Conference on Experimentation at LEP*, Phys. Scr. 23, 492 (1981).

22. P. K. Kunz, et.al.,"Experience Using the 168/E Microprocessor for Off-Line Data Analysis, IEEE Trans. NS-27, 582 (1980).

23. J. E. Hirsch, et.al., "Monte Carlo Simulations of One-Dimensional Fermion Systems," NSF-ITP-82-44.

24. J. T. Carroll, et.al., "On-Line Experience with the 168/E," published in the *Proc. of the Topical Conference on the Application of Microprocessors High-Energy Physics Experiments*, CERN, Geneva, Switzerland, 4-6 May 1981.

25. D. Bernstein, et.al., "Snoop Module CAMAC Interface to the 168/E Microprocessor," IEEE Trans. Nucl. Sci., Vol. NS-27, No. 1 (1980).

26. D. J. Nelson, et.al., "The VAX CAMAC Channel," IEEE Trans. Nucl. Sci., Vol. NS-28, No. 1 (1981).

27. R. Melen, "A New Generation Control System at SLAC," IEEE Trans. Nucl. Sci., Vol. NS-28, No. 3 (1981). J. D. Fox, et.al., "Application of Local Area Networks to Accelerator Control Systems at the Stanford Linear Accelerator," IEEE Trans. Nucl. Sci., Vol. NS-30, No. 4 (1983). R. Melen, "Centralized Digital Control of Accelerators," invited talk to the Europhysics Conference, Computing in Accelerator Design and Operation, Berlin, West Germany, 20-23 September 1983.

QUESTIONS AND ANSWERS

Q: How fast is the ASP memory trigger, and ASP data readout?

H. Kasha

A: I am not fully qualified to give the answer. Fastest available memory chips have been used. The readout should be accomplished in about 3 msec.

Q: About the trigger at SLC; signal rate is expected to be ~1/4 Hz, negligible background, so why bother?

P. Lebrun

A: The trigger will be single (single track); but, even with such a low trigger, one has to choose between ~ 4 crossings due to data acquisition dead time. Also, the synchrotron radiation might be larger than expected.

Q: 1) How much development effort was required to emulate IBM Run-time Library in the $370/_E$?

2) If IBM sold you a 370 architecture microprocessor, would you use it?

E. Siskind

A: 1) 3/4 man-year for software.

2) Yes

A PARALLEL, PIPELINED, EVENT PROCESSOR FOR FERMILAB EXPERIMENT 605

D.M.Kaplan
Fermilab

ABSTRACT

I describe the E605 event processor, which represents the first use of the Nevis Laboratories processing system[1] in an experiment. The processor is constructed of simple general-purpose modules designed to operate on 16-bit data words synchronously and in parallel at speeds up to 40 MHz. It uses information from hodoscopes and wire chambers to reconstruct tracks in the bend view of a magnetic spectrometer, calculating an approximation to transverse momentum (p_t) for each track and mass for the most massive pair in an event. It also distinguishes tracks originating in the target from tracks originating in the beam dump. The results from the processor may be used to prescale events of low mass or p_t, and to simplify further analysis of the events by an on-line array processor or off-line computer. The processor was debugged during the Winter run of the Tevatron and is being utilized in the Spring run.

Introduction

Fermilab Experiment 605 is a spectrometer for the study of particles and pairs of particles produced at high transverse momentum (see Figure 1). Its open geometry permits simultaneous detection of leptons and hadrons, and its high degree of segmentation, sophisticated triggering scheme, and high-speed data acquisition system are designed to allow data taking at the very high beam intensities required to study phenomena representing 10^{-11} or less of the total proton-nucleon cross section. After a test run in the Spring of 1982 and a 400 GeV data run in the Fall of 1983, we are now taking data at 800 GeV and 10^{10} proton interactions per accelerator pulse (10^{9} interactions per second).

To enhance the power and flexibility of our triggering scheme we have constructed a parallel pipelined event processor to reconstruct particle tracks on-line. Using the granularity of the wire chambers, this processor is able to make more precise decisions than the first- and second-level triggers, which are based on hodoscope and calorimeter information, and it is faster and more powerful, though more specialized and less flexible, than the array processor attached to the on-line computer, which constitutes the final level of on-line filtering[2]. In addition to reconstructing tracks, the processor computes an approximation to transverse momentum (p_t) for each track and mass for the most

massive pair of tracks in each event. These can be used to reject or prescale events of low p_t or mass.

Originally the processor was conceived as a means of rejecting particles originating in the beam dump rather than in the target, but our goal has evolved into the distinguishing of target events from dump events, with different p_t and mass thresholds applicable in the two cases. This evolution was motivated by our realization that although muon pairs produced in the beam dump have worse mass resolution than those from the target by more than an order of magnitude, they have five times larger acceptance in the spectrometer and are accepted in a different region of phase space (large Feynman x), where interesting physics may be found.

2 General Description of Processing System

The processor is constructed using the Nevis Laboratories data-driven processing system[1], which consists of a set of processing modules and a standardized bus and protocol for interconnecting them. Each module implements some simple operation such as addition of two 16-bit quantities, comparison of a 16-bit quantity with upper and lower limits, or computing an arbitrary function of an 8-bit quantity via table lookup. The modules used in the E605 processor are listed in Figure 2, and Table 1 summarizes the processor bus protocol.

The modules are designed to operate with a cycle time of 25 ns. It takes typically two or three cycles for an input to propagate through to the output, but the modules are internally pipelined so that a new operation can begin every cycle. Data transfers between modules are also pipelined, so that after an initial period in which the processing pipelines are filling, all modules in the processor are operating simultaneously to the maximum extent possible. Data transfers among modules are synchronized to a central clock, which may be single-cycled, speeded up, slowed down, or run in bursts for diagnostic purposes, and every register and counter of every module may be read or written under computer control, facilitating thorough testing of the system.

Compared to hard-wired special-purpose processors, the system is quite flexible, since the modules may be recabled or the lookup tables reprogrammed to make changes in the algorithm. It also differs from typical hardware processors in that the algorithm is embodied in the interconnection of the modules rather than in control logic, so that no bottleneck arises when the system is expanded; as more modules are added, the total processing power increases proportionately.

The modules are constructed on 9"-square two-layer printed circuit boards. Most modules consist of a single board, containing typically 60 ECL 10,000 chips, and can be duplicated

for approximately $200 (an exception is the Map module, which takes two boards). Each module connects to one or two input processor busses and one output processor bus, as well as to power and control busses. The control bus is a simple bit-serial bus used for initialization, downloading of data, and testing.

3 Reconstruction Algorithm

Reconstruction is performed only in the y-z plane (the plane in which the magnets deflect), using the six wire chambers Y1A, Y1B, Y2, Y2', Y3, and Y3'. Y1A and Y1B are 2-mm-spacing proportional chambers, while Y2-Y2' and Y3-Y3' are pairs of drift chambers with cell widths of 1 and 2 cm (respectively), and with primed chambers offset by half of a cell with respect to unprimed chambers.

For all particles traversing the three stations of chambers, the positions measured by the chambers are linearly related:

$$Y1 = aY2 + bY3 + c \ .$$

The trackfinding loop forms all possible track hypotheses, consisting of a hit in station 2 and a hit in station 3, projects to station 1 using the above relation, and demands a hit in station 1 consistent with a particle originating in the target or in the upstream end of the beam dump.

The present version of the processor uses only wire numbers and ignores drift times, except to cut out hits whose drift times are too small (since with our drift chamber electronics, the first two time bins contain only hits from previous proton interactions). At typical beam intensities, this cut eliminates approximately 30% of drift chamber hits. With ~10 hits per plane, this algorithm is capable of finding all the tracks in an event in typically 5 microseconds or less. The efficiency of the algorithm depends on the efficiencies of the hodoscopes and chambers; since these are all well above 90%, we can expect an efficiency greater than 95%.

4 Detailed Description of the Processor

Figure 3 is a schematic diagram of the processor. The following paragraphs describe the function of each module, proceeding roughly downwards from the top (i.e. in the direction of data flow).

In order to reduce sensitivity of the algorithm to chamber dead time and other inefficiencies (typically 10% per chamber plane), all track hypotheses having a hit in each station plus at least one additional hit are accepted as tracks (i.e. two of the six planes may be missing). This is facilitated by merging the

hit address lists from primed and unprimed drift chamber planes before hypothesis generation, using the Ordered Merge and Associator modules. The Ordered Merge merges the lists while maintaining monotonicity of wire number, and it appends to each address a low-order bit indicating primed or unprimed plane. The Associator finds pairs of adjacent hits in a primed and an unprimed plane and squeezes them into a single address, appending a low-order bit to indicate pair or single hit. Hit addresses emerging from the Associator thus represent position in units of 1/4 wire-spacing.

Between the Ordered Merge and the Associator are placed a Block Buffer and a Comparator. The Comparator eliminates early hits from previous interactions (as mentioned in Section 3) by cutting on drift time. The Block Buffer is a 256-word memory with three ports: a processor write port, a processor read port, and a bidirectional Transport Bus port. (The Transport Bus[3] is Nevis Laboratories' answer to CAMAC and FASTBUS, and serves to interconnect the on-line computer, the processor, and the various pieces of the data acquisition system.) The Block Buffer stores up to 255 hit addresses emerging from the Ordered Merge and sends them to the processor. If the processor accepts the event, the hit addresses are then read out to the on-line computer via the Transport Bus. The Block Buffer is designed to connect subsystems having independent clocks, allowing the processor to cycle at a higher speed than the Transport Bus or readout system.

4.1 Hodoscope Masking

The three modules following the Associator use hodoscope information to reduce the number of drift chamber hits to be used in hypothesis generation. Since the hodoscopes are sensitive to a much narrower time window than are the drift chambers, many particles from previous or subsequent interactions register in the drift chambers but not in the hodoscopes; these "out-of-time" drift chamber hits are eliminated by keeping only those hits which correspond to hodoscope hits. The usual computer algorithm for this requires searching a list of hodoscope hits for each drift chamber hit (or vice versa), which can take considerable time if there are many hits. In the processor, this search time is eliminated by using the Map module, which contains an associative memory structure: when presented with an input position, it immediately fetches the state of the counter or wire at that position, along with the state of up to four adjacent counters or wires to either side.

The first Normalizer converts position in the drift chamber into units of hodoscope counter widths; its output is thus the number of a hodoscope counter which should have fired if the drift chamber hit is in-time, as well as some fraction bits indicating where within the counter the particle passed. After all hits from the appropriate Y hodoscope plane have arrived at the Map's write

port and been stored, the Map can accept input from the Normalizer. It uses the integer part of the Normalizer output to fetch the state of the requested counter plus one counter to either side, and it puts this out along with the input bits. The second Normalizer then makes a decision as to whether the drift chamber hit should be kept, and it puts out the hit address along with a bit indicating its decision. If the drift chamber hit is near an edge of the central counter, the hit is also accepted if the adjacent counter fired, but if the hit is away from an edge then the central counter alone is considered.

(It is evident that the Normalizer is an unusually flexible module, and a few words on its design are in order. The Normalizer consists of two 256-word x 16-bit tables whose outputs are summed. The high-order 8 bits of an input word can be sent to one table and the low-order 8 bits to the other, allowing any function of the form

$$f\,(x_{\text{high-order}}) + g\,(x_{\text{low-order}})$$

to be computed by preloading the tables appropriately. For example, a Normalizer can be loaded to put out its input times a constant. The Normalizer may be divided into "pages," each page to be used in a different case as determined by the input data, by sending some bits from the input to both tables in common. The partitioning of the input bits is determined by a jumper patch programmed by the user.)

Note that the hodoscope data pass through a Normalizer prior to reaching the write ports of the Maps. This Normalizer is used to transform the hodoscope hit addresses into counter numbers, and it also puts out bits indicating plane, which allow each Map to decide which hits to accept. Thus one Map accepts hits from plane Y2, another from Y3, and a third from Y1, and hits from the remaining hodoscope planes are ignored.

4.2 Trackfinding Loop

After being masked with the hodoscopes, the drift chamber wire addresses are stored in the Lists and counted by the Index Generator. The Index Generator puts out index pairs (8 bits per index) which call forth from the Lists all possible track hypotheses consisting of a hit in Y2 with a hit in Y3. The Page Generator and List-Counters generate multiple passes around the loop for each hypothesis. The Page Generator simply repeats each index pair, once for target and once for dump. The List-Counters pass the index pairs through and also store them internally for potential use on subsequent passes. The passes are identified using the Name bits of the processor bus.

On the first pass, the four Normalizers and two Adders (Arithmetic Operators jumpered to perform addition) generate

predicted positions in the Y1A and Y1B proportional chamber planes, and the Maps and Binary Table accept or reject each track hypothesis, requiring a hit consistent with the prediction in Y1A or Y1B, and a total of at least four out of the six chamber planes. Outputs from the Binary Table pass through the Buffer to the List-Counter's read port, and accepted track hypotheses are re-issued by the List-Counter for the next pass. (The Buffer is required in order to prevent Holds from propagating back around the loop and stopping the data flow. It is a fast FIFO buffer, capable of performing both a read and a write on each cycle.)

On the second pass, those hypotheses accepted on the first pass are masked with the Y1 hodoscope plane. On this pass, only one of the two Maps is used, having been loaded both with Y1B chamber hits and Y1 hodoscope hits in two pages. The two hit streams are merged using the Switch module (an Arithmetic Operator jumpered to pass data through unmodified, switching back and forth as needed from one input port to the other).

4.3 Transverse Momentum and Mass Calculation

Hypotheses accepted on the second pass are repeated by the second List-Counter for calculation of p_t and mass. On this pass the wire address pairs of good tracks are written into a Block Buffer, to be read out and written to tape along with the rest of the event information; this allows the off-line analysis program to monitor the correct functioning of the processor, and it may also permit the array processor and the analysis program to find tracks more quickly, since the y-view reconstruction need not be repeated.

Actually only the y-component of momentum is computed (since no x information is available to the processor), and we choose to approximate (accurate to 20%)

$$p_t \doteq p_y , \qquad m \doteq p_{y1} + p_{y2} .$$

At the cost of building a more complex processor we could have included information from the other chamber views and avoided these approximations, but they are adequate for triggering purposes. The set of four Normalizers and two Adders is used once again, this time to compute the quantities θ_0 (production angle in the y-z plane) and 1/p, which, like positions in station 1, are linear combinations of positions at stations 2 and 3.

To compute p_y, these quantities must be divided. It is not worthwhile to design a divider module capable of cycling at 25 ns just for this one use, so we resort instead to subtraction of logarithms. Since these quantities may be either positive or negative, their absolute value must first be taken; this is accomplished using Normalizers, and the signs are recorded in Name bits. The Tables take logarithms to 8-bit accuracy (which is

adequate for our purpose), which are subtracted using an Arithmetic Operator, and a third Table takes antilog to yield p_y, again with an accuracy of 8 bits.

Since the processor does not use all available chamber information, it does not have the best rejection against accidental tracks, so we must be prepared for events in which the processor finds many tracks, of which only one or two might represent real high-p_t particles. An algorithm for triggering on high-mass events must therefore be able to deal with large numbers of tracks, and to compute the mass of the most massive track pair in each event. To accomplish this we first find the two tracks with the most positive and most negative values of p_y, using the Min/Max modules, then add the absolute values of the two p_y's. This is done separately for dump and target tracks, hence there are four Min/Max modules and two Adders. The calculated p_y and mass values are merged into one data stream using Switches and written into a Block Buffer to be read out with the event, making them available for off-line monitoring.

4.4 Trigger Generation

The ultimate result of the processor's decision is a command issued on the Transport Bus, telling the Block Buffers and other data sources either to read out the event or to skip it and reset the readout system for the next event. This decision is based on the reconstructed p_y and mass values. It is undesirable to reject low-p_y and low-mass events outright, however, since then any bias which the processor might introduce into the data sample can never be corrected. Instead, we prescale events by different factors: events having high p_y or mass are "prescaled" by 1, but only one in sixteen (say) low-p_y and low-mass events are accepted.

This prescaling is performed by the Event Generator Source (EGS), shown at the bottom of Figure 3. The EGS is a Transport Bus module which receives a processor bus input and issues Read and Skip commands on the Transport Bus. It also maintains an event count, which it puts out on the Transport Bus when the event is read out. Its input data stream is a sequence of prescale values terminated by a Complete word. Prescale values may be any power of 2, from 2^0 through 2^{15}. On receipt of Complete, it uses the smallest prescale value of the sequence: if that value is 1, it issues a Read command; if 2, it issues a Read if the low-order bit of the event number is zero; if 4, if the two low-order bits are zero, etc. In this way, fractions ranging from all to 1/32,768 of events can be selected. The prescale values are determined from the calculated p_y and mass values according to the preloading of the two Tables.

We have also implemented a feature allowing the processor to be bypassed; for example, certain classes of study triggers which have already been selected and prescaled by the second-level

trigger logic are accepted regardless of the processor's decision. The leftmost data stream shown in Figure 3 comes from the Trigger Bit Latch system and indicates which second-level triggers fired. If a study trigger fired, the Table sends a prescale factor of 1 to the EGS, forcing the event to be accepted. This also provides a simple way of disabling the processor entirely: the Table can be loaded so that all events are forced through. Since the processor's verdict is still read out, this mode is useful for testing, and we ran the processor in this mode until we established to our satisfaction that our algorithm was working.

5 Current Status

The processor is now debugged and working. At our typical beam intensities the events have many background hits, causing the processor to find many accidental tracks. Some typical events are shown in Figure 4. An average of 10 or 20 tracks are found per event. We are working on improving the accidental-track rejection; options being considered include requiring five out of six chambers per track, using drift time information, and (for muon triggers) using muon proportional tube information.

I wish to thank my collaborators on E605 for their help, and especially Bob Hsiung for his able work in assembling and debugging the processor and its support software.

Postscript (5/14/84): At 10^{10} interactions per pulse and for our typical mix of triggers, the processor reduces the event rate by a factor of 2, with p_y and mass thresholds set at 6 and 8 GeV and prescale factors at 16. The rejection improves to a factor of 5 at 10^9. (These relatively modest factors are due to the already high selectivity of the second-level trigger.) Efforts to reduce the sensitivity of the algorithm to accidentals and improve the rejection are in progress.

REFERENCES:

1. W. Sippach, G. Benenson, B. Knapp, IEEE Trans. Nucl. Sci., NS-27, 578 (1980); H. Cunitz, Y. Hsiung, B. Knapp, W. Sippach, "Data Driven Processing," in Proceedings of the International Conference on Instrumentation for Colliding Beam Physics, SLAC Report-250, June 1982.

2. J.P. Rutherfoord, "On-Line Filtering of High Energy Physics Data With an Array Processor," this conference.

3. J.A. Crittenden, G. Benenson, H. Cunitz, Y.B. Hsiung, D.M. Kaplan, W. Sippach, B. Stern, "A Data Acquisition System for Elementary Particle Physics," to be published in IEEE Trans. Nucl. Sci.

PROCESSOR BUS FORMAT

24-bit Cable:

Data	Name	Control
$X_{15} \cdots X_0$	$N_3 \cdots N_0$	C V H B

Control Bits:

V	Valid: here is a data word
H	Hold: couldn't accept that word
C	Complete: end of block
B	Block Reset: abort this block

NOTE: H and B travel against the flow of data, i.e. from receiving module to sending module.

Table 1

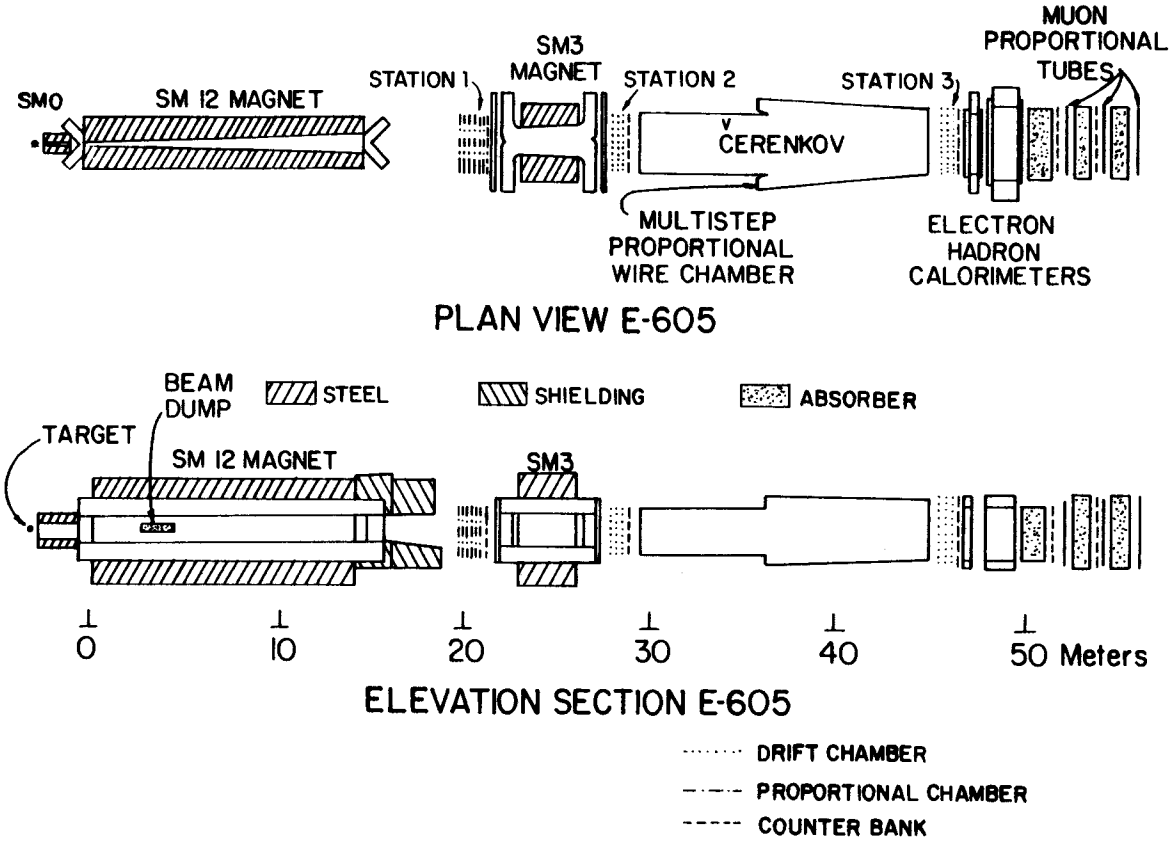

SM0
SM 12 MAGNET
STATION 1
SM3
MAGNET
STATION 2
STATION 3
MUON
PROPORTIONAL
TUBES
CERENKOV
MULTISTEP
PROPORTIONAL
WIRE CHAMBER
ELECTRON
HADRON
CALORIMETERS
PLAN VIEW E-605
BEAM
DUMP
STEEL
SHIELDING
ABSORBER
TARGET
SM 12 MAGNET
SM3
0
10
20
30
40
50 Meters
ELEVATION SECTION E-605
DRIFT CHAMBER
PROPORTIONAL CHAMBER
COUNTER BANK

Figure 1

Processor Modules

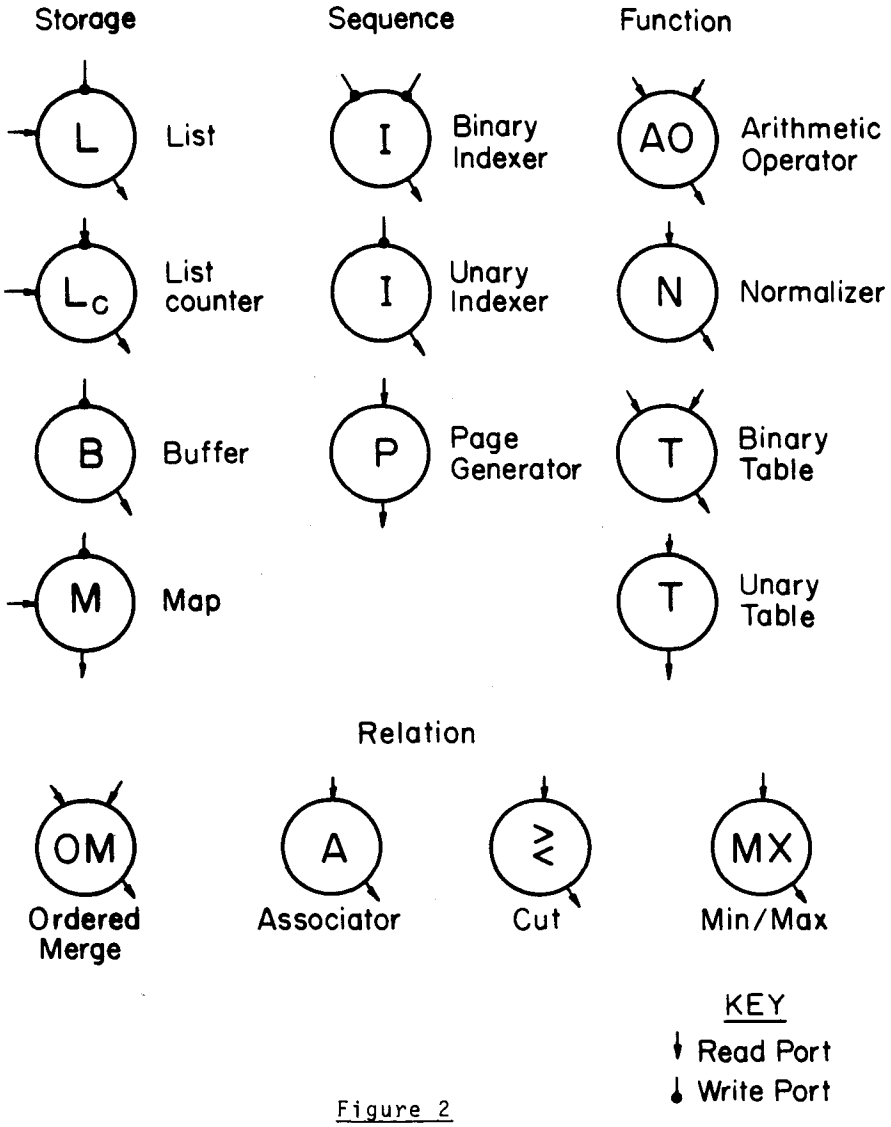


Figure 2

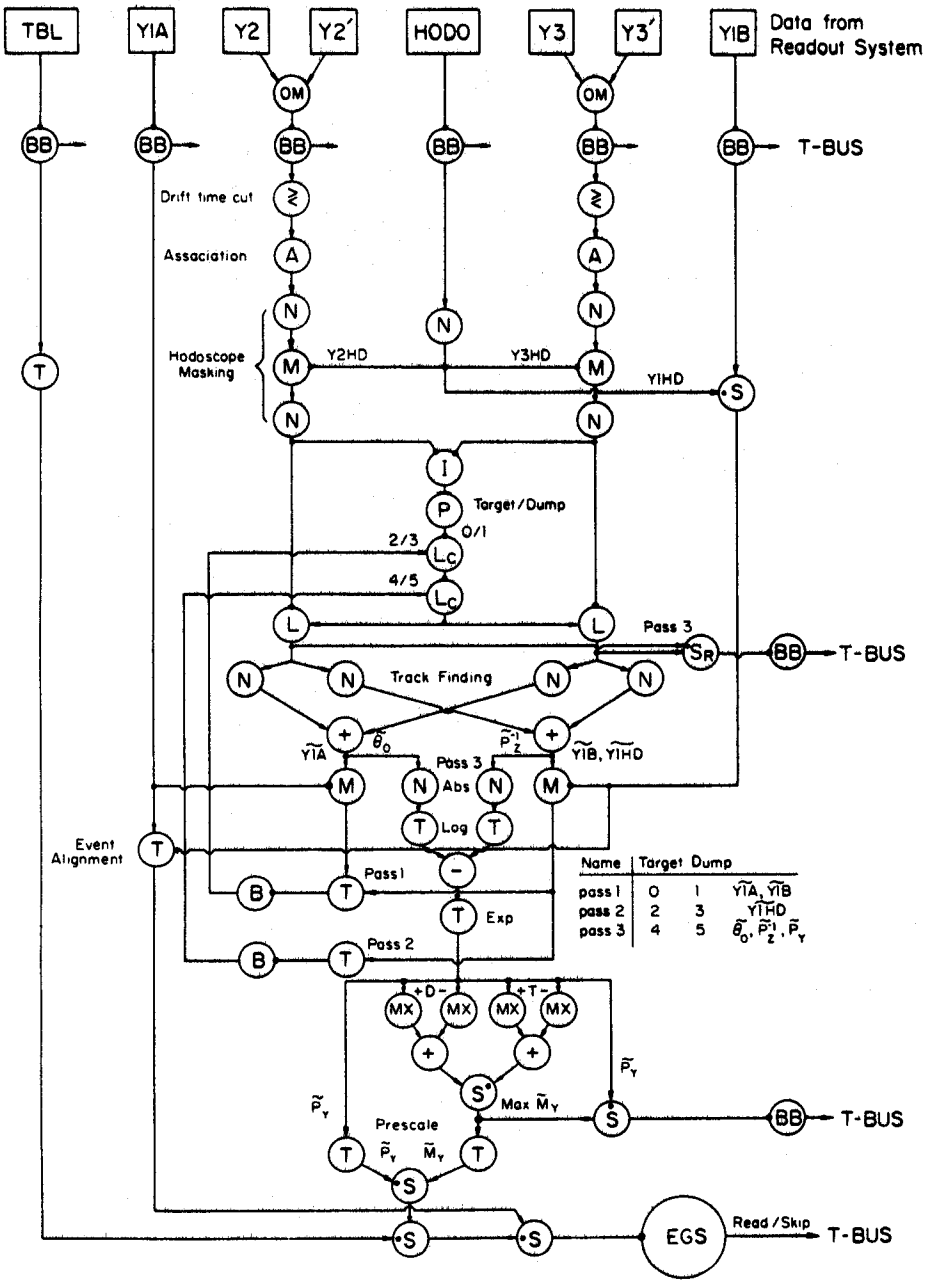


Figure 3: Processor Schematic

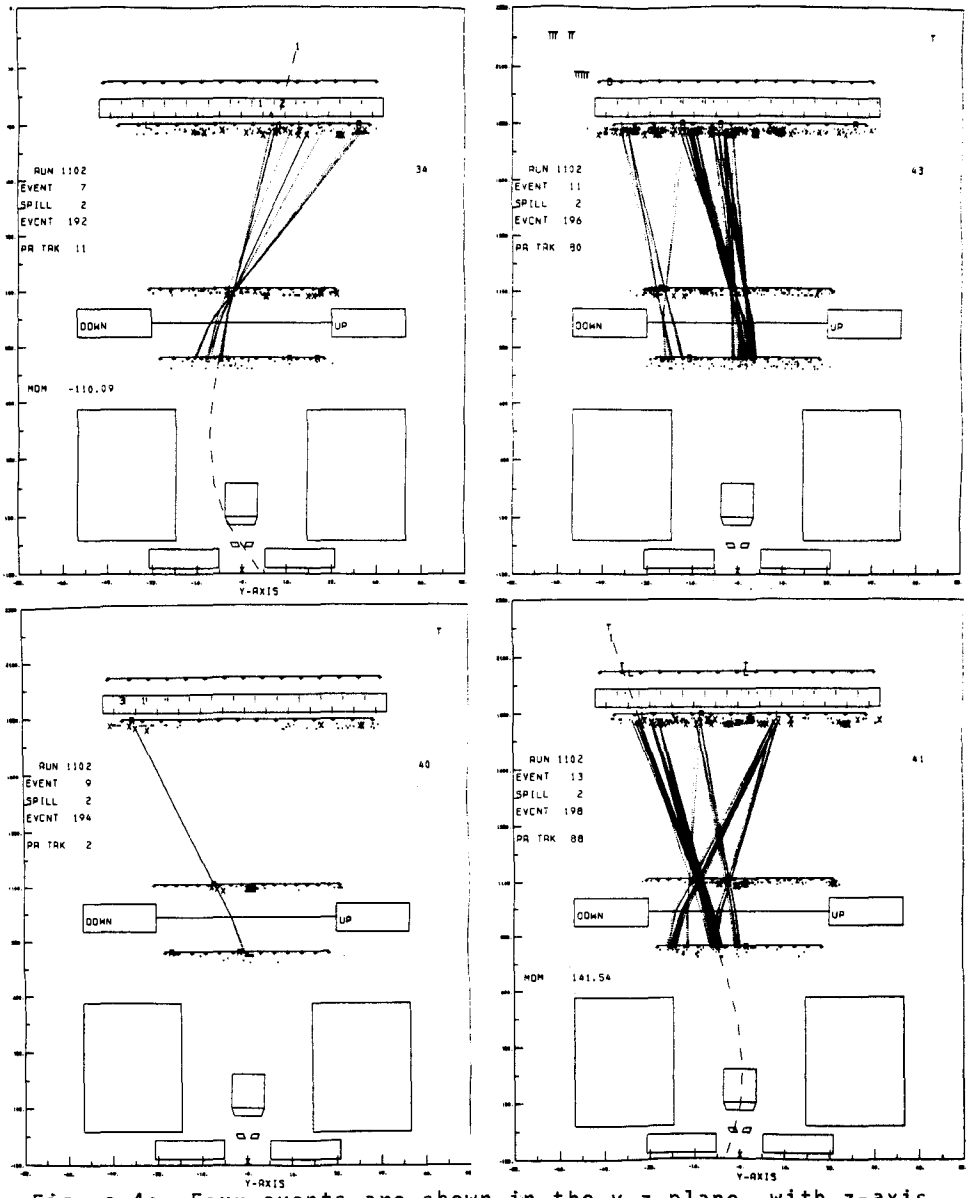


Figure 4: Four events are shown in the y-z plane, with z-axis compressed a factor of 10 relative to y-axis. Tracks found by the off-line analysis are drawn with long dashes, those by the processor with short dashes.

A DATA DRIVEN PARALLEL PIPELINED HARDWARE RECONSTRUCTION PROCESSOR*

L. Borten, M. Church, E. Gottschalk, R. Hylton, B. Knapp, W. Sippach, B. Stern
Columbia University, Nevis Laboratories, Irvington, NY 10533

E. Hartouni, D. Jensen, M. Kreisler, M. Rabin, J. Strait
University of Massachusetts, Amherst, MA 01003

C. Avilez, W. Correa, J. Escalona, H. Morales, P. Salas, A. Zentella
University of Mexico Instituto de Fisica, 10000, Mexico DF, Mexico

C. Christian, G. Gutierrez, S. Holmes, R. Huson†, A. Wehmann
Fermilab, Batavia, Illinois 60510

ABSTRACT

An overall view is presented of Brookhaven E766 and Fermilab E690, which includes the construction and development of an on-line processor. The primary goal is to investigate the diffractive production of heavy quarks by studying exclusive multiparticle final states. The close relationship of the detector and read-out system with the processor and the physics goals that motivate this research program are discussed. In addition, we discuss the functions of the processor and explain an example of a module that forms part of the track fitter. We estimate we will reconstruct on-line 10^5 events/sec.

Introduction

By the end of the decade of the 60's it was expected that with the availability of high-energy beams, an explosion of data produced in high-energy collisions would lead to the systematic study (among other things) of many particle final states, hoping to learn more about hadron dynamics.[1] With the advent of the higher energy machines in the U.S. and Europe during the decade of the 70's, a wealth of events in high-energy collisions appeared; however, the exclusive multiparticle final states expected to provide more insight into the interaction of high-energy particles could not be studied. The problem was not only to produce those states, but to detect them and properly present them for further analysis.

---

*Presented by C. Avilez, on sabbatical leave at Fermilab, Fellow of the John Simon Guggenheim Memorial Foundation.

†Currently on leave at Texas A&M.

In order to detect and produce the appropriate information that leads to the full reconstruction of an exclusive reaction, the dead time of the apparatus has to be minimized and its read-out fast enough to achieve the proposed goals. Several levels of logical triggers have been found to be insufficient to identify the events under discussion, and the necessity of on-line processing is by now widely accepted.

With a highly segmented detector, fast read-out and on-line processing, it is possible to study the details of the dynamics of hadron physics in exclusive reactions.

E766 (BNL) and E690 (Fermilab)

The experimental high-energy physics program of the collaboration University of Massachusetts (Amherst)/Columbia University (Nevis)/ University of Mexico (IFUNAM)/Fermilab, is to explore the physics of the exclusive multiparticle final states.

Our primary physics goal is the comprehensive study of hadronic spectroscopy and production mechanisms, eventually covering strange, charm, and bottom particles at Tevatron energies. The construction of a multiparticle detector capable of measuring moderately complex reations (up to 20 particles) with high resolution and efficiency at moderately high rates is underway. A crucial feature of the experiment is detailed on-line reconstruction of up to 10^5 events/second chosen from more than 10^6 interactions/second. With the E690 spectrometer, this should allow the measurement of more than 10^4 fully reconstructed charm pairs and as many as 10^2 bottom pairs per hour of Tevatron beam.

The detector for this program is seen in Fig. 1, which also shows the part of the detector already in operation at Brookhaven. To perform accurate measurements over a wide range of particle types, angles, and momenta, the detector components must fit and function together.

The detector in operation at BNL consists of six drift chambers totaling 11,500 signal wires in a low-field, wide-aperture magnet (7kG, 4 ft×8 ft). Charged particle trajectories are efficiently reconstructed with momentum resolution $\delta p/p < 1\%$ (FWHM) from 200 MeV/c to 10 GeV/c. Mass resolutions for K_s and Λ^0 are presently 8 and 2 MeV (FWHM), with further improvement expected. Direct particle identification is provided by 102 time-of-flight counters and a segmented-threshold Cherenkov counter (96 channels). We have uniformly high acceptance for a wide range of topologies, particularly final states containing heavy particles. We intend to record a large number of events, fully reconstructed and identified as such, typically recording both raw and reconstruction results.

One of the main endeavors of E766 is to produce a sample of $\sim 10^6$ $\bar{\Omega}^-$ for which the use of a real-time processor system is an absolute necessity. Our program at BNL also includes the search for highly-inelastic baryon resonances and further studies of hyperon polarization. In particular, we are interested in understanding Λ^0 polarization in exclusive reactions, since virtually nothing is known about the contribution of specific final states to this phenomenon.

The detector and processor form a hierarchy of independent subsystems linked to each other by high-speed buffers and linked to host computers through the Nevis transport system.[2] Transport system buffers between the detector and the processor allow data acquisition to bypass the on-line processor and allow data recorded on tape to be fed back to the processor.

The Hardware Processor

The processor is a way of dealing with very well-defined events, however rare, in a high rate environment, where very large-scale numerical computation is expected. It can adapt in the most elastic and flexible way to the computational needs of a wide variety of situations encountered in high-energy physics experiments. In a somewhat restricted way, it has been tested in Fermilab's E605,[3] and its first large-scale applications are E766 and E690.

The processor is a **data driven synchronous pipeline.** It has distributed memory and a modular structure with general interconnectability to match the computational needs of a given project. The pipeline is realized in terms of a parallel processor whose operands receive data when the data are present and the destination is available.* The synchronous aspect of the pipeline, in conjuction with two control bits, allows alignment of the data, avoiding juxtaposition of boundaries of blocks of information.

Data Processor For E766 and E690

A central feature of the processor is its extreme capability to match the computational requirements of a modern high-energy physics experiment. The problem we are interested in is the track reconstruction of all the particles in a moderate multiplicity (7 to 20) final state. The fitted track parameters provide us with charged particle momenta and trajectories which are used to reconstruct interaction and decay vertices. The track reconstruction consists of four stages: track finding, matcher, track fitter, and clean-up.

The track reconstruction by itself is most challenging and involves two extreme situations: pattern recognition to find tracks, which requires relatively little computation but a large number of combinatorial situations (actually dealt with by loops), and the track fitting, with a smaller number of decisions involved but requiring much more computation.

The track finding problem, which for straight lines has been discussed extensively,[3] is primarily solved with modules that involve bit parallel structures in a sequential word process. Experiment #605 at Fermilab has used the track finding in a single view of drift chamber with marginal constraints. In E766 (and E690) we look for tracks of charged particles in a magnetic field. In each view of the drift chamber, we obtain three wire numbers and the deviation from a straight line to represent the track. In Fig. 2 we show

---

*This is the data-driven principle that assures maximum hardware utilization, minimizing the number of idle components at any time.

the algorithm to find tracks in a moderate magnetic field by a single view. The results from different views are matched with strong constraints. For instance, the single-view curvature parameter is independent of view, providing a quick constraint. This pattern recognition, i.e., finding lines in single views and matching them to form track candidates, uses only wire addresses, while ignoring drift times.

We now concentrate on the other extreme, namely, the track fitter. In this part of the processor, as the computation proceeds, more bits are added to accurately represent the result of the calculation, changing thereby the size of the word involved.

The coordinates that specify the track are non-linear functions of the measured parameters. The process of fitting consists of finding the best set of parameters by minimizing a χ^2. Following Newton's method, a solution can be found to the non-linear problem, and in fact a few iterations, if an appropriate starting point is available, i.e., a zeroth order set of parameters $P^{(0)}$. The process of track finding provides us with this first approximate parameterization. In an operational way one first calculates,

$$P_i^0 = \sum_n A_{ni} X_i + A_{0i}, \quad n = 1,.., 6; \; i = 1,..., 5$$

The wire numbers (X_i) are ten-bit words and the resulting five parameters (P_i^0) are 12-bit long. This kind of operation is done in hardware by a module called a sum multiplier which is explained below.

After having the zeroth order parameters we are interested in forming higher order nonlinear products of them, producing a set of 11 parameters (P_j, $j = 6, \ldots, 11$) to be used in a Taylor expansion beyond the linear terms. Again, using a sum multiplier we predict wire coordinates

$$X_i = \sum_{n=1}^{5} A_{in} P_n^{(0)} + \sum_{n=6}^{11} A_{in} P_n^{(1)}.$$

This produces 16-bit wire coordinates, 10-bit integer wire number and a 6-bit fraction. One may now get the difference of the predicted value from the nearest measurement within a Map, and by using a sum multiplier again produce a first correction to the zeroth order set of parameters. After a few iterations the drift time may be included to obtain the best predicted value for the coordinates within the measured precision of the experiment. See Fig. 3.

Comments on Modules; One Example of Bit Serial

The modules of the processor have three general requirements, namely, that they perform simple operations, have simple communication control, and if needed, have small storage independent of the size of the net. In the preceding talk, Dan Kaplan explained in more detail some features of the bit parallel modules and their protocol. Here we would like to explain the function of one of the most important modules of the track fitter, the sum multiplier. In an extremely simplified form showing the essence of the idea behind the design, let us do the following calculation

$$\Sigma\ a_n X_n,$$

where the X's are represented by three bits. The constants a_n can in principle be described by all the bits we like, independent of the "size" of the X's. We may write, assuming n=1,2

$$\Sigma\ a_n X_n = a_1 X_1 + a_2 X_2$$
$$= a_1\ (X_1^0\ 2^0 + X_1^1\ 2^1 + X_1^2\ 2^2 + \ldots)$$
$$+ a_2 (X_2^0\ 2^0 + X_1^2\ 2^1 + X_2^2\ 2^2 + \ldots).$$

We now rearrange this expression and factor out all powers of 2

$$\Sigma_n\ a_n\ X_n = (a_1 X_1^0 + a_2 X_2^0)\ 2^0$$
$$+ (a_1 X_1^1 + a_2 X_2^1)\ 2^1$$
$$+ (a_1 X_1^2 + a_2 X_2^2)\ 2^2.$$

The expressions in parentheses can be calculated in advance, in terms of the different possibilities of the weights of the binary expansion and loaded into a look-up table. The two vertical arrows show the two words that, serially fed into the module, provide the address to retrieve the precalculated combinations from the look-up table. In Fig. 4 we schematically show the structure of the sum multiplier.

The associative memory concept is extensively used both in the bit parallel and in the bit serial modules, in modules called Maps. Here the data are retreived by value and provide us with a considerable speeding-up factor. In the track finding problem, the use of a Map reduces an N^3 problem to an N^2 one. Using first and third chamber information to predict the position at the intermediate one allows us to retrieve by value the experimental result. In the least square process of the track fitting, we systematically have to compare results of evaluations with the experimental measurements in the first iteration, or in subsequent ones, with the result of the previous step. A Map module has two input ports, one to write on the memory of the module, the other a read port that receives whatever value was predicted and has to be tested for its validity. A third port of the module is just an output that provides a "road" of values around the tested one. Provision is made to mask the word at the read port to select fields of bits that are asked to be represented by the data written on memory. The Map itself does not "decide" if the tested value passed the test or not, but by providing a road of possible values around the predicted one, produces a word to be fed into a look-up table with precalculated results of the test as a function of the road.

Comments and Remarks

After recognizing the complexity of the problems of event selection and data analysis, we have set out to provide a general approach to large scale computation. Our first large scale operation will be at Brookhaven in 1985. This should provide an adequate demonstration of a more generally useful

inexpensive approach to the larger computation problems of high-energy physics experiments, as well as other fields.

References

[1]See for instance, C. N. Yang, "Hypothesis of Limiting Fragmentation" in the Third International Conference on High Energy Collisions (Gordon & Breach, 1969), p. 509.

[2]J. A. Crittenden, G. Beneson, H. Cunitz, Y. B. Hsiung, D. M. Kaplan, W. Sippach, and B. Stern, "A Data Acquisition System for Elementary Particle Physics," to be published in IEEE Trans. Nucl. Sci.

[3]See D. M. Kaplan, talk at this symposium. W. Sippach, G. Beneson, B. Knapp, IEEE Trans. Nucl. Sci., **NS-27**, 578 (1980); H. Cunitz, Y. Hsiung, B. Knapp, W. Sippach, "Data Driven Processing," in Proceedings of the International Conference on Instrumentation for Colliding Beam Physics, SLAC Report-250, June 1982.

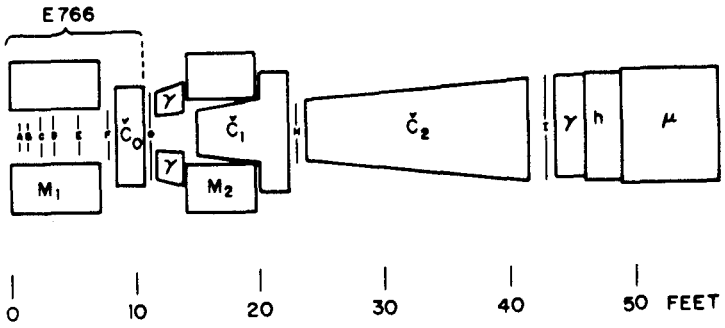


Fig. 1. Schematic layout of E690 (FNAL). A-I are drift chambers; M1 and M2 are large aperture modest field analyzing magnets; C_0, C_1, and C_2 are highly segmented threshold Cherenkov counters and downstream a subsystem of calorimeters. E766 is the part of the detector operating at Brookhaven National Laboratory.

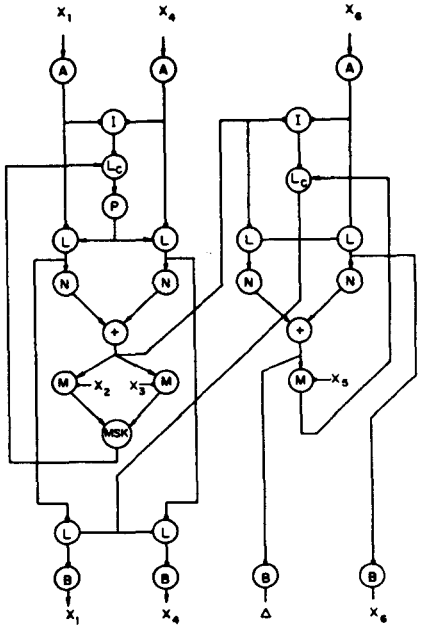


Fig. 2. Moderate magnetic field track finder. Uses wire numbers of single views of 6 drift chambers. X_1 and X_4 are used to predict values at either X_2 or X_3 (done at different cycles. This is why the straight line finder X_1-X_4 requires a page generator module [P]). The predicted value (X_2 or X_3) is used with X_6 to predict a value for the deviation from a straight line tested in a Map by X_5.

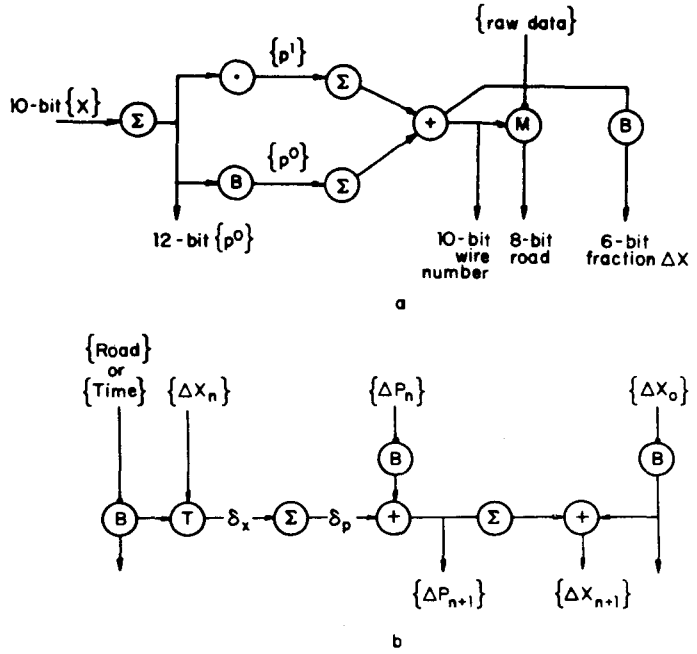


Fig. 3. Track fitter. Schematic diagram. The symbols stand for sum multiplier (Σ), buffer (B), adder (+), Table (T), Map (M), multiplier (•). For every high order product of $p^{(0)}$, one introduces a multiplier. In part a, (•) stands for all the multipliers needed to produce the required higher order terms of the parameters. The same applies to Σ.

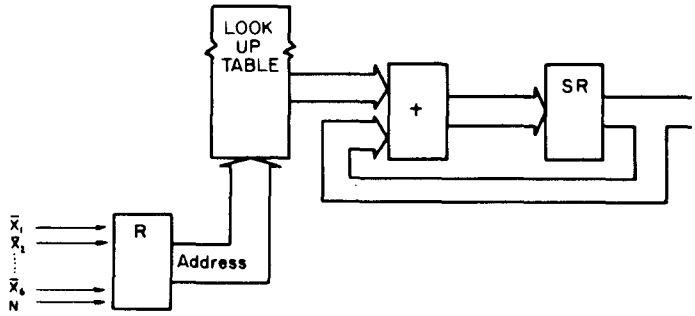


Fig. 4. Operation of the sum multiplier. Words are fed bit serially into a register. The nth bits of all the input words provide an address used to retrieve from a look-up table precalculated values of the linear combination $\Sigma\ a_n\ X_n^m$, where the X_n^m are the weights of the binary representation of X_n.

QUESTIONS AND ANSWERS

(Editor's Note: The question and answer sessions for the talks by Kaplan and Avilez were combined.)

Q: How much pay off do you get from the "refit" using the drift time information, compared to computation time it takes?

P. Lebrun

A: We need ~ 100μ to 200μ resolution. A normal, usual size PWC cannot give such a resolution. Therefore we do have to do this "refit."

C. Avilez

Q: Needs for off-line analysis?

A. Brenner

A: Most experiments will require only minimal off-line analysis.

M. Kreisler, C. Avilez

Q: Considering that typical off-line reconstruction codes consist of **many** pages of FORTRAN, how can you expect to build systems of that many modules?

M. Fischler

A: The mapping of FORTRAN in the most general sense into hardware modules is not the way to examine the problem. Rather the question is whether one can reconstruct a large number of tracks from complex topologies in a finite (i.e., small) number of modules? The answer to that is yes.

M. Kreisler

Q: How much does one of your modules cost?

M. Delfino

A: $100 to $300, depending on volume.

M. Kreisler

Q: How many experiments have the resources of talent to put together a data-driven system that will "fully" reconstruct their data?

T. Nash

A: At present, very few although we expect that to change as this technology is developed.

M. Kreisler

Comment: At E766, we currently write data limited by tape writing speed yielding approximately 4 tapes/hour. Such tapes take ~ 4 CPU hours on a Cyber. This would yield a mass of data not easily passed in toto through an off-line computer center. Thus the bulk of the data is filtered by the processor and then taken to an off-line system. Experiments such as ours will need computer time, of course, but **not** at the scale one would first guess by the magnitude of our data stream.

M. Kreisler

Q: A pipeline can only compute as fast as its slowest element. For either configuration, what is the efficiency of the pipeline for a typical event?

R. Fine

A: No answer

ON-LINE FILTERING OF HIGH ENERGY PHYSICS DATA WITH AN ARRAY PROCESSOR

John P. Rutherfoord
University of Washington, Seattle WA. 98195

In a high data rate experiment (E605) at Fermilab we are using a programmable Array Processor, an FPS100E, to reject events using criteria which are either too complicated for the existing hardware triggers or are not anticipated with enough lead-time to build a special hardware trigger. The advantages and limitations of this approach are discussed.

Modern elementary particle physics experiments have a number of levels of trigger decision. Typically the first level trigger is very fast and simple and provides a high rejection rate. Successive levels are usually slower and more sophisticated and each filters out all but a fraction of events presented to it until finally the rate is low enough that surviving events can be written to magnetic tape. As magnetic tape technology has improved and as physics experiments acquire data over many years, it is possible to fill data tapes at a rate much faster than off-line computers can analyze. It is now increasingly important to make even more sophisticated trigger decisions before data is written to tape. An inexpensive Array Processor can perform filtering algorithms on data at rates comparable to the speed with which that data can be written to mag tape. An example of such an application is described here.

Experiment E605 at Fermilab is a large, fixed target experiment optimized to search for narrow structure in the dilepton spectra at large invariant mass. The primary proton beam intensity is a significant fraction of that available from the accelerator. Because the processes of interest are a tiny fraction of the total hadronic cross section, the challenge for the experimenters is to reject the huge backgrounds. As is typical in experiments of this kind, off-line analysis shows that many background events survive all levels of hardware trigger.

In this experiment an event is subjected to three levels of hardware filters, the third of which is the Nevis Trigger Processor described at this conference by Dan Kaplan. During the accelerator spill the data from an event (~500 16-bit

words) surviving these filters is stored in a fast (150 nsec/16-bit word) buffer memory with a capacity of 4 Mbyte. Up to 4000 events can be stored during the 15 second accelerator spill with ~2% readout dead-time. Between spills the data can be dumped onto mag tape. The full 4 Mbyte can be written in 16 seconds. As the current minimum time between spills is 50 seconds, this speed is more than adequate. In this way we can fill a 6250 BPI, 2400 ft. data tape in about 30 minutes.

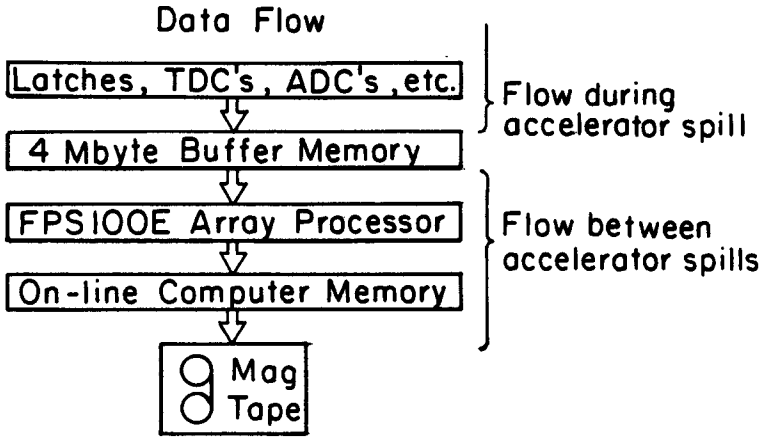
Our current, first level off-line analysis program (not yet optimized for speed) which runs on a Cyber 175 takes an average of 100 msec. per event or ~3 hours of CPU time per data tape.

Rather than write the data from the buffer memory directly to tape, a Floating Point Systems FPS100E Array Processor DMA's the data, one event at a time, into one of two buffers in its own data memory, performs a filtering algorithm on that event, and, if it survives, DMA's that event out to the on-line computer's memory and from there onto mag tape. Because there are two data buffers in the Array Processor, the analysis on one event overlaps in time the reading-in and writing-out of the event in the other buffer. The Array Processor's DMA speed was measured to be 4 microsec/16-bit word. Thus it takes 8 seconds to read in 4000 events of 500 words each and another 8 seconds to write them all out again. (Usually only a fraction are written out.) Because neither the Array Processor DMA speed nor the mag tape drive challenge the natural bandwidth of the on-line computer's bus (a Unibus on a PDP 11/45), the full 4 Mbyte from the buffer memory can be passed through the Array Processor and written to tape in the same 16 seconds as when the Array Processor is removed from the chain.

The current filtering algorithm, programmed into the Array Processor, looks for two or more muons penetrating our calorimeter and shielding and counting in two hodoscope planes and in three planes of proportional tubes. The algorithm is written in Fortran IV. Editing, compilation, linking, and testing of algorithms are all done on a VAX. The load module is transfered to the on-line computer on a floppy and automatically loaded into the Array Processor's program memory at the beginning of a data acquisition run. The algorithm takes about 1 msec. per event. On a VAX 11/780 with FPA this same algorithm executes in 2 msec. and on the Fermilab Cyber 175 in about 250 microsec. With no degradation in the speed with which data flows through the Array Processor, more sophisticated algorithms could take up to 4 msec/event. Or 12 msec/event would still allow all events to be processed during the 50 seconds between spills.

This low cost ($35K) Array Processor has a maximum program memory size of 4K 64-bit words. This is roughly equivalent to 10 to 20 pages of Fortran code. We are currently using a little over 3k words for our routines. The program memory size is the first limit we would hit if we were to try to expand our

application. Were we to program in the Array Processor's assembly language the same algorithm would require much less memory and would run much faster (~ factor 4). Fortran IV is not a good match to the architecture of the Array Processor but it is the only higher level language supported by Floating Point Systems. The programmer time required to write an algorithm in the Array Processor's assembly language (more similar to microcode) compared to coding in Fortran IV is approximately an order of magnitude for an experienced programmer.



QUESTIONS AND ANSWERS

Q: Is it correct that the Array Processor part of the analysis is only 1/3% of the full analysis?

M. Fischler

A: Yes

Q: Have you compared the results of the array processor with the Cyber results?

L. Leipuner

A: No

Q: Why the choice of a FPS100 rather than, say, an Analogics 500 which can swallow data at a much faster rate (5-6 MHz)?

R. Fine

A: No answer

Comment: Speed of data transfer for FPS100 to UNIBUS is limited by UNIBUS speed not by speed of array processor. FPS100 itself could transfer at ~ 4×10^6 words/second.

J. Amann

A CHARGED KAON TRIGGER USING THE M7

The E-400 Collaboration presented by Paul Lebrun
Fermi National Accelerator Laboratory*

Introduction

The goal of experiment E-400 is to accumulate a large sample of hadroproduced charmed events. Since the most likely decay of a charmed particle is via the emission of a kaon, a charged, high momentum kaon trigger has been designed using a previously build fast processor (the M7 computer)[1,2] for experiment E-401.[3] This report demonstrates the flexibility of such a processor.

The Experiment

The experiment is running in the neutral-wide band beam at Fermilab; the relevant parts of the spectrometer and the data acquisition system are shown in Fig. 1. The 10% interaction length target consist of Tungsten, Beryllium and active silicon and is followed by a vertex detector and a trigger counter. The first analysis magnet, characterized by a transverse momentum kick of 400 MeV/c, is followed by 3 multiwire proportional chambers and the first Cherenkov counter. The second analysis magnet (transverse momentum kick of 600 MeV/c) followed by 2 MWPC's gives us good momentum resolution for central, high momentum tracks. P3 has 2 mm wire spacing in all 3 views (x, non bend view, u and v, characterized by a stereoangle of 11 degrees). P4 has 3 mm wire spacing in the x view, 2 mm in the u and v view with the same stereoangle. The counters C1 and C2 have a pion threshold set at 5.4 GeV and 10.8 Gev, respectively. Each of those Cherenkov counters has 34 cells, the geometry for C1 and C2 is displayed in Fig. 2. On the downstream face of C2, a scintillation counter hodoscope (CH2) matching the C2 cell geometry is mounted, allowing Cherenkov related fast triggers.

A fast trigger is obtained by requiring a coincidence between the target counter and the trigger counter located downstream of P4. The second level trigger is obtained by requiring (i) a signal from the active target, (ii) the MWPC hit multiplicity average over P0, P1, and P2 being at least 4, average over P3 and P4, 2; (iii) the total energy deposit in the lead glass array and in the hadron calorimeter being greater then 100 GeV, allowing us to trigger on the high energy component of the neutron spectrum. This second level trigger achieves a factor of 4 in background rejection.

The M7 Trigger

The M7 is a small fast computer with an instruction set designed to match the on-line pattern recognition problem of a high-energy physics experiment. The processor is a five address, microprogrammed pipelined, ECL machine with simultaneous memory access to four operands which load two parallel multiplexers and an ALU.

A charged kaon with momentum between 20 and 40 GeV will produce light in C1 and not in C2 and hence will fire a fast electronic trigger consisting of CH2 C1 $\overline{C2}$. Unfortunately pions with momenta between 5.4 and 10.8 GeV will also fire this fast electronic trigger. In order to discriminate against these lower momentum tracks, the M7 processor provided a crude momentum measurement using hit information from chamber P3 and P4. Coarse hit information are defined as the "OR" signals from 8, 16, or 32 signal wires, depending on the location of the wires in the chamber; the central region being always the high density, fine grained region. This defines 32 active "bands" per plane. The M7 search for a triplet (x, u and v) in P4 aligned with an active Cherenkov cell, assumed the presumed track is issued from the target, and search for a corresponding hit in P3x (x refers to the non bend view). A triplet in P3 completes the candidate tracks, the track momentum can be measured by virtue of the different spectrometer momentum dispersions present at these two chambers. Because of the crudeness of this measurement, only a minimum momentum cut of 18 GeV is applied to discriminate against pions below the C2 threshold. Above 120 GeV, no momentum information is available.

The whole program was about 120 instructions long, the excecution time was typically 50 to 200 us, depending on the complexity of the event. Prior to data taking, the hardware has been checked on previously recorded data, by sending the pseudo hit information issued from the PDP 11/45 to the M7 interfaces through CAMAC. The M7 gives it's answer through the Transfer Memory modules (TM) to the 11/45, relevant tracks coordinates were compared, the set up was found to be better than 99% efficient. During normal data acquisition, the efficiency of the processor was constantly monitored by comparing TM data block with off-line FORTRAN emulators.

The M7 kaon trigger rejects 2/3 of the events satisfying the second level trigger while being 70% efficient at passing events with an analyzable kaon track identified by C1 and C2 in the extensive offline analysis. The 30% inefficiency is not due to hardware failure, but reflects problems inherent in the greatly simplified Cherenkov algorithm employed by the M7, such as sharing light of a given track amoung several Cherenkov cells or tracking confusion in high multiplicity events. Fig. 2 illustrates the algorithm by showing a typical event. Tracks locations are black dots, the circles represent the Cherenkov light ring, shaded bands are the active MWPC bands in P4 inducing the trigger. Finally, this trigger is fully programmable, allowing great flexibility. Triggering on proton versus antiproton, $K^{\pm}$, $K^{\pm}P$, or $K^{*}K$ is presently being considered.

References

*Operated by Universities Research Association Inc. under contract with the U.S. Department of Energy.

[1]The M7 - A Computer for On-line Track Finding, D. Harding et al., IEEE Trans. Nucl. Sci., Vol. NS-30, No. 5 (1983).

[2]The M7 - A High Speed Digital Processor for Second Level Trigger Selection, T. Droege et al., IEEE Trans. Nucl. Sci., Vol 25, No. 1, 698 (1978).

[3]Psi Photoproduction at at Mean Energy of 150 GeV, M. Binkley et al., Phys. Rev. Lett., 50, 302 (1983).

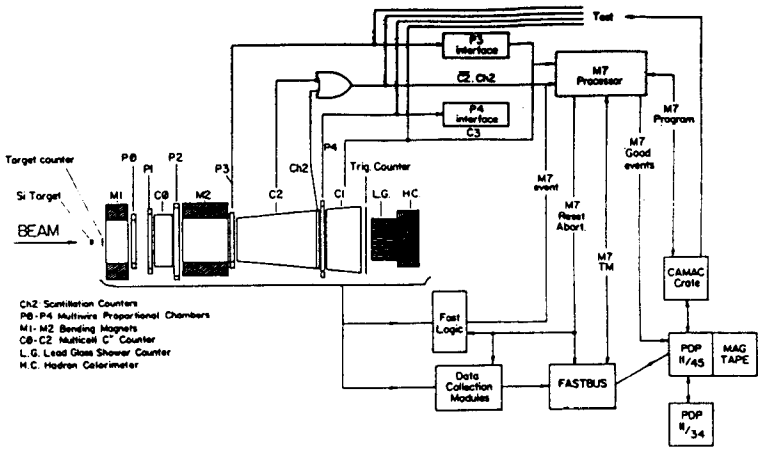


Fig. 1. The E-400 trigger elements and data acquisition.

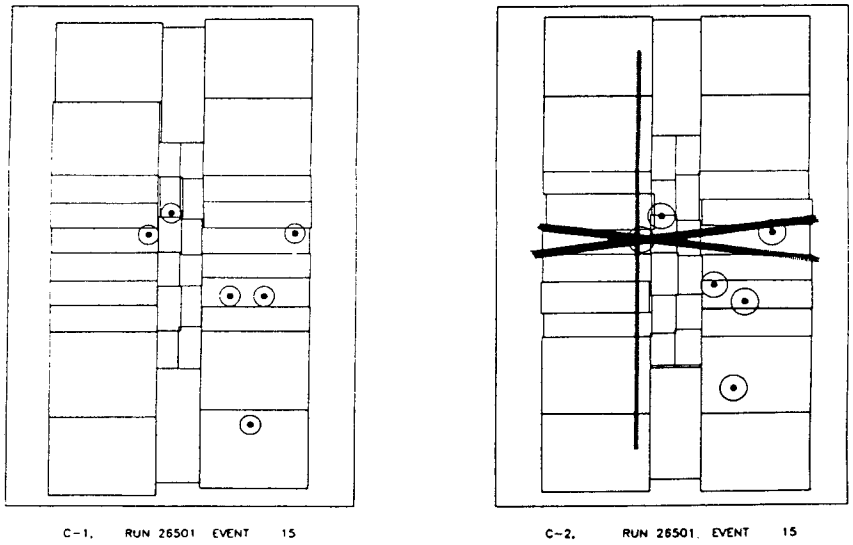


Fig. 2. Typical event display.

QUESTIONS AND ANSWERS

Q: 1) How many tapes do you write?

2) How do you analyze your tapes?

A. Brenner

A: 1) 1 tape/12 minutes.

2) Off line.

Q: You said you write one 6250 BPI tape per 12 minutes. How many tapes do you end up with after one "season" (i.e, year) of running?

M. Delfino

A: Approximately 1500 tapes.

A TRIGGER PROCESSOR FOR A FERMILAB DI-MUON EXPERIMENT

John F. Greenhalgh
Princeton University
Princeton, N. J. 08544

A trigger processor is described which is currently in use in Fermilab di-muon Experiment 615.

Introduction

Fermilab Experiment 615 is designed to study the characteristics of muon pairs produced in the forward direction by negative pion beams incident upon a tungsten target.[1] In the continuum region of mu-pair invariant masses between the J/ψ and Υ resonances, the production occurs predominantly through quark-antiquark annihilation (the Drell-Yan mechanism[2]), an electromagnetic process with extremely small cross section (less than 100 pb) compared to the total $\pi^- p$ cross section (~ 24 mb). Consequently, an experiment dedicated to studying specific kinematic regions with good statistics, such as this one, requires both a high intensity beam and a powerful means of rejecting unwanted events at the trigger level.

Overview of Apparatus and Trigger

A schematic of the apparatus is shown in Fig. 1. A 7.3-m-long dump consisting of BeO, Be, and C absorbs most of the particles produced in the target, except muons. The detection apparatus downstream is live through the beam region in order to achieve good acceptance at large x_F. The dump fills the tapered gap of a dipole magnet which tends to focus the low momentum member of asymmetrically produced muon pairs, thereby affording good acceptance in the interesting angular variable $\cos\theta^*$ from -1 to +1.[3] With its transverse momentum kick of 3.2 GeV/c, this "selection" magnet is largely responsible for the ability of the trigger processor to discriminate between sought after high mass muon pairs and background events, simply on the basis of the topology of the muon trajectories through the downstream apparatus.

Four scintillation counter hodoscope planes, C, D, E, and F, are used to

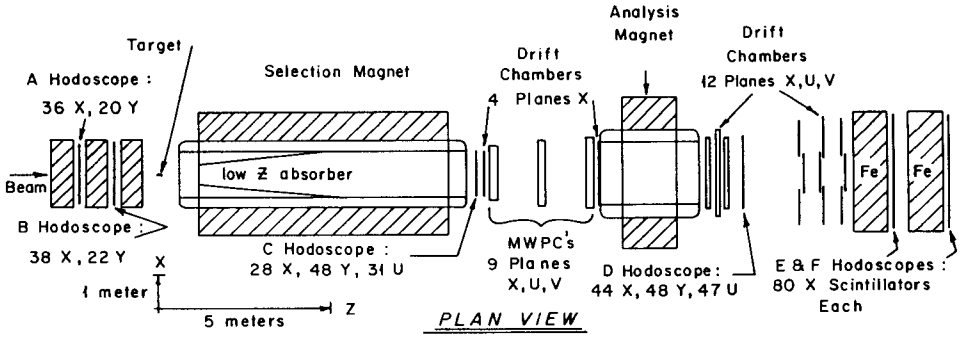


Figure 1. E615 apparatus in 250 GeV/c pion beam configuration

trigger the apparatus on muon pairs, and two others, A and B, to veto triggers containing muons from beam pion decays (halo muons). The E and F banks, situated behind one and two meters of steel, respectively, provide an unambiguous identification of muons.

The trigger for the experiment is made in three successive stages. Under normal data-taking conditions, only events satisfying all three levels of logic are written to tape. (Actually, one in a thousand Level 1 triggers also goes to tape for diagnostic purposes.) The first level of logic is designed to select events with at least two distinct muons that are in time coincidence with the accelerator rf signal and that have no beam halo particles among them. The second level logic insists that every event have at least two tracks that point back to the target in the non-bending (elevation) view. The Level 1 and Level 2 circuitry by itself constitutes a fairly sophisticated trigger. Nonetheless, a significant improvement in the trigger is achieved at the third level.

The third stage of the trigger imposes the requirement that the muon trajectories bear a resemblance to those of high mass pairs. Unlike some other trigger processors,[4] no mathematical computation of an invariant mass is actually made.[5] Instead, two 15 bit words are constructed, each one describing a trajectory through the C and D hodoscopes. In principle, these 30 bits could furnish an address into a 2^{30} bit memory, each location of which contains a 1 or 0 depending upon whether or not the candidate track pair is to be accepted or rejected. This is the essence of the philosophy adopted for the trigger processor described here, though the implementation does not actually use such a memory.

Level 3 Trigger

The "raw data" used in making the Level 3 trigger consist entirely of the latched signals from the C and D hodoscope X, Y, and U counters. Only those Y counters which contributed to a Level 2 trigger participate in Level 3. A three-fold coincidence of X, Y, and U counters (the result of which is also latched) creates rectangular "pads" in each hodoscope, as shown in Fig. 2, which depicts an actual event from the point of view of the third level processor.

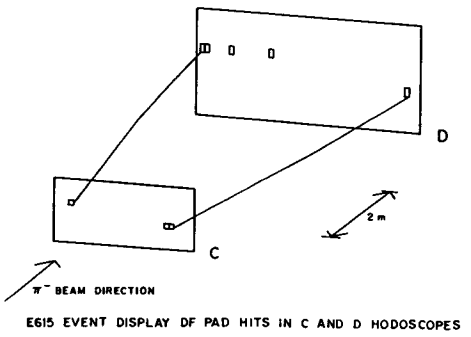


Figure 2. Detector as seen by Level 3

Given a set of struck pads, a hardware "track pair finder" assigns pairs of pads, one in the C hodoscope and one in the D hodoscope, to a candidate particle trajectory. Two such candidates are constructed, called α and β. Ultimately, each track is described by a 15 bit word. The lowest order 5 bits specify the X coordinate of the struck pad in the C hodoscope, the next 6 bits give the X coordinate of the struck pad in the D hodoscope, and the remaining 4 bits identify the Y coordinate of the C hodoscope pad. The Y coordinate of the struck pad in the D hodoscope need not be encoded because the track pair finder will associate a given C hodoscope pad with only those pads in the D hososcope belonging to Y counters that are roughly on a line with the target, in keeping with the Level 2 logic. Furthermore, the track pair

finder will not permit the α and β track pads to share a Y address in either hodoscope, in conformity with Level 1 nonadjacency requirements.

As can be seen in Fig. 2, it may be possible to construct many candidate track pairs for a given event (though only the tracks actually reconstructed using the wire chambers are shown in the figure). Accordingly, the track pair finder not only performs an encoding function, but "loops" over all possible pairs of candidate tracks, subject to the Level 1 and 2 trigger requirements. Each pair is presented in turn to Fermilab ECL/CAMAC Memory Look-Up modules (MLU's),[6] where a comparison is made with stored patterns. The track pair finder also knows when to quit. That depends not only upon the outcome of each comparison, but on the number of track pair candidates that have been constructed and compared so far, for a given event.

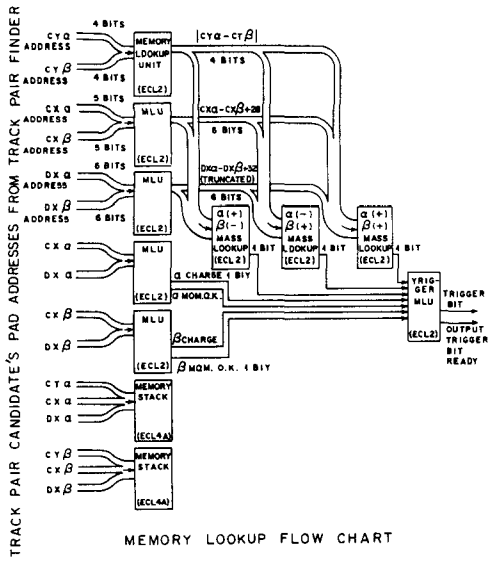


Figure 3. Memory look-up sequence

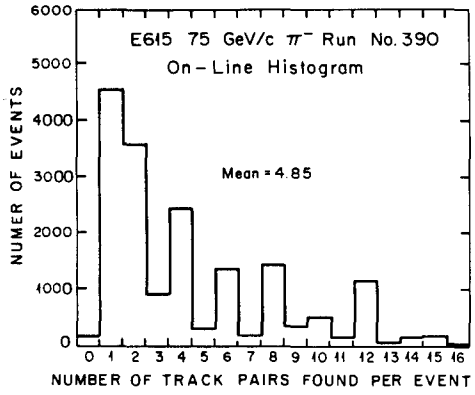


Figure 4. Typical track pair multiplicity

The MLU's are interconnected as shown in Fig. 3, though it should be emphasized that once the two words describing a track pair candidate are made available, the bits may be manipulated in any fashion desired, simply by re-cabling and re-loading the MLU's. In this experiment it has proven convenient to assign a charge to each track, to determine whether or not each track's curvature is compatible with a physically allowed momentum, and to determine whether or not the pair of tracks has a topology similar to that expected for high mass di-muons.

Charge is specified using a two-dimensional matrix that has the C hodoscope pad's X counter address providing one index and the D hodoscope pad's X counter address providing the other index. The elements of the matrix used to load the appropriate MLU's are set to 1 for positively charged tracks, and to 0 for negatively charged tracks.

A momentum selection ("good" or "bad") is based on another matrix with the same indices, loaded with ones in kinematically allowed regions and zeroes elsewhere.

Invariant mass is strongly correlated with the plan view track separation at the C hodoscope (ΔCX) and the track separation at the D hodoscope (ΔDX). The high mass region in the ΔCX vs. ΔDX plane varies according to the vertical separation of the tracks, so that a

number of two-dimensional arrays must be specified (one for every $|\Delta CY|$) in order to load a Mass MLU. Separate Mass MLU's are needed for track pair candidates in which track α is assumed to have one charge and track β another, since the 16-bit-wide input to the MLU is exhausted by the ΔCX, ΔDX, and $|\Delta CY|$ inputs.

The Trigger MLU at the end of the chain is easily programmed to set the output "trigger bit" to 1 or 0 depending on the state of its seven input bits. An "output ready" signal tells the track pair finder when the trigger bit is valid so that it may decide whether to assemble the next track pair candidate or quit. An internal counter may be set so that no more than 1, 2, 4, 8, 16, or 32 track pair candidates are constructed. (Two 32-word-deep Memory Stack modules record, for diagnostic purposes, the 30 bit description of each track pair candidate presented to the MLU's.) During data-taking, the stack limit has been set to 16. Of course it is possible that no track pair candidates at all can be constructed for a given event, or that all possible track pair candidates are exhausted before the stack limit is reached. In the first case the event is rejected. In the latter case the event is rejected if none of the candidate pairs satisfied the Trigger MLU, and accepted otherwise. We have chosen to reject events when the stack limit is reached, regardless of the state of the trigger bit associated with the candidates tested. A representative histogram of the number of track pairs found per event is shown in Fig. 4. (The entires at 0 and 16 are associated with the prescaled Level 1 trigger.)

Electronics Design, Construction, and Testing

Most of the trigger electronics was designed and constructed at Princeton using ECL 10,000 series chips. The Level 1 and Level 2 circuits were built using a Multiwire technique.[7] The track pair finder (5 boards) and U counter-to-pad fanout modules (4 boards) were wired using the insulation displacement technique.[8] This permitted somewhat faster wiring than the wire-wrap method, but at the price of a greater susceptibility to bad contacts. Nevertheless, since having been debugged, the entire trigger processor has operated with only two or three failures during the last nine months, those failures having been single dropped bits in the MLU's. The veto hodoscope logic was designed and built at the University of Chicago. The only commercial electronic modules employed in the trigger are the phototube discriminators (LeCroy 4416's), a few latch modules (for the A, B, E, and F counters), and assorted NIM modules.

Operational Characteristics

The trigger processor has made possible a marked reduction in the number of events written to tape per beam spill while significantly enriching the sample of interesting events recorded on each tape.

Some typical rates per spill ($\sim$ 14 sec in duration) are tabulated below for runs with a 75 GeV/c π^- beam (400 GeV/c protons) and a 250 GeV/c π^- beam (800 GeV/c protons).

Typical Rates per Beam Spill

	75 GeV/c	250 GeV/c
protons on primary target	4.9×10^{12}	3.6×10^{12}
π^- on experiment's target	2.7×10^{9}	2.4×10^{9}
hits per downstream hodoscope	$\sim 10^{8}$	$\sim 10^{8}$
Level 1 triggers	2.4×10^{4}	6.3×10^{4}
Level 2 triggers	1.6×10^{4}	3.3×10^{4}
Level 3 triggers	1390	1260

The ratio of Level 1 to Level 3 triggers varies between 20 and 50. The deadtime of the experiment varies between 30% and 40%, most of it attributable to the on-line computer and the A/B veto rate. The deadtime introduced by the Level 3 logic is only about 2.5 μsec per Level 2 trigger. The minimum time interval between the start of the Level 3 logic and the assertion of the "trigger bit" is 570 nsec, about 370 nsec of which is spent in the sequence of memory look-ups. If no track pair candidates can be found, a minimum of 90 nsec passes before an abort is issued.

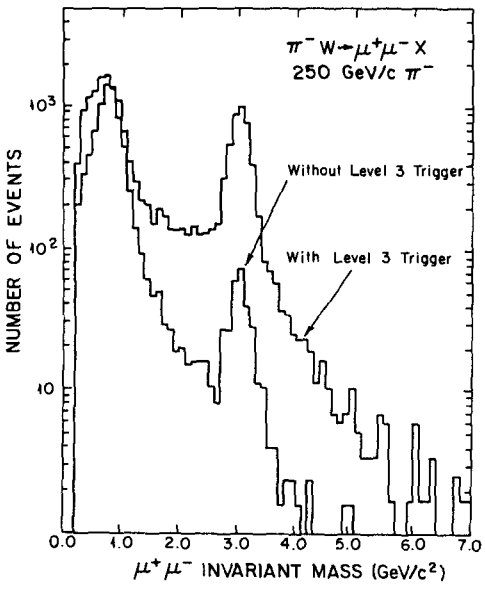


Figure 5. Analyzed data from two runs

Another measure of the effectiveness of the trigger processor is displayed in Fig. 5, which compares data taken with and without the Level 3 trigger, at a π^- beam momentum of 250 GeV/c. The $\mu^+\mu^-$ invariant mass spectrum reveals a suppression of events below the J/ψ relative to those above by more than a factor of 15 when the Level 3 trigger is employed. Both data sets have been normalized to 100,000 triggers written to tape. Minimal event selection criteria were imposed in the analysis of the data to make the comparison a fair one.

Summary

A fast, ECL-based trigger processor featuring a specialized, hard-wired "track pair finder" and modular, programmable memory look-up units has been implemented in a di-muon experiment at Fermilab, and has achieved better than an order of magnitude reduction in trigger rate and rejection of background events.

References

1. For brief descriptions of the experiment and its goals, see A. J. S. Smith, "Production of μ-Pairs in the Forward Direction: Fermilab Experiment 615," in Lepton Pair Production (Proceedings of the Moriond Workshop, Les Arcs, Savoie, France, January 25-31, 1981), pp. 141-147; W. C. Louis, "Status of Experiment 615 at Fermilab: Production of Muon Pairs in the Forward Direction," in Proceedings of the Drell Yan Workshop, Fermilab, October 7-8, 1982, pp. 271-277; W. C. Louis, "Status of Experiment 615 at Fermilab: Production of Muon Pairs in the Forward Direction," in Gluons and Heavy Flavours (Proceedings of the XVIIIth Recontre de Moriond, La Plagne, Savoie, France, January 23-29, 1983), pp. 407-412.

2. S. D. Drell and T.-M. Yan, Phys. Rev. Lett. 25, 316 (1970).

3. $\cos\theta^*$ gives the polar angle of the μ^+ with respect to some suitable axis, such as the beam direction, in the mu-pair c.m. frame.

4. See, for example, H. Areti et al., Nucl. Inst. Meth. 212, 135 (1983).

5. The basic scheme of this trigger processor is due to Kirk T. McDonald.

6. E. Barsotti et al., IEEE Trans. Nucl. Sci. NS-26, 686 (1979); ECL/CAMAC Trigger Processor System documentation, Fermilab TM-821, 2nd ed.

7. Multiwire is a trade-mark of Kollmorgen Corp., Photocircuits Division, Glen Cove, New York 11542.

8. Gold-plated pins supplied by Robinson-Nugent, Inc. were used with "kluge" boards fabricated at the Univ. of Chicago.

A REVIEW OF TRIGGERS AND SPECIAL COMPUTING HARDWARE FOR THE FERMILAB FIXED-TARGET PROGRAM

Sergio Conetti

Institute of Particle Physics and McGill University
Montreal, P.Q., Canada, H3A 2T8

Introduction

The design and implementation of a "trigger", that is the selection of a particular process from the many ones induced by the interactions between elementsry particles, have always been among the most crucial aspects of particle physics experiments. In the case of fixed target experiments in particular, the rate at which the reactions under study are produced, rather than being limited by the accelerator's luminosity, is very often determined by the achievable trigger rste and the consequent dead time produced by the data acquisition system.

Thanks to technological advances in the field of micro-electronics, new and powerful tools have recently become available to upgrade the capabilities of the triggering systems. The community of particle physicists has welcomed the advent of the microprocessor which, together with the ever-increasing availability of integrated circuits with a more and more favourable cost/performance ratio, has allowed the implementation of very sophisticated, multi-level schemes of event selection.

The first comprehensive coverage of the marriage between micro-electronics and particle physics was given by the 1981 CERN "Topical Conference on the Application of Microprocessors to High Energy Physics". In his contribution to the Conference[1], Tom Nash, in addition to a more detailed discussion of the required performance and general properties of "intelligent triggers" in the fixed target environment, reviewed the activity in such a field connected with Fermilab experiments. In that paper thirteen processing devices, employed in eleven different experiments, were described: three years after the CERN Conference the rapid growth in the field is clearly indicated by the presence of thirty different processors, employed in twenty-four fixed target experiments at Fermilsb. A more detailed breakdown can be obtained using the Fermilab's Situation Report tables, where approved experiments are listed under the categories: "Experiments completed during the past year", "Experiments in progress or to be set-up within a year" and "Other approved experiments". When the Summer '83 Situation Report is employed, and after excluding purely bubble chamber or emulsion experiments, one recognizes that in the three categories, respectively, 7 out of 15 (47%), 6 out of 7 (86%) and 11 out of 17 (65%) experiments employ one or more processing devices for the online event selection. In the presence of so many different systems, and before describing them in more detail, it is useful to introduce a classification scheme into which the various devices can be accommodated.

Classification of Processors

Like numerous other attempts, in various scientific disciplines, to find some underlying order in an ensemble of complex systems, the scheme presented here will involve some arbitrariness and expeditious overlooking. Trying to force all of the existing units into a rigid pattern will cause the appearance of hybrids, hermaphrodites, chimerae and other monsters. These reservations notwithstanding, there are some definite advantages in introducing a classification scheme, even if not fully satisfactory nor necessarily unique, so we will proceed towards such a goal. As a first step, processing devices will be grouped into Fixed Flow and Variable Flow: a Fixed Flow processor will be one for which the sequence and total number of processing steps is always the same, independent of the features of individual events; for the second category, the processing flow will obviously vary with different events. Fixed Flow processors will be further sub-grouped into Logical (FFL) and Arithmetical (FFA), while the Variable Flow ones will be of the Data-driven (VFD) or Program driven (VFP) type.A more detailed description of the four major groups just introduced follows.

Fixed Flow Logical. When an interaction takes place, the data most immediately available from a detector is in the form of bit strings,obtained by imposing a "binarising" threshold to analogue data , carrying information related to pattern of hits in wire chambers or counters.An FFL processor would consist of hard-wired or programmable logic elements capable of examining the patterns to decide upon the acceptability of each event received.The operation could involve sequential steps or be completely parallel. As the most extreme case for the ultimate processor, one can think of achieving a very fast (20-50 nsec) and arbitrarily sharp event selection by presenting in parallel all the information recorded by a suitable detector to a huge look-up memory: performing different experiments would only require reprogramming the memory ..... The only drawback of such an approach is that, even for a relatively simple detector, the number of bits needed in the memory is of the same order as the number of molecules in the galaxy. A more down-to-earth and commonly employed category of devices is represented by the "shifters", for which speed of execution is traded off versus number of components. Such devices make use of some shift register to sequentially examine partial sections into which the complete bit pattern has been broken. Typical cycle (step) times for FFLs are 20-50 nsec.

Fixed Flow Arithmetical. These devices involve the encoding (often done via priority encoders)of wire/counter hits into coordinates and the fast digitization of analogue data. Arithmetical (and logical) operations are performed on the data as it cascades through the various stages of the processor. In order to obey the requirement of fixed flow, some selection criteria (e.g. largest, leftmost)is imposed in the case of multiple entries ,such as more than one hit in a detector plane, etc. In this way the sequence of operations is completely pre-determined and can be built into the processor structure. Typical single cycle times are 50-100 nsec.

Variable Flow Data-driven. VFD processors are the obvious evolution of the FFA ones (or is the FFA just a special case of the VFD?). For this type of processor, an attempt is made to digitize, for every event to be processed, all the relevant data and to perform the desired computations on all the different combinations arising from multiple entries. To achieve this, the processor must be capable of executing single or nested loops, conditional jumps, etc. The logic of such a processor becomes more and more elaborate, but the potential for fast computations of complex algorithms is very large. Given the sophisticated structure, it is advisable to develop such processors with a modular approach and with a well defined mechanical and signal standard. As we will see, two such systems have been independently developed for fixed target experiments at Fermilab.Cycle times are usually the same as for FFAs.

Variable Flow Program-driven. In this category we include all the systems that employ, as their central intelligence agency, "conventional" computers, driven by a stored program. In most instances, the systems are centered around one or more microprocessors, often developed ad hoc to optimise the performance with respect to the required application, yielding some very unconventional and most interesting architectures. Typical cycle times for such devices are in the hundreds of nanoseconds.

Processors at Fermilab

The large set of processors developed for the fixed target experimental program at Fermilab is summarized in Table 1. The four major categories introduced in the previous section are well represented, with some experiments exhibiting more than one processor in their set-up. E-605 and E-705 in particular, channel the flow of incoming events through an FFL-VFD-VFP sequence, so achieving a powerful multi-level trigger (it should nevertheless be mentioned that E-690 aim is an equally effective event selection through the exclusive use of a very sophisticated VFD device). Performance figures describing the processing speed and rejection power of each system are deliberately not included in the table: such entries might lead to perform a comparison among different systems, which is somewhat meaningless when a large set of boundary conditions is not taken into account. In short, one could say that each processor was designed to attain the speed and rejection power that were needed for the particular application.

The functions performed by the various systems cover a wide spectrum, reflecting the ample degree of differentation exhibited by fixed target experiments. One particular application nevertheless stands out unmistakably: as many as twelve experiments made use of their processors to evaluate the invariant mass of a set of particles, described by their

Table I.

1) EXP	FFL	FFA	VFD	VFP	2) REF	STATUS	3) FUNCTION	MAIN CHARACTERS
326	X				C,2	C	dimuon mass	Sumner,Halling,Isaila
361	X				3	C	synch. rad. pattern	Dworkin
400(687)				X	CG,24	R	high p_t kaon	Lebrun,Gaines,Droegge
516(691)			X	X	C,4 15	C	missing mass event reconstruction	Nash,Barsotti,Bracker Martin,Shepard,Luste
537(705)			X		C,14	C	dimuon mass	Areti,Conetti
605	X		X	X	C G,22 G,26	R	track finding high p_t,high mass event reconstruction	Glass Kaplan Rutherfoord
609/683	X			X	4 27	C D	multi-jet pattern event reconstruction	Erwin,Nelson Thomson
612	X				C	C	track reconstruction	White
615			X		G,17	R	dimuon mass	Greenhalgh,MacDonald
617/731		X			9	R	K^0 (6 photons) mass	Gollins,Winstein
621	X	XX			5	R	photon clusters K^0 mass	Heller Thompson
623		X			C,10	C	$\phi\phi$(4K) mass	Fenker,Green
665	X				7	D	high Q^2 muon	Pitt,Kobrak
672			X		23	R	dimuon mass	Crittenden,Smith
687			X	X	19	D	high p_t kaon	Lebrun,Gaines
690			X		G,20	D	event reconstruction	Knapp,Sippach,Avilez
691		X			15	D	high forward mass	Tagged Photon Collab.
704				X	28	D	Λ^0 mass	Birsa,Villari
705	X		XX	X	6 16 30	D	high p_t photon dimuon,di-γ mass event reconstruction	Rameika,Wegner,Lynch Conetti,Tzamarias Haire,Kuchela
715	X				8	C	trans.rad.pattern	Leningrad
732		X			11	D	Ξ^0 mass	Sheaff
743				X	29	D	vertex reconstr.	Rome

NOTES

1) Bracketed numbers represent experiments performed by the same group that appear again in the table with some additional system. Number separated by slashes(/) signify that the same processor was used in two experiments.
2) C : the processor was described in the CERN '81 Conference (ref. 1). G : a contributed paper is contained in these Proceedings.
3) C : completed. R : running. D : development

measured momenta (or energies) and directions. Be the sought-after final state a di-muon, a K^0, or any other, it can really be said that the advent of processors offered the most effective solution to a problem that until then could only be partially and insufficiently solved.

A description, necessarily very brief, of the individual systems grouped into the four major categories follows.

Among the Fixed Flow Logical processors, the first two we encounter, E-326 and E-361, are of the "shifter" type. The very effective di-muon processor for E-326 has already been described in the literature [1,2]: in a set-up where the detector and the processor appear to be well integrated parts of a single body, counter hit patterns are identified and compared, via a shift register, to the pre-computed set associated with high mass di-muons. E-361[3] had a simpler requirement: particles associated with the beta-decay of Λ hyperons produced, in a Xenon chamber, a characteristic pattern (fig.1), induced by the particles themselves plus a photon from synchrotron radiation. A processor (fig. 2) was built to recognise the presence of the correct number of AND and XOR coincidences, counted while shifting the data from the Xenon chamber two planes.

Skipping the E-605 front end processor and the E-612 track finder, already presented in reference 1, we describe in more detail the E-609 calorimeter matrix [4]. The calorimeter was composed of 132 towers, each one giving an output proportional to the energy deposited times the sine of the tower polar angle. A trigger was formed when the sum of the energies in a group of towers was above a threshold energy. The sum was obtained by using an analogue adder and a cross-point matrix (fig. 3).A given module (fig. 4) performed 40 analogue sums in parallel (through the cross-point matrix): specific triggers were programmed by connecting the desired cross-point resistors, allowing an easy reconfiguration of the system.

Still in the realm of calorimeter processors and somewhat similar in scope, although very different in actual realization of the system, are the E-621 and E-705 cluster finders. Both these experiments process the analogue data from an electro-magnetic glass calorimeter to recognise and count clusters of energy deposition, likely associated with incoming photons or electrons. The E-621 ECL processor [5] looks for neighbours on two adjacent edges of each glass block hit (fig. 5): if none of the neighbours is hit, then it is called a cluster edge. The total number of identified clusters is available as the processor's output. The E-705 cluster finder [6] compares the energy deposition in every block of a scintillating glass calorimeter with the one in its eight neighbours, looking for local maxima which are identified as cluster centers. The total energy of each cluster (peak + neighbours) is weighed by its radial position, to identify high p_t photon candidates. A digital list of cluster energies and positions is also prepared, to be analyzed by the next level of triggering.

The new, high energy, muon beam at the Fermilab Tevatron will be utilized by E-665. The experimental requirement of recognising muons scattered with large Q^2 will be satisfied by a processor [7], designed to analyze the information from 4 banks of X-Y proportional tubes imbedded in the spectrometer iron walls. Track finding will be done separately in each of the two views, using fast memories to recognise hit patterns appropriate to scattered muons coming from the target. In each view, each wire in the first chamber is the endpoint of one road. Each road is associated with a 4096 x 1 ECL memory, whose twelve address lines are connected to the appropriate chamber wires. The memories will be downloaded with the acceptable hit patterns, pre-computed through extensive Monte Carlo simulations. Finally, we mention the cluster finder developed for E-715[8] to recognise and count clusters in a system of proportional chambers forming the active component of a transition radiation detector.

Similar to FFLs, Fixed Flow Arithmetical processors are usually designed and hardwired having in mind a very specific application, so that their configuration is experiment-dependent and not easily transportable from one set-up to another. On the other side, since FFA processors, as mentioned earlier, can always be considered as a special case of the VFD ones, we will find in our list some FFA devices based on modules developed for the VFD systems.

From Table 1, we first encounter E-617 and its follow-up (E-731), a pair of CP violation experiments that require in their trigger, to recognize the $K^0 \rightarrow 3\pi^0$ decays for which all the photons are contained

within the detector[9]. To this goal, the information from an 800 blocks lead glass calorimeter (fig. 6) is treated in a processor that combines digital and analogue calculations to attain maximum speed and efficiency. For every event, the quantities computed are

ΣE_i (analogue sum), $\Sigma x_i E_i / \Sigma E_i$ (analogue sums, digital division)

and $\Sigma r_i^2 E_i \cdot \Sigma E_i$ (analogue sums, digital multiplication)

where E_i is the energy deposition in the i-th block, and x_i, r_i its transversal and radial coordinates. Suitable cuts on the above quantities provide, in $\sim$150 nsec, the desired signal.

Another CP violation experiment, E-621, was already mentioned earlier for its photon cluster finder, which is used in conjunction with an arithmetical processor to select $K^0 \rightarrow \pi^+\pi^-\pi^0$ decays in the presence of a competing background [5]. Still in the setting up stage, the experiment features two processsors utilizing the information from two different detectors (proportional chambers and counters) to evaluate the same quantity. Using respectively a set of look-up tables (the Lecroy version of the Fermilab ECL-CAMAC system, see below), or a home-built subtract and divide circuit, the two processors compute the quantity

$$(B_{left} - A_{left})/(B_{right} - A_{right}) \quad , \text{ where } A,B_{left(right)}$$

represent the left (right)-most hit from the two planes of detector A and B (fig. 7). Such quantity is related to the mass of the detected Vee decay. Not too dissimilar in concept, but more complex because of the presence of four candidate kaon tracks, the E-623 $\phi\phi$ mass processor has already been described elsewhere [1,10].

The identification of yet another invariant mass, this time the Ξ^0, is the aim of E-732.[11] The exact formula for the mass evaluation, shown in fig. 8, involves a sequence of arithmetical operations. The proposed processor is hard-wired to diligently go through the required set of operations, making the largest possible use of parallel processing, as detailed in fig. 8. Finally, the current plan of the E-691 collaboration (father of the Fermilab ECL-CAMAC system), is to reconfigure their E-516 trigger processor into a simpler system suitable to satisfy the requirements of the new experiment.

In addition to the rich choice of application dependent projects described so far, a very important feature of the Fermilab fixed target program is the presence, as mentioned earlier, of two very powerful general purpose systems, representing the global approach to trigger processing that is the distinctive character of Data-driven, Variable Flow devices. The first of the two systems, the Fermilab ECL-CAMAC, has already been described in detail [1,12]. The first implementation for which it was developed, the recoil trigger for E-516[13], was followed by the E-537 di-muon mass processor[14]. The modules from both of E-516 and E-537 processors are going to be used again in a new round of Tevatron experiments: E-691 will reconfigure the recoil processor to achieve a forward mass evaluation[15], while E-705 will implement an improved version of the basic di-muon processor[16]. Another experiment, E-615, has already run with a hybrid home-built/ ECL-CAMAC di-muon processor, described in these Proceedings[17].

The ECL-CAMAC system has been very successful: one can foresee that more and more applications of it will appear in the future, based on its commercial version, now produced and marketed by Lecroy Research Systems. The two most fundamental building blocks of the ECL-CAMAC system, the Stack, used as a buffer and supplier of operands and intermediate results, and the Memory Look-Up, the general purpose operator, have already been produced by Lecroy, together with other general purpose logic and arithmetic modules. It appears that all the elements exist to assemble sophisticated processing systems similar to the ones developed in the original ECL-CAMAC standard, although with some notable differences. An important element of the Fermilab system was the Do Loop Indexer, a module capable of requesting, as soon as they become available, all possible pair combinations from two lists of elements which are filled concurrently with the pair extraction process. Such property yielded a faster processing capability as compared to the one given by a "nested loop" logic, where the outer index cannot be advanced until all the elements belonging to the inner index have become available. The Lecroy Corporation has decided[18]

not to produce such a module, providing instead the Stack with "nested-loop" capabilities. Such a choice represents a very reasonable trade-off of potential processing speed (not always exploitable) versus system simplicity. Another important feature of the commercial version that will greatly facilitate its adoption is the packaging into modules residing in standard CAMAC crates and obeying the CAMAC protocol. Various groups are already using or planning to use the Lecroy ECL-CAMAC modules: in addition to the already mentioned E-621, E-705 and E-687 will implement respectively a di-photon addition to the di-muon trigger and a front end processor to the M7 computer[19].

The second general system for VFD processors, the Nevis Lab data-driven pipeline, is described in these Proceedings[20] as well as in earlier publications[21]. The approach is even more ambitious than the ECL-CAMAC one: in its ultimate configuration, the system is expected to perform a complete reconstruction of all the beam interactions in the target, without introducing any dead-time. A system of such a performance is already running at 10^5 interactions/ sec. for Brookohaven experiment 766, while Fermilab E-605, as described in these Proceedings[22], is utilizing the same modules to look for high p_t particles and high mass pairs in a low multiplicity environment. E-690, as described in ref. 20, will demand the ultimate performance from the Nevis Lab project.

So far all the systems in the VFD category have represented implementations of the ECL-CAMAC or the Nevis efforts; there is one notable exception, the E-672 di-muon mass processor[23], whose self- explanatory block diagram is shown in fig. 9. In what turns out to be a very elegant development, the designers have given up some of the speed attainable when maximum concurrency is demanded, to realise instead a system which is particularly compact and easy to commission. The E-672 algorithm for evaluating di-muon masses is very similar to the one utilized by the E-537 ECL-CAMAC processor, so that a comparison of the two systems is meaningful. Trading speed for simplicity, the E-672 system needs about 5 μsec to process the ideal event (only the 2 muon tracks present) as compared to 1 μsec for E-537. The E-672 system nevertheless is totally contained in $1\,1/2$ crates (versus $\sim$ 5 crates for E-537) and required a total manpower of about 2 man/years, small figure when compared to the large effort involved in the ECL-CAMAC system.

The last category, the Variable Flow, Program-driven computer-like devices, has not had a strong representation at Fermilab, especially when compared with the CERN program. Until recently, the only important presence in the VFP category was the M7, a stored program device designed to optimally perform track reconstruction algorithms. The device has proved its value, having already been used in E-401[1], E-400 (ref. 24, these Proceedings), with a projected utilization for E-687[25].

The E-605 project, presented at this Conference[26], consists of hanging a fast processor onto a DEC UNIBUS, to analyze and filter events between spills, a procedure similar to the one foreseen for E-683[27]. The next two entries, E-704[28] and E-743[29], merely represent the intention to transport to Fermilab two microprocessor systems currently in use at CERN, so that the only major new development is the E-705 project[30]. The problem facing this experiment is an expected trigger rate in excess of 200 events/sec, with $\sim$ 2000 words/event and a spill of more than 10 seconds. In the presence of a pre-existing heavy dotation of CAMAC read-out modules, it was necessary to develop a system capable of reading out the CAMAC crates in less than 1 msec, inclusive of the several hundreds of microseconds required for data digitization. To this goal, a set of smart crate controllers, built around a Motorola 6809 microprocessor, is being constructed: the controllers will be loaded with a list of CAMAC operations, that can be executed within a few microseconds after reception of a trigger. The controllers, daisy chained into a subset of parallel data channels (fig. 10), will unload the data into a set of FIFO memories, residing into a VME crate, deep enough to handle the worst case instantaneous trigger rate. An array of Single Board Computers (based on the 68000 microprocessor) will sit in the VME crate to sort, compress and filter the events, and eventually to transmit them to a CAMAC memory for conventional recording. The system, currently under construction, is scheduled to run in 1985.

One final entry, E-516, appears in the VFP column of table 1. Rather than an on-line processor, the entry acknowledges the University of Toronto effort in the construction of a system of eight IBM-168 emulators (168/E), that were run to analyze a large fraction of the E-516 raw data tapes.

Conclusions

The impressive amount of processors of all types employed in a majority of the current and future experiments proves that such devices should now be considered as a standard tool available to the researchers. In the presence of such a variety of developments, one still has the feeling that there is not a "best system". For each particular case, the choice will depend first of all on the requirements of the given experiment and secondly on the experimenters' ambition and availability of manpower and/or commercial hardware. There is no doubt however that, in order to optimize the performance of any given system, the processor's structure and requirements should be integrated, at the earliest possible stages, in the overall design of the experimental set-up.

Apologies and Acknowledgements

I apologize for any omissions or misrepresentations which may have resulted because of the rapid changes and often minimal documentation characteristic of the material here covered. I would like to thank the Organizers of this Conference for their invitation to prepare the present review. Travel and living expenses at Fermilab and Guanajuato were covered by a grant from the Natural Sciences and Engineering Research Council of Canada.

References

1. T. Nash, Proceedings of the Topical Conference on the Application of Microprocessors to High Energy Physics Experiments, CERN 81-07, p.132 (1981).
2. A. Halling, M. Isaila and R. Sumner, IEEE Trans. Nucl. Sci. 29, 437 (1982).
3. J. Dworkin, University of Michigan, Ph.D. thesis, February '83.
4. K.S. Nelson and A.R. Erwin, IEEE Trans. Nucl. Sci. 30, 146 (1983)
5. R. Handler et al., Phys. Rev. D25, 635 (1982) and T. Devlin, Rutgers University, private communication.
6. R. Rameika, Fermilab, private communication.
7. R . Pitt, Mass. Inst. of Tech., private communication.
8. T. Lach, Fermilab, private communication.
9. B. Winstein, University of Chicago, and G. Gollins, Princeton University, private communication.
10. H.C. Fenker, D.R. Green, S. Hansen and T.F. Davenport, Fermilab Pub. 82/62-EXP. (1982).
11. M. Sheaff, University of Wisconsin, private communication.
12. E. Barsotti et al., IEEE Trans. Nucl. Sci. 26, 686,(1979).
13. J. Martin et al., Proceedings of the Topical Conference on the Application of Microprocessors to High Energy Physics Experiments, CERN 81-07, 164 (1981).
14. H. Areti et al., Nucl. Instr. and Meth. 212, 135 (1983).
15. J. Martin, University of Toronto, private communication.
16. S. Conetti, McGill University and S. Tzamarias, University of Athens, E-705 internal note.
17. J. Greenhalgh, these Proceedings.
18. L. Levitt, Lecroy Research Systems, private communication.
19. P. Lebrun, Fermilab, private communication.
20. C. Avilez, these Proceedings.
21. W. Sippach, G. Benenson and B. Knapp, IEEE Trans. Nucl. Sci. 27, 578 (1980), and H. Cunitz, R. Hsiung, B. Knapp and W. Sippach, Proceedings of the Conference on Instrumentation for Colliding Beam Physics, SLAC-250, June 1982.
22. D. Kaplan, these Proceedings.
23. R. Crittenden, Indiana University, private communication.
24. P. Lebrun, Fermilab, these Proceedings.
25. P. Lebrun, Fermilab, private communication.
26. J. Rutherfoord, these Proceedings.
27. M. Thomson, University of Wisconsin, private communication.
28. A. Penzo, INFN Trieste, private communication.
29. S. Reucroft, CERN, private communication.
30. S. Conetti, M. Haire and K. Kuchela, McGill University internal note.

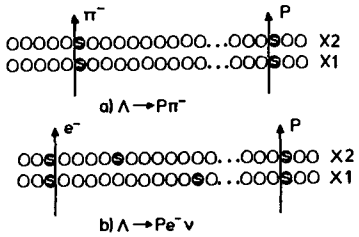


Fig.1. E-361 synchrotron radiation pattern.

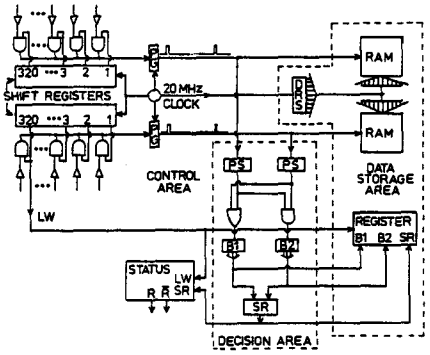


Fig.2. Block diagram of the E-361 processor.

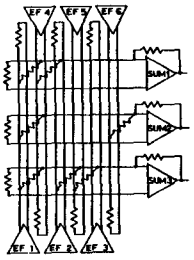


Fig.3. E-609 matrix adder.

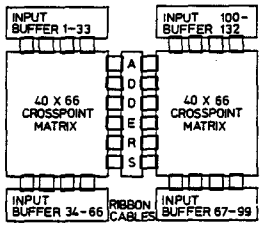


Fig.4. Layout of the E-609 processor components.

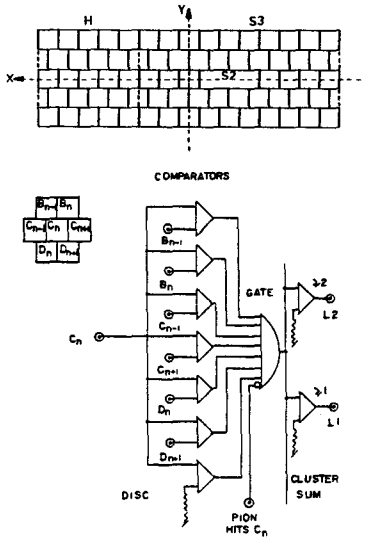


Fig.5. E-621 cluster finder.

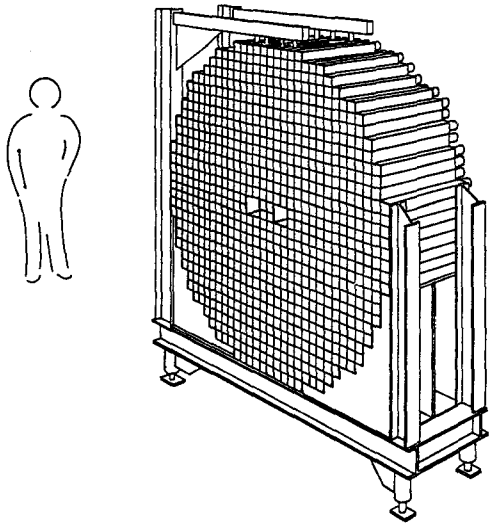


Fig.6. Glass block array of E-617.

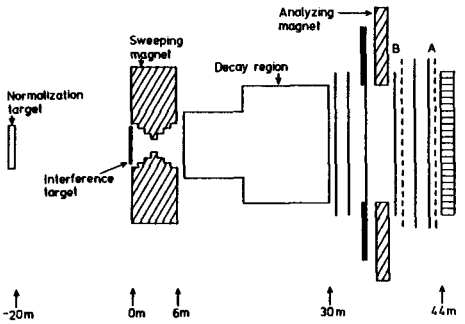


Fig.7. E-621 layout.

$$p_+p_-\theta^2 \simeq \frac{1.96\left\{\left[(\bar{x}_2+x_2)(z_1+z_B)-(\bar{x}_1+x_1)(z_2-z_B)\right]^2+\left[\left\{(\bar{y}_1+y_1)-(\bar{y}_2+y_2)\right\}(z_B-z_D)\right]^2\right\}}{\left\{x_2(z_1-z_D)-x_1(z_2-z_D)\right\}\left\{(\bar{x}_2(z_1-z_D)-\bar{x}_1(z_2-z_D)\right\}}$$

TIME SLOT	OPERATION	NUMBER OF SIMULTANEOUS CALCULATIONS	CHIP SIZE	ESTIMATED TIME (nsec)
1	ADD	4	8 + 8	50
2	MULTIPLY	8	8 x 8	100
3	SUBTRACT	4	16+16	50
4	MULTIPLY	3	12 x12	100
5	ADD	2	16 +16	50
6	MULTIPLY	2	12 x12	100
7	COMPARE	1	16 +16	50

Σt=500 nsec

Fig.8. Algorithm for the E-732 processor.

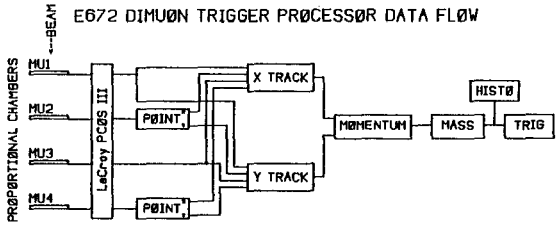


Fig.9. Block diagram of the E-672 processor.

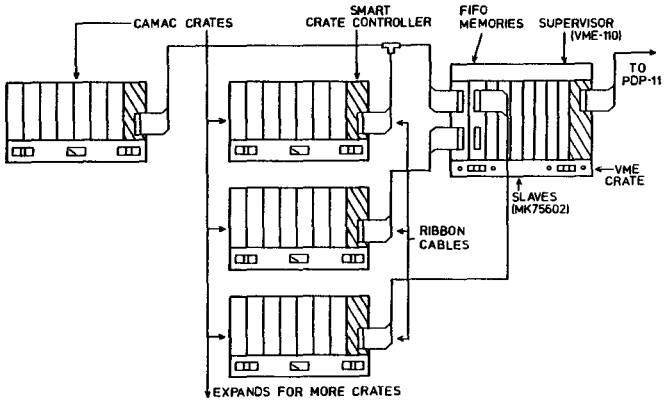


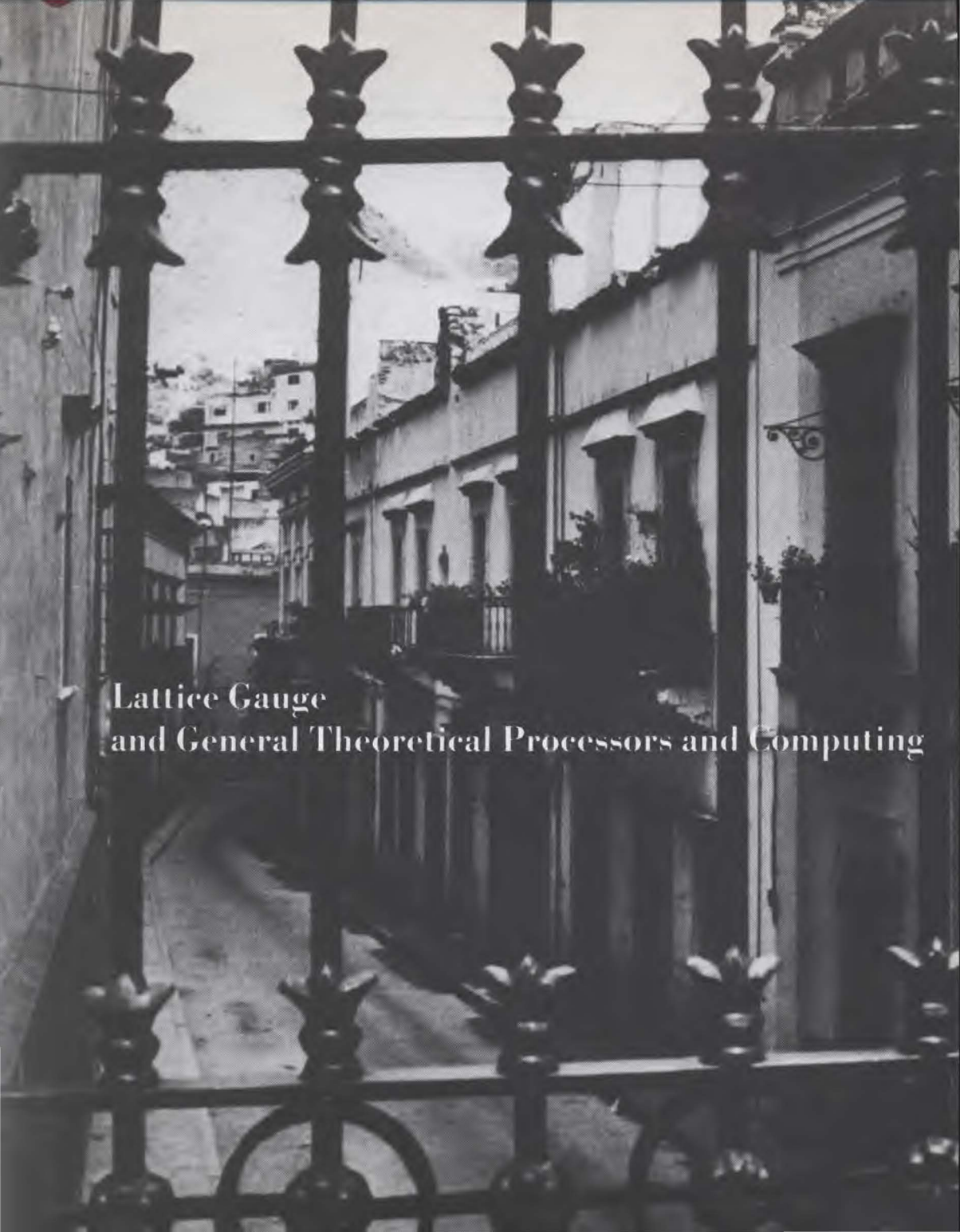
Fig.10. The E-705 data acquisition system.

QUESTIONS AND ANSWERS

Q: Do you write raw data or calculated data on tape?

R. Poutissou

A: Both.


Lattice Gauge
and General Theoretical Processors and Computing

[Photograph of Guanajuato by Eduardo Rangel Cerrillo (Guayo) of Guanajuato.]

PARALLEL SUPERCOMPUTERS AND LATTICE GAUGE THEORIES

Anthony Terrano
Physics Department
Columbia University
New York, NY 10027

Numerical studies of lattice gauge theories are severely limited by the power of the computers which can be brought to bear upon them. This limitation has a number of consequences. First, the number of samples which can be taken in a given calculation will be small, resulting in, at best, a large statistical uncertainty. Indeed, with limited resources, it may impractical to measure the autocorrelations of the quantity being calculated with the result that the statistical errors in a calculation are simply unknown. Secondly, the number of different values of the parameters in the theory which can be investigated is limited, making a thorough study of systematic effects difficult. Perhaps worst of all, there is little possiblity of studying algorithms or alternate formulations of the problem. In order to carry out a definitive study, we should be able to spend 75% of our resources learning the nature of the system and perfecting the procedures for studying it and have the remaining 25% suffice to do a high statistics study.

Of course, this limit is not imposed by the computers themselves, but by the finite amount of money which we can realistically hope to be able to spend. There are a variety of approaches for relaxing this constraint. One can try to find "free time" on unused or partially used computers, or try to increase the level of financial support for computational physics. Norman Christ and I have pursued the approach of trying to increase the number of floating point operations which each dollar will buy. Surprisingly, there isn't a lot of difference between currently available computers when they are rated by the number of flops per dollar: a Cray-I provides about 10flops/$; a VAX somewhat less at 2flops/$; an IBM-PC (with a floating point coprocesser chip added) also about 10flops/$.

The starting place for a more powerful and cost efficient machine are the microprocessors which are generically refered to as a "VAX on a chip". These computers will typically execute 1 million instructions per second and cost on the order of $100. However, to study lattice gauge theories we need millions of floating point operations, not integer instructions, per second. The next building block are VLSI floating point arithmetic chips. The first such units were made by TRW and include a 22+22 bit adder with a cycle time of 125ns, or 8Mflops. These chips use a special format consisting of a 16 bit significand and a 6 bit exponent. A comparable multiplier can be assembled from a 16x16 bit integer multiplier coupled with an integer adder for the exponents. An arithmetic unit built around these two units could possibly perform 16 million floating point operations each second, at a price in the neighborhood of $500. These chips were the state of the art when we began our design. Since then, however, a chip set which performs full 32 bit arithmetic and runs at the same speed has become available for

about $1000. The final element is of course fast memory, which as the density per chip increases becomes more economical each year. With these elements, the outlines of our overall strategy become clear: we can build inexpensive units which will operate in the 10Mflop range; we need to find a way to bring (indefinitely) many such units to bear upon a single calculation. Our supercomputer will consist of an array of single board computer/array processors.

ARCHITECTURE

To begin, I will describe the architecture of the system. Each element of the array consists of a processor and memory. However, it is useful to think of the system as an array of memories joined to one another by the processing elements (Figure 1). The memory nodes are mapped uniformly onto the x-y plane of the problem: each node is associated with a specific rectangular region in the x-y plane. The regions cover the entire plane and have the same size, shape, and orientation. Note that each of the regions may include several points in the x-y plane of the lattice. The data associated with the lattice point (x,y,z,t) is stored in the memory associated with the region in the x-y plane containing the point (x,y).

The interconnections provided between the elements of the two-dimensional array are arranged to provide direct communication between all nearest-neighbor pairs of memories. In lattice gauge theories, all calculations are built from operations involving only quantities associated with nearest neighbor sites in the four-dimensional physical lattice. These elements will necessarily be stored either in the memory of a single node or in the memories of two adjacent nodes. In either case, with the interconnections shown in Figure 1 there is a unique processor with direct access to the pair of operands. Thus all three of the memories which are attached to a particular processor must be included in that processor's address space.

Although each memory is accessible to more than one processor, no contention will arise if all processors execute the same program in lock-step. Since the calculations of interest are spacially homogeneous, it is possible for each node in the array of processors to be executing identical code. However, only one processor need control the addressing of a particular memory since that processor (P) can anticipate the needs of a neighboring processor (Q) for data, provided a second neighbor (R) will do the same for (P). Thus all the communication required between two nodes can be carried by sixteen data lines. In addition to the lock-step operation described above in which data can be transferred between nodes, a second asynchronous mode allowing only local activity is provided. This is described below.

Often in the problem of interest one imposes "periodic" boundary conditions, effectively joining with links those planes of sites lying on opposite boundaries. These conditions can be realized by our array if the connections indicated in Figure 1 between top and bottom and between left and right are provided. Thus the connections in the array form the surface of a two-dimensional torus. It should be noted that if the

processors are physically placed in a two-dimensional plane, the inter-processor connections required to form this torus can be achieved with wires whose lengths need not grow as the number of processors increases. (In our case the maximum length required is less than eight inches.)

IMPLEMENTATION

Let us now consider the implementation of this design. The processing element at each node of the array consists of an Intel 80286/287 microprocessor with a 16K x 16 bit program memory and a microprogrammable floating point vector processor. Each node also contains 128KB of data memory divided into two independent, simultaneously addressable 32K x 16 bit banks, which can be accessed by both the '286 and the vector processor, and a switch to allow the desired pair of (local and/or remote) memories to be addressed. All of the data paths are 16 bits wide. In addition, each node is provided with an Intel Multibus port to allow connection of additional local memory, accessable by the '286. The operation of the array and the transfer of data and code to and from the host computer is directed by a simple central controller which contains a 16KB data buffer, and provides the control signals and common clock for the array. Mechanically, each node occupies a seperate board which is provided with a standard Multibus edge connector. The seperate nodes are connected to each other and to the central controller by ribbon cable. Figure 2 shows the architecture of a single node.

a) Microprocessor. The Intel '286 microprocessor supervises all the activity on the node. It can read from and write to all the memory on the node as well as the data memories on two of the adjacent nodes. In addition to controlling the vector processor, the '286 must perform all of the scalar processing required to complete the calculation. The problem at hand requires a scalar processor fast enough to execute at least one instruction for every 4 to 5 floating point operations performed by the vector processor in order to roughly balance the scalar and vector execution times. Further, the possibility of including additional memory at each node mandates a multi-megabyte address space. The Intel 80286 microprocessor with the 80287 coprocessor nicely meets this requirement.

Each '286 has its own independent program memory so that the '286 can operate concurrently with the vector processor which appears to it as a second coprocessor - a program running on the '286 will stop on a "wait" instruction until the vector processor completes its present program. Another control signal is provided for the purpose of re-synchronizing the array. When a period of asynchronuous operation, for example data-dependent branching or node-dependent subroutines, has ended, the central controller is informed and conditions this control line. When all nodes are done, the central controller sends the reset signal, and all processors are restarted synchronously.

b) Vector Processor. The floating point vector processor is based on the TRW MPY-16HJ, 100ns 16x16 bit multiplier and the TRW TDC 1022 100ns, 22 bit (16 bit significand and 6 bit

exponent) floating point adder. The vector processor is pipelined, with 125ns stages, and is microprogrammed to perform the matrix-matrix and matrix-vector multiplications described above. A complex number is stored in six consecutive bytes: the first two bytes contain the 16-bit significand of the real part, the third byte contains the 6-bit exponent of the real part, the fourth byte contains the 6-bit exponent of the imaginary part, and the fifth and sixth bytes contain the 16-bit significand of the imaginary part.

The vector processor has two independent inputs, and a new operand can be strobed into each one on every 125ns clock pulse. Each input to the exponent adder has two 16-bit latches, allowing a total of eight exponents to be stored temporarily (Figure 3). The input latches of the TRW multiplier are also separately controlled: by suitably ordering the complex multiplication no additional temporary storage is needed for the incoming significands. The outputs of the multiplier and exponent adder attach directly to the (sole) input of the floating point adder. Since matrix operations involve the accumulation of a series of products, the adder can be run in its accumulate mode and requires only one new operand on every clock pulse. Further, since the adder requires two clock pulses to complete an addition, it can accumulate both the real and imaginary part of the sum simultaneously with out any external storage. With this arrangement the vector processor can sum a string of products of complex numbers at the maximum rate allowed by the 8MHz clock = 16 million floating point additions and multiplications each second.

c) Writeable control store. The operation of the vector processor is controlled by a writeable control store (WCS) made up of static RAM arranged in 4K 48-bit words which can be accessed by the '286. The microcode provides the signals necessary to control the arithmetic unit and latches shown in Figure 4, as well as determining the addresses of the source and destination operands. The '286 initially loads two 4-word 16-bit latches which contain the base addresses for the operands. In each cycle, for each of the A and B memories, the microcode chooses one of these 16-bit base addresses and supplies an 8-bit offset to be added to it to determine the effective address of the operand. The additions are performed by two dedicated 16-bit adders. The operands are thus required to be less than 512 bytes long; there is no restriction on their alignment.

The microcode sequencer is simply a counter driven by the system clock. This counter is started when the '286 writes to a reserved 4KB range of addresses. The particular location addressed provides a preset value for the counter, and hence the starting address in the WCS for the microcode subroutine to be executed. The vector processor is stopped when a microcode bit resets the counter and releases the coprocessor busy signal from the '286.

d) Memory. In order to run the vector processor at full speed, two numbers must be obtained from memory every 125ns. To accomplish this, the memory at each node is split into two independent banks. The memories are made from 45ns static RAM,

which eliminates the need for further high speed registers to feed the vector processor. Either memory bank provides one input to the vector processor on the same node. The other bank feeds the second input of the vector processor on that or on one of the two adjacent nodes. Switching between banks is accomplished at the same time as switching between nodes, and the delay due to all switching is buried in the pipeline. There is no delay for data coming from a neighboring node. Thus the fast arithmetic is supplied by internode data transfer at a rate of 16MB per second

e) Switch. The interconnection of the '286, the vector processor, the local memory and the four neighboring nodes is accomplished by the switch diagrammed in Figure 4. It is composed of eight 8-bit transceivers, four D-type latches and four latching multiplexers. The latches and propagation delays introduced by these elements are incorporated into the pipeline which makes up the vector processor.

f) Multibus port. In order to increase the flexibility of our individual nodes, each microprocessor-vector processor combination is provided with its own, private Multibus. Thus, in addition to the memory built onto the board, the '286 has the possibility of addressing additional, conventional bulk storage attached to this Multibus. For example, a problem needing 32MB of storage which would otherwise require a full array of 256 nodes could be tackled using a much smaller array of 16 nodes if four standard 1/2 MB boards were connected to the Multibus of each node. This port also allows the direct connection of magnetic disk storage to some or all of the nodes in the array.

g) Controller. The transfer of data and code between the array and the host computer, in our case a VAX 11/780, is managed by the central controller. The input/output from the array is performed in bucket brigade fashion. Two of the nodes (through their Multibus port) can access buffer memories on the central controller, one for input and the other for output, shown as wavy lines on Figure 1. When I/O is initiated, each processor copies a page of data from a standard I/O buffer in memory at one node R (or from the controller) to the appropriate location in another node P. The processor also copies the data into the I/O buffer area in P (or into the controller), preparing for its transfer to node Q during the following cycle. Since our I/O requirements are small, we are using a single input and a single output port so that the transferred data must thread its way through the entire array at a rate no larger than 16MB per second. However, since each node is provided with a Multibus port, the system can be reconfigured to create additional ports serving smaller subsections of the array. Thus the data transfer rate is in fact limited only by the bandwidth of the interface to the host computer.

In addition to the I/O function, the central controller provides some simple global communication and commands: (i) It broadcasts a synchronous clock signal to all nodes of the array. (ii) It issues a synchronous reset signal which initiates the one page I/O transfer described above. (iii) It initiates synchronous program execution, also using the '286's reset

signal. (iv) It resynchronizes the processors after they've executed differing subroutines (v) Finally it receives an error and a finished signal from each node and transmits the appropriate composite message to the host.

PROGRAMMING

Since high level languages are available for the '286, we expect to do essentially the entire calculation using the array, from the generation of an ensemble of matrix configurations, to the evaluation of the various observables of interest. The host computer will be used to generate the code, to move data to tape periodically, and for the final fitting of the data and error anaylsis. The great majority of the calculations to be performed on the array can be carried out in the synchronous SIMD mode and are being written in a combination of FORTRAN86, PLM86 and ASM86. The programs are being written, compiled and linked on the VAX using standard Intel utilities. The resulting absolute object files are transfered to the array using a loader which resides partly on the VAX and partly in PROM on each node.

We have written an assembler which translates simple mnemonics for arithmetic operations using the vector processor into microcode. Since the arithmetic unit has a single, short pipeline and a large number of uniformly addressable registers, effective use of the full power of the machine does not require highly vectorized code, and it will be possible to write an optimizing compiler for it using FORTRAN or C - like syntax. With this compiler, the programmer will generate a set of high-speed subroutines which perform the bulk of the arithmetic in a given calculation, and which are then called from a controlling program written in a conventional language.

PERFORMANCE

A single board has a nominal speed of 16Mflops. For real programs, this limit will be reached only momentarily. A more practical measure of its performance is given by running lattice gauge theory Monte-Carlo programs. It will take the product of two SU(3) matrices in 20us, which corresponds to a speed of 10Mflops; a 10-hit Metropolis update takes less than 1.5ms at present. These numbers can be compared with those for a Cray-I and for a Cyber-205:

	Nominal	SU(3) product	10-Metropolis
Cray-I	160Mflops	100Mflops	75us
Cyber-205	200Mflops		40us

The cost of producing a single board is less than $2500; the cost for a system of 256 boards, including the necessary disks and other equipment will be less than $800K. At a nominal speed of 4Gflops, we will be able to provide more than 5000flops/$, more than 3 orders of magnitude better than commercially available computers.

CONCLUSION

This problem, in common with a large number of interesting problems in phyics, has a number of features which make it well

suited to solution by a special purpose computer. (i) The calculation is dominated by the multiplication of 3x3 complex matrices associated with the links of the lattice and the product of these matrices with three-dimensional complex vectors defined at the lattice sites. These products can be efficiently evaluated by a pipe-lined, multiplier-adder. (ii) All of the calculations are local: only matrices and vectors associated with contiguous links and sites are to be combined. (iii) The problem is homogeneous: the same products of the variables associated with the links and sites are to be carried out for all the points in the lattice. (iv) Because statistical methods must be employed the results are generally not expected to be accurate to more than a few percent and great precision is not required in the arithmetic computations. By exploiting these special features and by taking advantage of powerful, commercially available VLSI chips, we have designed a parallel array of inexpensive single board computers which will perform 4 billion floating point operations each second. The device is presently being constructed in the Physics Department of Columbia University

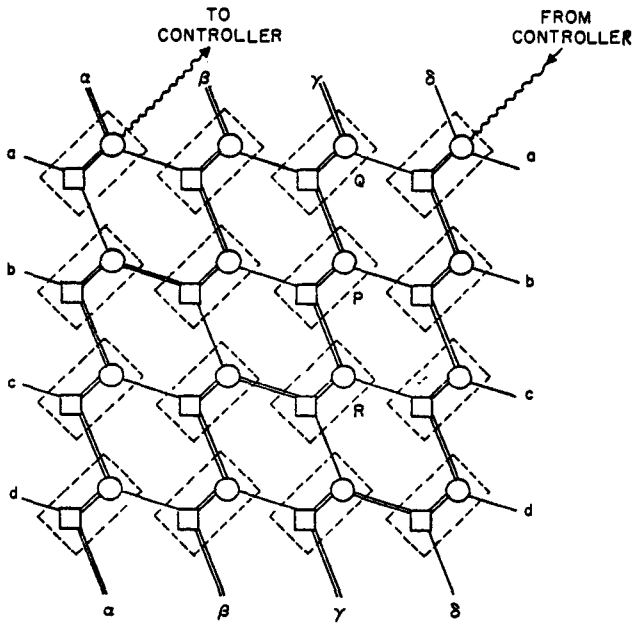


Fig. 1

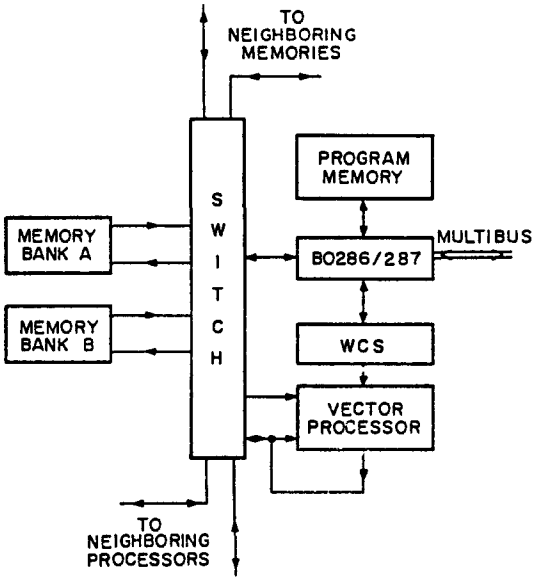

TO
NEIGHBORING
MEMORIES
PROGRAM
MEMORY
S
W
I
T
C
H
MEMORY
BANK A
MEMORY
BANK B
B0286/287
MULTIBUS
WCS
VECTOR
PROCESSOR
TO
NEIGHBORING
PROCESSORS

Fig. 2

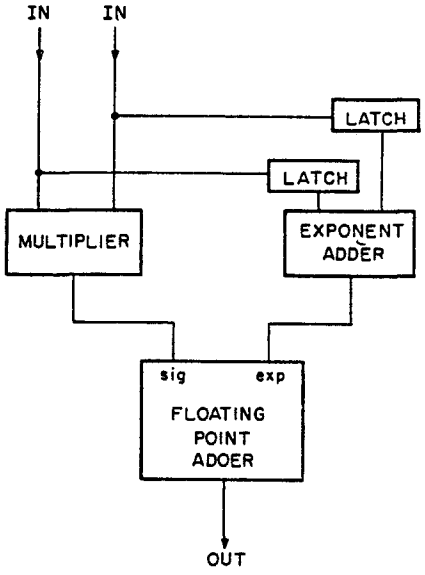

IN
IN
LATCH
LATCH
MULTIPLIER
EXPONENT
ADDER
sig
exp
FLOATING
POINT
ADOER
OUT

Fig. 3

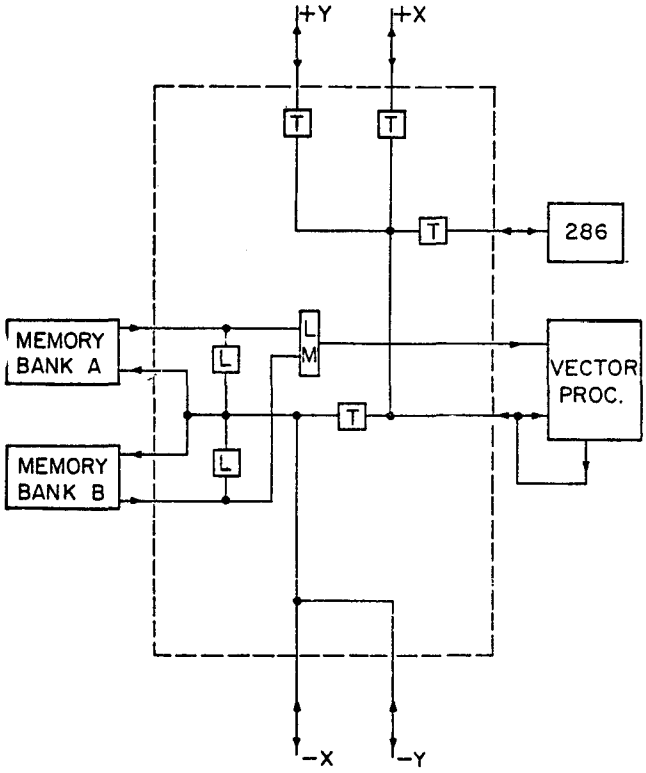


Fig. 4

QUESTIONS AND ANSWERS

Q: The funding agencies are now trying to make new large scale (Class VII) super-computers available to the research community. If such a machine became available to you, or Caltech or others, would you abandon these efforts and turn your efforts to calculating physics?

A. Brenner

A: No, it's still much more cost effective to do it this way and we get a larger amount of computing done this way.

Q: I'm tremendously impressed with the progress you people have made! Isn't it true that 3 or 4 years ago Norman Christ first appeared at Nevis Labs to ask Bill Sippach about TTL and how to use a soldering iron?

D. Kaplan

A: Yes. We cut our teeth on a small board we wired ourselves, but this project is much more rationally designed.

Q: In addition to the re-education of funding agencies, I am amused by the re-education of theorists. When a board comes back from the assembler and it doesn't work, who debugs it? - you or Norm?

M. Kreisler

A: Most of the time, it's Norm. It's part of a deal we struck when we started the project.

GIBBS - A Programming Environment and Workstation for Scientists

The GIBBS Group†

Cornell University
Ithaca, N.Y. 14850

ABSTRACT

GIBBS provides a new framework for the development, maintenance, and documentation of complex codes written in Fortran or other high level languages for scientific applications. It facilitates the creation of and implementation of highly modular code without sacrificing efficiency. Programs are organized according to the logic of the problem rather than the needs of the computer, and are therefore much more readable and changeable than programs written and documented in more conventional styles.

1. Introduction

Computing hardware is changing rapidly over time periods as short as a single year. The difficulties encountered in adapting existing computer codes and creating new ones are a major obstacle to the efficient utilization of this new hardware. Complex programs, written in Fortran, Pascal, C, *etc.*, are vitally important to science and engineering. They are however very difficult to read and modify, even with liberal use of comment cards, indenting, top-down programming, and the like. Consequently, researchers are hard pressed to find the time to write, debug, adapt, or document large-scale computer programs. This problem is particularly acute for students, who are generally required to complete significant projects in relatively short periods of time. The GIBBS Project is an attempt, conceived by Ken Wilson and involving Cornell's Computer Science, Computer Services, and Physics departments, to deal with this problem.

The heart of the problem with programs written in conventional programming languages lies in the organization of the program. This organization is

† The GIBBS Group includes D. Bergmark, A. Demers, D. Gries, P. Lepage*, D. Moitra, A. Neirynck, M. Nesheim, TK Srikanth, and K. Wilson. Cornell undergraduates participating in the project include C. Cady, and D. Freed. Additional information about GIBBS can be obtained from D. Bergmark, Cornell Computer Services, G-02 Uris Hall, Cornell University, Ithaca, N.Y. 14850.

*Presenter

generally dictated by what the computer should do next and not by what should be explained next. As a result, the central ideas embodied in the program become completely scrambled, and the program can be understood, if at all, only by constantly flipping back and forth among many pages of hard-to-read code. This point is illustrated in Fig. 1 by a simple program for studying one-dimensional diffusion. A scientist would begin describing this problem by writing down the diffusion equation, and then describing initial and boundary conditions for the equation. Only then would he get into the nitty-gritty details of discretizing the t and x derivatives, setting up data structures, and optimizing the code. These last details must be addressed in the very first line of the Fortran code. Furthermore elements of the same idea appear diffused throughout the entire code, while at the same time any given line of code many involve several different ideas. It is this complexity that makes even simple programs, let alone 60 pages of such code, difficult to understand and modify. In addition, Fortran has a rigid structure built on the ANSI character set. Thus the scientist, coding in Fortran, is denied the use of his natural language - i.e., sophisticated mathematical notation combined with English, French, or whatever.

2. The GIBBS Project

The GIBBS Project has adopted a textbook analogy for program specification. A problem is first broken down into a large number of simple modules called 'Chapters'. Like a good textbook, each chapter of a GIBBS program deals with a single idea. Chapters might describe an equation, a numerical method, an abstract data type, or perhaps an optimization targeted for a particular piece of hardware. Also in analogy with a textbook, the author is free to order the presentation of the program according to the logic of the problem. Typically, central equations appear first, followed by data type definitions and restrictions, numerical algorithms, and optimization procedures. GIBBS helps the author in establishing the interrelations between different chapters. Also, the programmer is permitted full scientific notation in specifying his problem, ultimately through the use of a graphics workstation.

The GIBBS style for writing programs is illustrated by Fig. 2. Following an introductory chapter outlining the problem, Chapter 1 deals with a key equation in numerical studies of the nuclear force. The equation is entered with a structured editor. Thus, for example, GIBBS knows that $n+\hat{\mu}$ is a subscript on Ψ, and that $U_{n\mu}$ either multiplies or operates on Ψ (it finds out which in Chapter 4). The cross-references listed at the end could be generated by GIBBS; again through the structured editor, the system understands that Chapter 5, for example, is needed to understand the significance of K and r. Chapter 4 illustrates the definition of a new data type - Gauge_Field, an array of complex numbers labeled by two indices of type color (Chapter 7), one of type nearest_neighbor

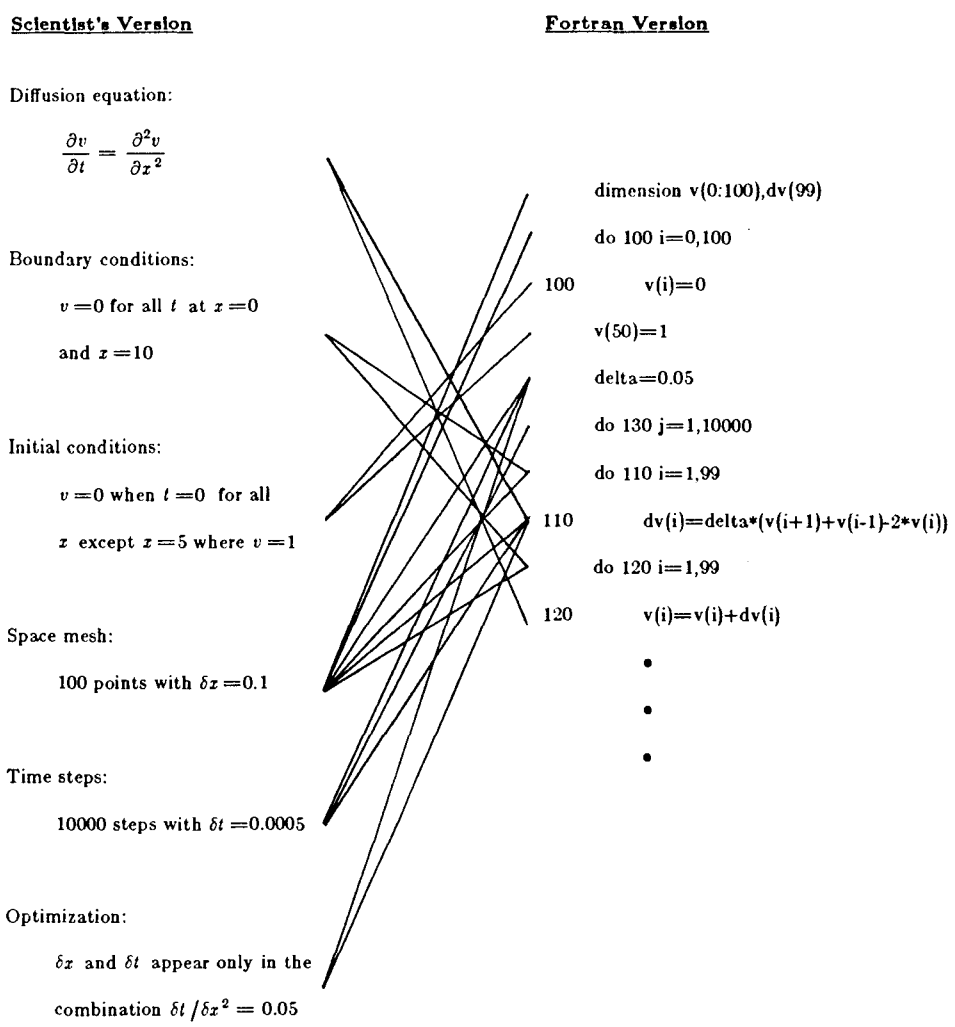


Figure 1 - Two versions of a one-dimensional diffusion problem. The lines indicate some of the correspondences between the two descriptions.

Chapter 1

The evaluation of Φ_n represents the most time consuming step in the numerical analysis of hadron structure using lattice QCD.

$$\Phi_n = K \sum_{\mu} (r + \gamma_\mu) U_{n\mu} \Psi_{n+\hat{\mu}}$$

For K and r: see Chapter 5 (Flavor)
For μ and n and $\hat{\mu}$: see Chapter 6 (Grid)
For γ_μ: see Chapter 10 (Dirac Matrices)
For $U_{n\mu}$: see Chapter 4 (Guage Field)
For Φ_n and Ψ_n: see Chapter 2 (Fermion Fields)

Chapter 4 - Gauge Field

The gauge field is the mathematical representation of the gluon field that holds quarks together. For each grid location and axis direction, the gluon field is represented by a color matrix.

type Gauge_Field = array(color, color, nearest_neighbor, grid) of type complex
variable U is type Gauge_Field

For color: see Chapter 7 (Color)
For nearest_neighbor: see Chapter 6 (Grid)
For grid: see Chapter 6 (Grid)

Chapter 6 - Grid

The theory is defined on a simple hypercubic grid of points, labeled by an integer 0...N-1 for each of D directions.

```
type grid = array(axis) of type integer
restrict 0 ≤ n_μ < N for all n of type grid and all μ of type direction
n is type grid
define 'loop on n' to be
        loop on n_D
            loop on n_{D-1}
              ...
                loop on n_1
                    { BODY OF LOOP }

....
```

Figure 2 - Sample chapters from a GIBBS program for lattice QCD.

(Chapter 6), and one of type grid (Chapter 6). The field $U_{n\mu}$ in Chapter 1 is of type Gauge_Field, and, being a variable, it multiplies Ψ rather than operating on it. The lattice grid upon which the theory is defined is specified in Chapter 6. This chapter illustrates the definition of new data types, like type grid for the lattice coordinates of a site, the introduction of restrictions on these data types, and the definition of a new loop command specific to this problem.

3. The GIBBS Agenda

Although the GIBBS Project is still in its infancy, substantial progress already has been made. Today GIBBS is a promising new technique for writing, modifying, and documenting complex programs hand-coded in Fortran or similar languages. Several GIBBS programs have been written for problems in a variety of disciplines, including theoretical high energy physics, molecular dynamics, and numerical analysis. Some of these programs have been successfully compiled into Fortran and/or C by groups of Cornell undergraduates, functioning as a 'human GIBBS compiler.' This exercise demonstrates the potential for describing complex computer programs in natural language, and provides important insights into the problems and possibilities of the GIBBS approach. There is still much to be done in specifying the nature of the GIBBS compiler. Input from researchers outside the Project is welcomed - e.g., sample GIBBS programs. A manual describing the GIBBS methodology is now available.†

A structured editor for GIBBS programs hopefully will be available in the very near future. This prototype for the GIBBS editor runs on ordinary ASCII terminals. It supports some high level mathematical notation (subscripts, $\sum$, $\int$, ...), and understands the relations between parts of a chapter and between different chapters. This makes it a useful tool for program documentation. Farther into the future, it is hoped that the hand-generated Fortran of the finished program can be incorporated into the editor, allowing cross referencing between the GIBBS documentation and the source code.

Ultimately, GIBBS should provide facilities for interactive code generation in Fortran or other target languages. A variety of systems, like Speakeasy or Macsyma, have been developed in the past to allow the direct programming of scientific problems. While elements from such systems will undoubtably be incorporated, GIBBS will deal with more complex problems - e.g., problems like finite-difference approximations to nonlinear partial differential equations, or Monte Carlo simulations of large statistical systems, where highly optimized Fortran usually is required. The tremendous flexibility of GIBBS requires that code

† Information about GIBBS and the GIBBS Manual can be obtained from D. Bergmark, Cornell Computer Services, G-02 Uris Hall, Cornell University, Ithaca, N.Y. 14850.

generation be a collaborative effort between the compiler and the user. It is hoped that GIBBS chapters can function as operators on other chapters and on the Fortran code. Then a GIBBS program will become a series of very sophisticated transformations that convert equations written in standard scientific notation into executable Fortran. This will complete the relegation of Fortran to the role of a portable assembly language, thereby greatly enhancing the prospects for large-scale computational science and engineering.

QUESTIONS AND ANSWERS

Q: How do you distinguish between a description of a problem (=) and an algorithm (←) in GIBBS?

F. Beck

A: You can actually use those symbols if you like. You can define such symbols as you go.

Q: A symbolic language for reconstruction algorithms is badly needed. It would allow a) communication between algorithm writers now almost non-existent; b) GIBBS support in this area which would be of great value; and c) identification of algorithm kernels that can be processed in specialized hardware subroutines or coprocessors.

T. Nash

A: Yes.

Q: How are you doing in connecting the GIBBS mathematical equations and target parallel or vector machines?

A. Charlesworth

A: Trying to develop a high level notation for target machines. Have tried array processor: RPS-164 eight FPS-100's, etc.

Follow-up: Should try a parallel array like the CAL-Tech Cosmic Cube.

Q: I suspect that GIBBS might become too slow to run in a reasonable time on any but a supercomputer. Can you comment? This is why in MEXLAN I aim at something much less ambitious.

T. Brody

A: Structured editors at least already exist; with VLSI chips and so on, things should be limited by the speed of the interactive user.

Q: Have you any thoughts on debugging issue of the "pre-compiler"?

M. Fischler

A: Yes. GIBBS chapters describing debugging tests could be included in a complex code description. Hopefully GIBBS will support some sort of interactive debugging of the sort discussed in earlier talks.

The CMU Multi-Micro Computational Engine

Michael J. Levine

Physics Department, Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

I. Introduction

The CMU Multi-Micro Computational Engine is the hardware result of a continuing project within the High-Energy Physics group at CMU which aims to provide a very cost effective vehicle for doing some large scale calculations in theoretical particle physics. Below, we outline the model problem which has motivated this effort, the hardware of the current engine, the programming environment in which we function, the current status of the engine and our plans for the near future. Detailed information is available in a series of internal reports.

II. The Model Problem

This engine is composed of general purpose microprocessors, but it is structured to be especially cost effective for at least one specific class of problems. We first describe a physics problem from that class and then abstract from it certain characteristics which are important from a hardware standpoint.

The anomalous magnetic moment of the electron, a_e, is the most precisely measured and calculated quantity in physics. The order of magnitude of current experimental and theoretical errors is 10^{-10}. The dominant contributions to a_e are given by QED as a power series in α the coefficients of which are obtained by evaluating certain Feynman graphs. Theoretical work is now being done on the contributions from 4-loop graphs and a few 3-loop graphs. The usual Feynman graph techniques reduce these contributions to a set of integrals, in up to 10 dimensions, of rational functions. Those integrals are being done numerically.

Because we control infrared divergences and perform the ultraviolet subtractions numerically, the integrands must be evaluated with high precision and large dynamic range. There are about 50 integrals to be done. A typical integrand numerator contains 20k terms. The denominators are of negligible complexity. We estimate that more than 10^9 integrand evaluations will be required per integration. Because the many evaluations of the integrand are independent of each other, all need not be done on the same machine.

From this, we may abstract several machine requirements. The intrinsic machine arithmetic should be at least 'double precision' and have a dynamic range of at least 10^{600}. The aggregate arithmetic speed of the machine must be at least 30 Mflops/sec in order to do the estimated $3*10^{15}$ arithmetic operations in no more than a few years. Because of the intrinsic decomposability of this type of problem, it is possible to use many processors with an interprocessor bandwidth which is quite modest. Only small amounts of memory (1-3kB) are required for data storage for any single functional evaluation. Modest amounts of memory, which might be shared between processors, are required for the code necessary to evaluate an integrand (about 60k instructions).

III. Hardware Outline

Our need for large amounts of high precision floating point arithmetic coupled with a desire to minimize our electrical engineering efforts has led us to use the Intel 8087 Numerical Data Processor (NDP) as our basic arithmetic unit. The intrinsic 8087 arithmetic is REAL*10 with 19 digit precision and 10^{10000} dynamic range. We might view the engine as a mechanism for putting the arithmetic capabilities of a large number of 8087's at the disposal of the user with as little overhead as is possible. In this picture, we have a sequence of blocks [user, host, (controllers, computational modules, numerical processors)]

interconnected by a set of communication links. The last 3 of these blocks constitute the 'engine'. This view forms the basis for the following outline of the hardware.

The user communicates with the host, a VAX 11/780 running under VMS, over a terminal line. The host communicates with the engine (specifically, with the controllers) over a low speed interconnect composed of serial lines. The engine functions as a slave processor to the VAX. Most of the control function and voluminous, nonarithmetic user code is kept within the VAX. Most of the arithmetic is done within the engine. This partitioning of function is done in a way which minimizes the required bandwidth between VAX and engine.

Within the engine, each of the two controllers communicates with and has reset control over a set of eight computational modules. The controllers and modules are single board computers of our own design and construction. The controller-module link is a parallel, master-slave, 16 bit wide bus structure with hardware handshaking. It has a hardware speed of about 1 MB/s. Arbitration is done in software by the controller which is always bus master. Through this bus, the controller can receive from any module and can transmit to any set of modules.

The single board computers used for the controllers and for the computational modules are based upon the Intel 8086 chip family operating at 5 Mhz. They are of standard design but are limited in scope. Each has a CPU section, 32 kB of ROM & static RAM memory and parallel (96 bits) & serial (2 ports) I/O. The computational modules have, in addition, a quad 8087 arithmetic section which we use as a 'micro array processor'. The 8086 and four 8087's share the local, multiplexed bus. Additional circuitry, controlled by the CPU, determines which 8087(s) will execute the next NDP instruction. It is possible to use any single NDP in the usual manner.

The problem specificity of the engine lies only in the size of the memories and in the bandwidths and connectivity of the various interprocessor data paths.

IV. Programming Outline

In normal use, there are programs running concurrently in the VAX and in each processor of the engine. These programs consist of numerous code modules written in a variety of languages. In function, the modules range from problem dependent user code to I/O and other service routines which rarely change.

The problem dependent 'user level' code on the VAX, in the computational modules and for the 'array processor' is written in Fortran. We use cross language processors on the VAX to generate all '86/'87 code. We have constructed a set of language processors which convert Fortran arithmetic statements into special code for the array processors. In production running, successive batch jobs on the VAX allocate the lines to the engine, download code to and start execution on the engine and then exchange data with the engine at intervals.

On the VAX, the user level code can initialize the engine and exchange data with the computational modules by calling a set of service routines which are, themselves, written in Fortran and which contain some calls to VMS System Services. Similarly, in the engine, the user level Fortran code calls a set of service routines (which are written in PL/M) to communicate with the VAX. Some of these service routines are in ROM and provide the basic engine boot function. Together, these service routines provide a downloading capability and a packet communications facility between the VAX and engine. This VAX-module communications facility can do conversions between VAX and 8087 floating point formats so that the user code in each environment can function in its native format.

The large arithmetic statements which specify the integrands are processed into '86 object modules which are linked and downloaded with the other '86 code. Subroutine calls in the '86 user code invoke the array-processor to do multiple, parallel integrand evaluations. Code for the array-processor is more compact than ordinary '87 code. This reduces the code memory requirements by a factor of from 2 to 4. The array-processor execution code, which is not visible to the user, is written in '86 assembly language.

Synchronization between VAX and module programs is accomplished by waiting for I/O completion. The VAX program writes input data to a module and then, or perhaps later, issues a read on that module. The module, upon receiving input data, does its computation (typically lasting 1–2 hours), sends the result to the waiting VAX program and goes to wait for more input data from the VAX. The waiting VAX program does postprocessing on the data from the module and then goes to write another set of input data to the waiting module. In a typical job, this cycle is repeated, using all modules, over a period lasting anywhere from a few hours to a week.

It is up to the user to decompose the problem into many subproblems and to delegate each, in turn, to a computational module. For multiple integrals and other highly decomposable problems, this is a trivial exercise.

V. Current Status and Future Plans

The current version of the engine has been functional for nearly 10 months. 95% of that time has been spent doing production running on 3-loop graphs. The remainder of the time has been spent doing testing and implementing modifications. Preliminary testing of modifications is done on a some solitary computational modules. Using a single 8087 per each of 16 modules provides an arithmetic capability equivalent to twice a VAX 11/780 in double precision. REAL*10 cuts the speed by 20% compared to REAL*8. We are currently upgrading each module to 4 8087's. Tests indicate that this will effectively treble the strength of the engine.

By asking each module to do a subintegration rather than a single functional evaluation between communications with the VAX, the ratio of I/O time to arithmetic time can be made extremely small. In usual production running, the VAX spends less than 1% of its resources looking after the engine.

A few percent of all running time is spent redoing old calculations as a test of integrity and reliability. We have detected no aberrant behavior. Doing numerical integrations with successively finer integration meshes provides a built in check on errors and protection against them. A low error rate would simply slow the apparent rate of convergence; a high error rate would destroy convergence.

Modules which fail to respond to the VAX within a reasonable time are declared dead for the remainder of the job. Their work is given to other modules. Such 'dropouts' happen about once a month. They are largely attributable to severe electrical interference or to the low grade sockets which we used in the first few boards. The overall mean time to failure of the engine is considerably longer than that of the host.

We are currently building the next, more compact, iteration of the computational module. It will have more memory (128kB of dynamic RAM with parity) and no serial I/O. We hope to make up to 512 of those modules during the next academic year. This would give a useable full scale strength of approximately 40 Mflops/sec.

VI. Acknowledgements

I wish to thank and acknowledge the help of T. Kikuchi and S. Friend as well as the advice of many experimental colleagues at CMU. This work was supported in part by the U.S. Dept. of Energy under contract DE-AC02-76ER0306 and in part by material contributions from Intel Corporation. We would like to thank the organizers of this Symposium for their efforts.

QUESTIONS AND ANSWERS

Q: Do you do program testing and debugging with a simulator on the VAX?

T. Brody

A: No. We test FORTRAN source code on the VAX and then it has always worked.

Q: How do you detect and handle arithmetic errors in 8087's?

J. Amann

A: Integral computations tend to be self-checking due to use of various meshes -- if error is bad it won't converge. Also we periodically run test cases. We find micro processor engine is as reliable as VAX host.

Algorithms for Concurrent Processors

Steve W. Otto

Physics Department
California Institute of Technology
Pasadena, Calif., 91125

ABSTRACT

I describe the general techniques in the use of concurrent processors for scientific problems. It is described how one usually obtains linear speedup with a computational power that is not only proportional to the number of machines making up the processor but has a proportionality constant that is near 1. Examples from statistical mechanics, astrophysics, and high energy physics are discussed. Before concluding, I describe the current state and direction of the Caltech-JPL concurrent processor project.

Technological Motivation

The VLSI technology revolution is expected to lead to somewhat faster but, mostly, much less expensive computers contained on a few chips [1]. The expected increase in cost-effectiveness of these machines is quite impressive. As an example, the 32 bit multiplier-adder chip set of Weitek provides approximately 5 million floating point operations per second ("Mflops") of performance (when used in a pipelined calculation) for a cost of about $1000. A similar system for 64 bit arithmetic is probably not too far away. It is possible to exploit this technology and build very high performance computers by combining very many of these cost effective units into a single concurrent processor [2]. I will term the basic (VLSI) building block a "node" in the rest of the text; a node is itself a small but complete computer of modest power. Concurrent processing seems a more practical route to high performance than the design of a single

very fast machine. In fact, it is expected (or perhaps I should say, some of us expect!) that one can build machines consisting of about 10,000 nodes with each node being an individual computer capable of 10 Mflops. Such a design seems practical five to ten years from now and offers the promise of machines that are one thousand times as powerful as current supercomputers. Such "top of the line" machines would be accompanied by smaller collections (of, say, about 100 individual nodes) which would have a total power of some thousand megaflops at a cost of perhaps $100,000 (for the basic cpu and memory - I am ignoring such essential peripherals as disks). This increased power will revolutionize the computational approach to all scientific and engineering fields. For instance, one will be able to solve such difficult and important problems as weather prediction and the dynamics of quantum field theories.

The above, rather attractive, scenario is the driving force behind research in parallel computing. The main stumbling block to the use of concurrent processors is the difficulty of formulating algorithms and programs for them. Indeed this leads some to doubt the utility of these machines. The goal of this talk is to discuss the general techniques for using concurrent processors and illustrate them with some simple examples. It is our belief that these machines are fairly easy to use and are not specialized devices but rather can address the vast majority of computationally intensive problems. I will mainly confine myself to science and engineering fields (as opposed to, say, artificial intelligence) as in these cases the algorithms are well understood and so it is possible to quantify the effectiveness of concurrent processors. However, we believe that similar considerations apply to other applications [3].

This talk is divided into two parts. The first part will be a somewhat general discussion of the use of concurrent processors for the solution of scientific problems. Secondly, I will describe what we are doing in our project, both in terms of

hardware and software.

General Features of the Problems

Before moving on to specific examples, let us identify some of the general properties of computationally demanding problems. In Table 1, several examples are listed and, also shown, are some of their features which we have found important in their implementation on a concurrent processor. In each case, one must decompose the total problem into many parts -- one for each node. Typically, each problem is not demanding because of complexity of the algorithm in a conceptual sense. Rather, there is a relatively simple procedure (e.g., computing ∇^2), which must be applied to a basic "unit" (e.g., the field) in a "world" that consists of a huge number of such units. In finite difference problems, the unit is a grid point in a three dimensional world. In a study of the evolution of the universe, the unit is a galaxy and the world is the universe itself.

The first step in the decomposition of such a problem onto a concurrent processor is to divide the world into subdomains in such a way that each node is responsible for a single region. If we have N_n nodes and a total of D units (for example, grid-points) we find $n=D/N_n$ adjacent units in each node. This type of decomposition is only possible if $D \geq N_n$ and we will see later that in fact $D \gg N_n$ is desirable. This constraint is easy to satisfy; today, calculations with $D \gtrsim 10^6$ are commonplace and in every case the number of degrees of freedom in state of the art calculations is increasing with time!

There are, of course, exceptions where computationally intensive problems cannot be so decomposed. As an example, consider the N body gravitational problem for N=10 (the solar system), where we wish to integrate the 10 equations of motion for a very long time, T. This large parameter, T, **cannot** be as easily decomposed and we can use, at most, 10 nodes for the problem.*

* On the other hand, for the actual example discussed, one usually wants to examine the results of the integration for a variety of initial conditions. The problem can then be decomposed on the product space -- particles and initial conditions -- and so make effective use of

TABLE 1

APPLICATIONS AND FEATURES RELEVANT FOR CONCURRENT PROCESSING

Class of Problems:	Examples:	Unit and World:	Natural Load Balance?	Communication Range:	Topology:
Finite Diff. Finite Element P.D.E.	Geophysics Aerodynamics	grid point, space (x,y,z)	Yes	Short	3D Mesh
Statistical	Lattice Gauge	spacetime (x,y,z,t)	Yes	Short	4D Mesh
	Melting	Configuration space (x,y,z)	No	Short	3D Mesh
	Coulomb Gas	Particle Number	Yes	Long	Ring
Time Evolution 1/r Potential	N-body Gravity	Particle Number	Yes	Long	Ring
Time Evolution General Dynamics	Particulate Motion (sand, avalanches)	space (x,y,z)	No	Short	3D Mesh
Fast Fourier Transform	Evolution of universe,	"Bit space"	Yes	Long	Hyper Cube
	Fluid dynamics	space (x,y,z)			
Network Simulation	Circuit Simulation	Component, circuit	No	Long (sparse)	logarithmic graph (e.g., hypercube)
	Neural network	neuron, brain	No	Long	
Isolated	Ray tracing (graphics) Data Analysis Initial condition study	Event space	Yes	None Needed	
Image Processing (or see FFT)	Analysis of Satellite data	Pixel space	Yes	Long	Hyper cube
Artificial Intelligence	Chess	Inference, Decision tree	Yes	Short	tree
Event driven simulation	Industrial/ Economic/	cars on a freeway;	No	Mainly short	logarithmic graph (e.g., hypercube)
	Military ("war games")	tanks on a battlefield;			

General Features of our Approach

There are many possible designs of concurrent processors differing primarily in the number and nature of the nodes and their interconnection topology. We will consider as target hardware for our discussion what are termed ensemble or homogeneous machines by C. Seitz [4]. These are collections of identical nodes -- each a complete computer with its own arithmetic unit and memory. Although this is not necessary for every application, we will assume that each computer can execute its own instruction stream, i.e., that the target hardware is MIMD (Multiple Instruction, Multiple Data). The nodes may even have a more fine-grained level of concurrency within them, such as pipelining. We will allow the interconnection topology to be general and examine each problem to find the "natural" connectivity. Of particular importance is the so-called hypercube (more precisely, Boolean hypercube) topology -- $N_n=2^\gamma$ computers with the connectivity of a cube in γ dimensions. We will **not** assume that there is any shared memory accessible by all nodes; the simpler distributed memory architecture seems sufficient for our applications.

It is convenient to characterize the effectiveness of a concurrent processor by the *speedup*, S, defined so that the collection of N_n nodes runs, for the same problem, S times faster than a single node. Furthermore, define the *efficiency* ε so that $S=\varepsilon N_n$. We wish to examine the effects that reduce the performance of a concurrent processor and lower the efficiency from the nominally perfect value of unity. One is usually quite satisfied to find algorithms with **linear speedup** - those with an efficiency ε that is independent of N_n and of reasonable size -- say $\varepsilon \gtrsim .50$.

---

a large concurrent processor.

There are at least two issues that need to be addressed in discussing the efficiency. Firstly, the nodes must spend some time communicating with their neighbors. This is minimized if the inter-node communication, demanded by the algorithm, always proceeds by a "hard-wired" path. Note that communication in ensemble machines can be viewed as a mail system where messages may be sent between arbitrary nodes through intermediate nodes. Obviously, the "wasted" communication time is minimized if the amount of such message forwarding is small. In general, the "world" which is decomposed in a particular problem has a certain topology which dictates the appropriate hardware connectivity. The hypercube node connection is attractive because it includes the ring and (many different) mesh topologies as subsets as well as being that needed for the fast Fourier transform. Furthermore, the distance between arbitrary nodes grows only logarithmically with the total number of nodes. This means that the forwarding overhead is modest for problems (such as circuit simulation and "war-games") which have an irregular structure.

The second issue affecting performance is that of "load balancing"; one needs to insure that each node has essentially identical computational loads. The efficiency is typically reduced by a factor which is approximately the ratio of the mean computing load per node to the maximum load per node. For simple partial differential equation based problems, identical loads are achieved by assigning equal numbers of grid points to each node. For this regular problem, this corresponds to equal volumes of the "world" in each node. For homogeneous problems, it is generally easy to achieve balanced loads, but in some inhomogeneous cases, care is necessary. Consider a gravitational evolution, where we assign equal number of stars (or other celestial bodies) to each node. If we are working in a region where, say, binary stars are formed, then velocities will tend to be high and we may need a reduced time step for this case. So, nodes

NOTE

Pages 107-11 missing from original.

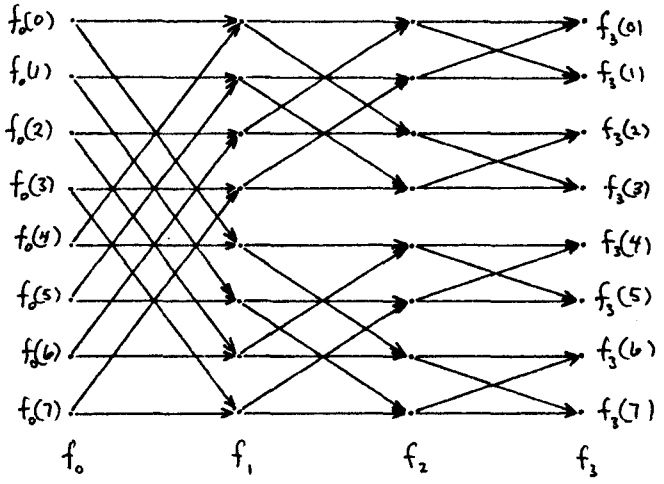

$f_0(0)$
$f_0(1)$
$f_0(2)$
$f_0(3)$
$f_0(4)$
$f_0(5)$
$f_0(6)$
$f_0(7)$
$f_3(0)$
$f_3(1)$
$f_3(2)$
$f_3(3)$
$f_3(4)$
$f_3(5)$
$f_3(6)$
$f_3(7)$
f_0
f_1
f_2
f_3

Fig. 3a

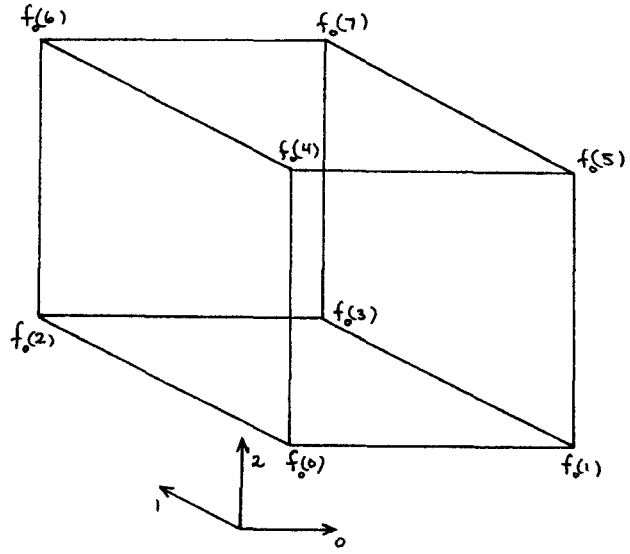

$f_0(6)$
$f_0(7)$
$f_0(4)$
$f_0(5)$
$f_0(3)$
$f_0(2)$
$f_0(0)$
$f_0(1)$
2
1
0

Fig. 3b

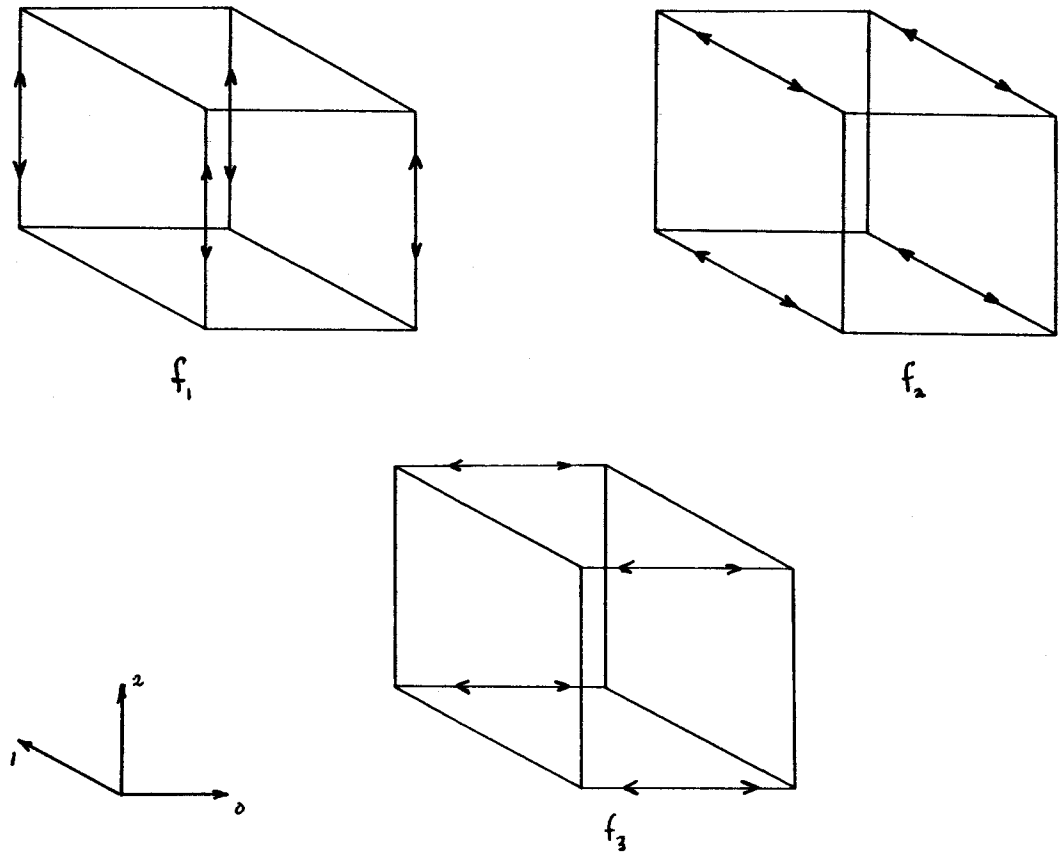


Fig. 3c

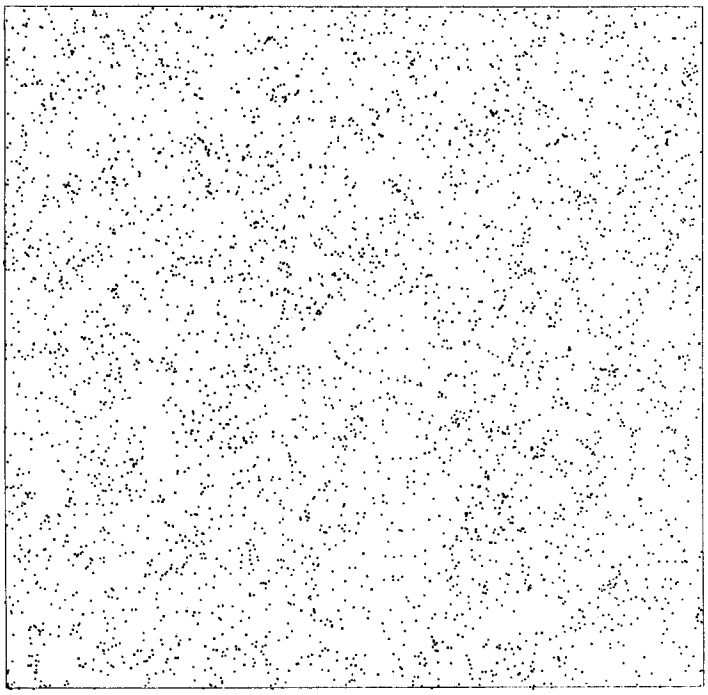

Hub0: .05
Omega0: 1
Expansion parameter: 2.079873
Number of particles: 4096
Filename:
partout64/cld16.12.40

Fig. 4a

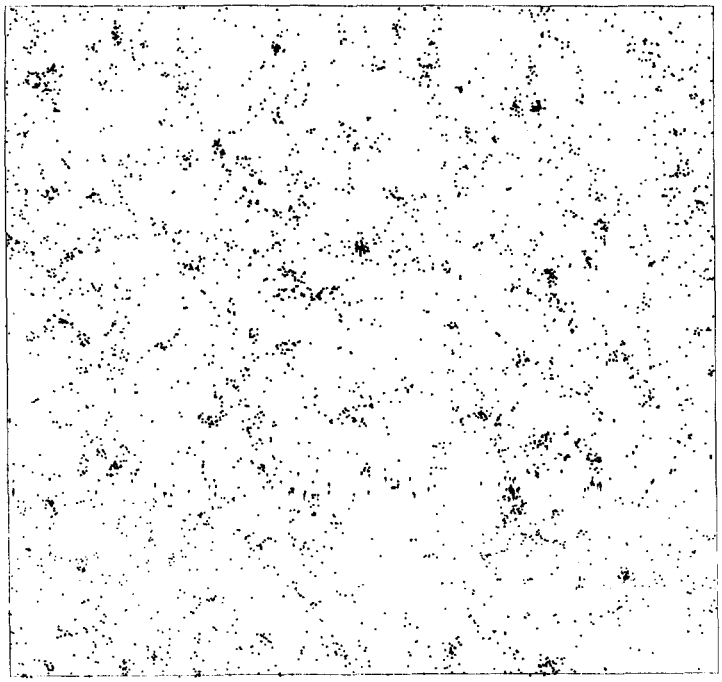

Hub0: .05
Omega0: 1
Expansion parameter: 4.326456
Number of particles: 4096
Filename:
partout64/cld16.12.160

Fig. 4b

is referred to [10] and references therein.

The Caltech-JPL Concurrent Processor Project

I would now like to describe what we have done and where we are headed in our project at Caltech.

The Prototype Computer

We have built, as a collaboration between Physics (headed by G. Fox) and Computer Science (headed by C. Seitz) a 64 node hypercube computer. Each node has 6 I/O ports (channels) connecting it with its 6 neighbors, is based on the Intel 8086-8087, has 128K bytes of RAM storage, and 8K bytes ROM for boots, memory test, and downloading of code throughout the hypercube. At 5Mhz, the node is $\approx \frac{1}{8}$ of a VAX 11/780 (C code to C code comparisons), so the cumulative power of the computer is ≈ 8 VAX 11/780s. In terms of Mflops, with the 8087s running flat out, the machine achieves 50Kflops × 64 = 3.2Mflops (for 32 bit). In actual usage (with the C cross-compiler) we typically get about 2 Mflops. The total memory of this machine is 8 Mbytes. In terms of reliability, the hardware supports single bit error detection in memory (we see one every ~2 weeks) and in software, checksums capable of detecting single bit errors in communications are kept (these are very rare and we have seen only a few).

The cost of a node, including parts, printed circuit board, and some of the labor, is approximately $1000, making the entire machine cost about $80,000 (I have added in the cost of some peripheral hardware). At the time we started, the best off-the-shelf floating point performance to be had was the 8086-8087. This is now changing and I will mention later our future plans.

Our collaboration has more recently grown to include the Jet Propulsion Lab and we have been involved with them in producing 300 more nodes. These

are basically a tuned-up version of our original node (almost twice as fast, twice the memory per node) and will be configured as a 2^7, a 2^6, and several 2^5 machines. The nodes are currently in production and the first 32 node cabinet will be "delivered" at the end of May. These machines will provide the "capacity" to serve the many potential users the project is attracting. In regards to this, our basic philosophy has been to provide the incentive (i.e., some substantial amount of computational power) to scientists and engineers to learn how to use these parallel machines. To a large extent, this seems to be succeeding; many groups at Caltech and JPL are learning about using a hypercubic, MIMD machine for their particular applications. Contrary to some beliefs, we have not found the programming of this MIMD machine extremely difficult. This is probably best indicated by the large number of applications which are being developed to run on the machines - these are listed in Table 2.

Future Machines

Our current machine should be regarded as an experimental proving ground toward the construction of much larger and faster multiprocessor systems. In collaboration with JPL, we are currently designing our next generation system, based upon one of the powerful 32 bit microprocessors coming out and the high performance floating point units now available. By summer of 1985 we expect to have a node capable of up to 4 Mflops in a pipelined calculation (for 32 bit; 2 Mflops for 64 bit), containing 1 Mbyte of memory, and costing approximately $6000. Furthermore, this node will contain 10 communication channels and so will be configurable in up to a 2^{10} cube. Though this huge, monolithic machine is attractive, an even better idea is, perhaps, to build a 2^6 version. One of these would be capable of up to 250 Mflops, would have 64 Mbytes of memory, and cost about $400K. A research group could afford to buy such a "desk top

TABLE 2: CODES WRITTEN FOR THE HYPERCUBE

Who	Scientific Field	Application	Algorithm	Decomposition
S. Otto, P. Stolorz	Lattice Gauge Theory	SU(3) quark potential	Monte Carlo	3D mesh
R. Gupta, S. Otto A. Patel	"	Real Space Renormalization for SU(2)	"	4 different 4D meshes
S. Otto	"	Finite Temp SU(3) with quarks	Double Monte Carlo (Pseudo Fermions)	3D mesh
M. Johnson	2D, 3D Stat Mech.	phases of gases, liquids	Irregular Monte Carlo	2D,3D mesh
F. Fucito, S. Solomon	"	vortices of planar xy model	Monte Carlo	2D mesh
J. Salmon	Cosmology	Large scale structure of universe	N Body - FFT	3D mesh - hypercube
W. Athas R. Faucette C. Seitz (CS)	Computer Science	general operating system	-	-
S. Mattison C. Seitz	"	Circuit simulation	-	-
G. Fox A. Gee	any	matrix eigen-value package	subspace iteration	2D
P. Hipes, A. Kupperman (Chemistry)	Chemical Reactions	Quantum Mech of collisions	Matrix inversion	2D mesh
P. Haff B. Werner	Particulate motion	avalanches, sand dunes	time evolution	3D
D. Meier (JPL)	Astrophysics	Black Hole jet dynamics	finite element PDE	3D, finite elements
S. Lewicki, S. Otto N. Warner	"	galactic dynamics	N Body - FFT	1D ring
R. Clayton,	Geophysics	exploration	finite diff,	2D, 3D

(Geophysics)		geophysics		
B. Hager (Geophysics)		geodynamics	finite elements Conjugate gradient inversion	2D, 3D
D. Jefferson (JPL)	Simulation	circuits, networks	Time Warp	random
E. Felton S. Karlin S. Otto	Optimization	Traveling Salesman Problem	Simulated Annealing	random

Cray"!

Conclusions

In this talk we have tried to show that a large class of computationally demanding problems can be done efficiently on a concurrent processor. As for the interconnection topology, it seems that the hypercube is fairly general - it includes the ring and meshes, matches the FFT and is a logarithmic graph, making it suitable for the inherently long distance algorithms such as circuit simulation.

In the past, the subject of parallel algorithms has been a somewhat esoteric pursuit, known to a few computer scientists. It has been mainly a theoretical subject, for the simple reason that few appropriate machines existed. VLSI technology is rapidly changing this situation, for it will soon be possible to cheaply build machines of very high processing capability. With this motivation, we believe that scientists will learn to use the parallel algorithms already known and no doubt invent better ones. This will not only delineate the basic principles of decomposition but help the development of tools (languages and compilers) to make concurrent processors (almost!) as easy to use as sequential machines.

References

[1] J. Matisoo, C. Seitz, "Engineering Limits on Computer Performance", Physics Today, May, 1984

[2] L.A. Conway, C.A. Mead, "Introduction to VLSI Systems", Addison-Wesley, 1980, chapter 8; and C. Seitz, "Ensemble Architectures for VLSI: a Survey and Taxonomy", Proceedings of the MIT Conference on Advanced Research in VLSI, Artech Books, 1982;

[3] A general reference for parallel algorithms is: H.T. Kung, "The Structure of Parallel Algorithms", Advances in Computers, vol 19, p. 65, Academic Press, 1980; see also, G.C. Fox, S.W. Otto, "Algorithms for Concurrent Processors", Physics Today, May, 1984.

[4] C. Seitz, "Experiments with VLSI Ensemble Machines", Journal of VLSI and Computer Systems, vol 1, no. 3 (1984).

[5] G.C. Fox, Caltech preprints, CALT-68-939 and CALT-68-986 (1983)

[6] E. Brooks III, et. al., Nucl. Phys. B220 [FS8], p. 383 (1983)

E. Brooks III, et. al.,"Nearest Neighbor Concurrent Processor", Caltech preprint, CALT-68-867 (1981)

[7] J. D. Stack, Phys. Rev. D27, p. 412 (1983)

N. Isgur, G. Karl, Physics Today, November 1983

[8] E. Brooks III, et. al., Caltech preprint, CALT-68-1112 (1984)

S. Otto, J. Stack, Caltech preprint, CALT-68-1113 (1984)

[9] "The Fast Fourier Transform", E. O. Bingham, (Prentice-Hall,1974)

[10] "Parallel Computers", R. W. Hockney, C. R. Jesshope, (Adam Hilger, 1981)

QUESTIONS AND ANSWERS

Q: 1) How do you measure efficiency on the system?

2) How have you handled the problem of random number generation in your lattice gauge calculation, particularly with respect to passing the random number generation procedure and the total random number space required?

C. Maples

A: 1) In the case of FFT as much of the problem was run on one node and compared with the 64 processor machine. Some problems could not be completely run on a single node but the scaling was straightforward.

2) We use different seeds for each processor. Because of the complex nature of the calculation and the utilization of random numbers, repeating or overlapping random number cycles are acceptable since the values will be utilized differently.

Q: At what point do you encounter interconnection problems when scaling up 2^2 hypercube of processors?

D. Kaplan

A: One-dimensional arrangement of processors works up to 4000 processors. Two-dimensional up to 32,000 processors.

Q: What processor will you use in your generation 2 machine to get 4 mega flops?

I. Gaines

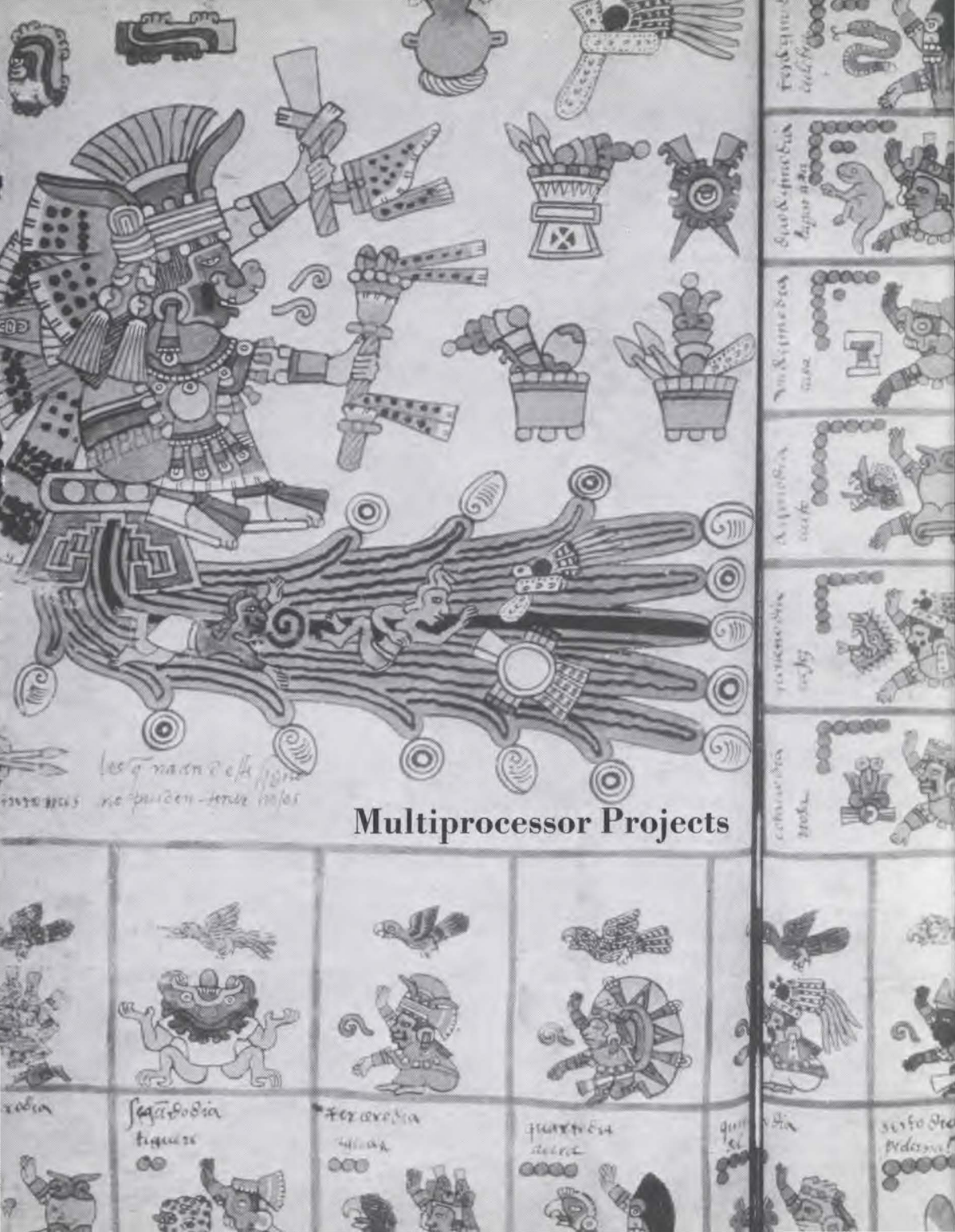
A: National 32032 Floating Point: Weitek

Q: Do you have reliability problems?

M.J. Levine

A: 1) Parity errors occur about 1/2 weeks.

2) Node failure ~1/3 months. We flush the process and restart (after fixing any hard errors).



Multiprocessor Projects

[Chalchiuhtlicue, "Jade Skirt," Goddess of Water. Codex Borbonicus, p. 5. Screenfold manuscript, panel 39 × 39.5 cm. Mexico City. Preconquest or Early Colonial.]

Problems in Parallel Processing

Daniel D. Gajski
Jih-Kwon Peir

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

1. Introduction

In this paper we will consider some essential issues in multiprocessor architectures. Previous work in this field considered taxonomies based on instruction and data streams [Flyn72], instruction and execution streams and types [Kuck78], or different models of computation and their implementations [Trel82]. We will not try to produce another taxonomy based on architectural features. Instead, we will discuss requirements needed to solve problems on multiprocessors without describing any particular architecture in detail. Nevertheless, we will discuss how each of the requirements is implemented in certain presently available or proposed machines.

2. Architecture Evolution

In this section we will consider three different types of architectures in their evolutionary order.

Von Neumann architecture is shown in Figure 1. It consists of a memory, a processor, and a bus between them. Since data and instructions are stored in the memory, and the processor controls and performs the computation, that is, it generates addresses for data and instructions, fetches them and computes on data, the bus is the most frequently used part of the system. To avoid this potential bottleneck, the designers of von Neumann architecture add a small fast local storage, general register, local memory, or cache to the processor. It is used to save local data and instructions under the assumption that they will be accessed more frequently by the processor. In what follows we will always assume that a processor may contain such a local storage.

Vector machine (Figure 2) introduced vector instructions in case when the same operation was performed on many sets of operands. This way only one instruction fetch is executed for many data. If a program contains only vector instructions, then the ratio between fetched instructions and data will be very small and the bus load will drop significantly for large vectors. Furthermore, vector machines usually increase performance by pipelining operations in the processor. Since memory read and write cannot be pipelined, interleaved memory organization must be used. Such an organization allows a vector of data to be read simultaneously from the same location in all memory banks and then sent to the processor over a transmission pipe. Although vector machines such as Cyber-205, Cray-1, and Fujitsu VP-200 are highest performance machines today, they are burdened with several problems. First, structured data that are not vectors of stride-1 are difficult to handle because of memory conflicts. Secondly, programs do not consist only of vector instructions. Thirdly, the market demand for increased performance cannot be satisfied effectively anymore by increased level of pipelining (more stages in the pipeline) or by faster circuit technology (reduced clock speed).

Multiprocessor architecture uses several identical processors to compute on one problem. This approach introduces three new requirements that have not been encountered before. First, each problem must be partitioned into tasks; secondly, each task must be scheduled for execution on one or more processors; and thirdly, synchronization of control and data flow must be performed during execution.

With respect to passing data between two tasks, two types of multiprocessors can be indicated. In the **shared-memory model** (Figure 3a), data are in preallocated locations in the shared memory where it can be accessed by each processor and operated upon without interruptions from other processors. In the **message passing model**, (Figure 3b) there is no global shared memory in the system. Each processor has an associated local memory and that data is passed from the producing processor to the consuming processor through the connection network. Both models require a general purpose connection network. The advantage of the message-passing model is that data is passed only once through the connection network while two passes (write and read) are needed for the shared-memory model unless the data is in the local storage. Yet another advantage of message-passage model is that for data-driven computation, data is passed through the network at generation time and not when it is needed. Thus, longer delays through the network can be tolerated in case when data is not used immediately after its generation. For demand-driven computation, data is fetched when needed and long delays through the network must be tolerated at every fetch.

Since connection networks are expensive only small networks are built as crossbars [Farm84]. Usually, a limited access network such as shuffle-exchange in Cedar [GLKS84], Banyan in TRAC [Brow84] or Boolean-cube in Cosmic Cube machine [Seit84], is built. For WSI technology it is reasonable to restrict connection network to the nearest neighber connections [AbGa84]. As long as each processor on wafer operates asynchronously from other processors and can access all other processors through the network, the WSI implementation is a general-purpose multiprocessor shown in Figure 3. If we limit each processor to execution of the same instruction at the same time and all processors to operate in a lock-step manner, the

systolic-array model [Kung82] is obtained. Each processor in systolic array performs simple arithmetic operations such as addition and multiplication and has few registers for storing data. A larger main memory outside the systolic array supplies data which moves across the systolic array as it gets operated upon. Data is skewed in space and time according to the shape and type of systolic array. This limits each systolic array essentially to one algorithm. When different algorithms are used, systolic arrays of different types must be combined. Regularity is lost and extra delay processors must be added for adjusting skewed data generation and consumption of two arrays (Figure 12 in [Kung82]). Furthermore, each systolic array of size n can only solve problems of size n or smaller. Larger problems must be partitioned, not a trivial task, into slice of size n and executed serially. Because of different topologies required by different algorithms, systolic arrays are not general purpose engines, but may serve as hardware accelerators for frequently used algorithms.

To overcome the problems of fixed connectivity, Snyder [Snyd82] introduced a programmable systolic array in which each processor is surrendered by switches which allow embedding of different topologies into the same physical array. The switches are set by an operating system before each phase of the algorithm. Because of synchronous lock-step operation and lack of memory in each processor it is very difficult to program such an array, particularly when two algorithms must share the same array or when slightly different computation must be performed on the boundaries.

Since systolic model and its derivitives are used for special-purpose computation, we will not consider them further in this paper.

3. Parallel Model of Computation

The model of computation is represented by a *control graph* in which *nodes* represent one or more transformations or movements of data and *arcs* represent order in which nodes are executed. Arcs rise from data dependencies when data produced by one node is used by its successor, or from control dependencies when an order of execution is specified by the user through a language with limited or no capacity to express parallelism.

In the **serial model of computation**, usually used with single processor, nodes are serially ordered (Figure 4), each node representing one machine instruction. In this case a simple program counter is sufficient in keeping track of the next executable instruction.

A **parallel model of computation**, characterized by a general directed graph, must be used for a multiprocessor. Three basic problems can be indentified in the parallel model:

(1) *Partitioning problem*: Partition a program into tasks, where each task is represented by a node in a graph. Such partition must be optimal with respect to performance or some other measure of 'quality'.

(2) *Scheduling problem*: Assign each node to one or more processors for execution.

(3) *Synchronization problem*: Determine all executable nodes in the graph and mark them for execution.

There are two sequencing methods in the parallel model of computation. In a **data-driven** execution the graph is executed in the direction pointed by arcs, that is, a node is executable when all the data needed for its execution are available. This is implemented by sending tokens down arcs. When tokens are available on all input arcs to a node, the node is executable. In the *dataflow model* of Treleven [Trel82], tokens carry data with them and no preallocated storage is needed for data. In Treleven's *control flow model*, tokens carry pointers to storage location where data can be found. (The anology is passing parameters by value and reference.) The former model is suitable for expression evaluation as well as those problems involving single data items. The later model is necessary for structured data, such as matrices, which may be only partially transformed by each node during the course of computation. It would be really inefficient to carry the entire matrix around if we want to change only one element or perhaps one row of it. Thus any general-purpose architecture must include tokens with reference to data structures, since tokens that carry values are not efficient. An example is I-structures in tagged token architecture of Arvind [ArTh80].

The **demand driven** execution processes the control graph in the opposite direction from data-driven. First, the result is demanded, which in turn requires evaluation of its arguments and so on. This process continues until constants are encountered in which case a value is returned to the demanding node. While data-driven execution is redundant, the demand-driven is not; that is, only those nodes whose values are needed in the final result are computed. In data-driven execution, for example, both *then* and *else* parts of a conditional statement are computed whenever the data are available, with one part of them being selected later. This allows parallel execution of the *condition*, the *then* part, and the *else* part. This redundant computation may increase the execution time. For example, if computation of the *condition* part takes 10 time units while 20 and 100 time units are needed for the *then* and *else* parts, then the entire statement will take 100 time units in the best possible case. On the other hand, if the *condition* is evaluated first followed by *then* or *else* parts, then execution time could be either 30 or 110 time units depending on which part was selected. We see that considerable gain in performance can be obtained if the *then* part is chosen in the above statement. So, in many dataflow model such as Arvind's U-interpreter, the conditional statement is executed in the demand-driven mode while the rest of the graph is data-driven. However, a data-driven model is still less efficient, since arguments for both *then* and *else* parts are evaluated in parallel, although only one set will be used later.

4. Partitioning

The *partitioning* of a problem into many tasks and their execution on a multiprocessor has a dual purpose: first, increasing the performance or execution speed of a single program;

and secondly, increasing the efficiency or the throughput of the machine in a multiprogramming environment. The partitioning can be performed by the user during algorithm design. In this case the user needs a language such as OCCAM [Wils83], that will adequately define separate tasks and communication of data between them. Furthermore, the partitioning can be performed by a compiler such as Paraphase [PaKL80], and BULLDOG [FiOD84], or by the machine at run time such as IBM 360/91, and CDC 6600.

So far we have been equating a node in the graph with a task, but we never defined what a task is. Usually it is assumed that each node in the graph represents one machine instruction. However, it can be as small as an arithmetic operation such as addition or multiplication. This fine granularity of parallelism is exploited by dataflow machines. The crude granularity is obtained when we combine more than one arithmetic operation into each node. In this case, each node may represent a vector instruction or an iteration of a loop. On an even higher level, we may consider each node to be a subroutine or the whole program. A task (node) is a unit of scheduling, which can be executed on one or more processors.

Nevertheless, there is a general relationship between granularity and performance. Figure 6a shows a fine granularity graph with 7 nodes and 9 arcs. A time penalty for scheduling of each node and for synchronization of each arc must be added to the execution time of the program represented by the graph. However, if we merge nodes 2 and 3, 4 and 5, and 6 and 7, we will obtain the graph shown in Figure 6b for which a much smaller time penality must be paid. On the other hand, all the parallelism available in the original graph will not be exploited. When we merge nodes 6 and 7 into new node Z, for example, we force sequential execution on nodes 5 and 6 since node Y is executed after node Z. Thus, as we merge or fuse more and more nodes together, we pay less in synchronization and scheduling overhead but more and more parallelism may be wasted.

The amount of parallelism wasted for random structures such as those orignated from expression evaluation is much higher than for regular structures such as those orignated from linear algebra. For example, addition of two vectors of size 100 can be scheduled on 10 processors in such a way that each processor generates the sum of every 10th element of the resultant vector. The synchronization is performed only once at the end after each processor executes all 10 additions. Scheduling and synchronization overhead is much higher if we consider each addition separately. This relation is shown qualitatively in Figure 7.

Every multiprocessor architecture attempts to exploit as much parallelism as possible at the lowest possible overhead. Proponents of dataflow architecture [Denn80], [Arla83], [WaGu82] believe that each problem can be transformed into expression evaluation with negligible scheduling and synchronization overhead and thus have chosen fine granularity as the main principle of their machines. On the other hand, proponents of crude granularity dataflow [GLKS84], [GaRo84], [HwSu83] believe that the solution to most of the important problems in science and engineering can be solved with structured data and operations on them and have chosen crude granularity as an underlying principle of their architectures. This way they hope

to overcome the overhead problem associated with fine granularity.

An obvious solution to the overhead problem is to hide it or, in other words, overlap it with execution, called instruction pipelining. If such a machine has approximately 10 stages in the execution pipeline, then the number of nodes executable in parallel must be 10 times the number of usable processors in the machine. If we have 5 processors then 50 nodes must be executable in parallel at every moment to keep all the processors fully utilized. In other words, a program will run only at 10% of it maximal speed if 100% efficiency of the multiprocessor is required. This relationship is shown in Figure 8, where a program profile with respect to number of parallel operations is shown. There are areas of high parallelism interleaved with areas in which only few operations can be executed in parallel. This kind of profile is the result of partitioning a large problem such as 2-D or 3-D simulations into smaller subproblems and then using one processor to solve each subproblem. The areas of low parallelism come from updating the points on the boundary of the subproblem (complexity $O(n)$) before computing the points inside (complexity $O(n^2)$). The architect must select a small number of processors for high efficiency. As the number of processors increases, the performance increases and efficiency drops. Therefore, to obtain high performance, we must tolerate some degree of inefficiency which can be minimized by not paying unnecessary scheduling and synchronization overhead for computation on regular structures.

5. Hierarchical Control

A task was defined in the previous section as a computation represented by a node in the control graph and scheduled as a unit. Partitioning problem deals with what comes into each node and can be divided into two subproblems. **Parallelism detection** determines all possible parallelism on the smallest level. **Clustering** combines several operations into tasks. Although tasks are indivisible from a scheduling point of view, they can be executed by several processors. A **process** is an indivisible unit with respect to processor allocation; that is, each process is executed on only one processor and each task consists of one or more processes. Processes can be combined into higher level structures. A *vector of processes* is an ordered set of non-interacting processes such as a DO loop in which no data is passed between iterations. In a *recurrence of processes* each i-th process supplies some data to (i+ 1)-th process. In a *two-sided recurrence* i-th process produces data for and consumes data from both (i-1)-th and (i+ 1)-th processes. Obviously, these ideas can be extended to higher dimensions.

At scheduling time a vector of processes can be allocated to n processors with the j-th processor ($j \leq n$) working on the j-th, (j+ n)-th, (j+ 2n)-th, ... processes, for example. Obviously other scheduling algorithms can be applied. On the other hand, one task can be just a random collection of interacting processes and still be assignable to more than one processor if the architecture provides a mechanism for it. Thus, four levels of control may exist on a multiprocessor architecture: *job, task, process,* and *instruction.* Very few machines have all these levels of control. In a batch system, jobs are running serially while in a time-sharing

environment, they are running in parallel. We will omit job level control in the following discussion. Each job consists of one or more tasks and each task consists of one or more processes while a process may have one or more instructions. A serial or parallel model of control can be used on each level.

A *serial single level* control, in which each node is single machine instruction, can be found in all von Neumann architectures such as VAX-11 or Motorola 68000. In this case, the entire program is a single process executed serially. Some data flow machines, such as the single-ring Manchester machine [WaGu82], have a *parallel single level* control in which each node of the control graph is a single machine instruction. In this case, there are no tasks and processes.

Cray-1 may be considered to have a *serial-parallel* control. As in von Neumann architecture each node is a single machine instruction. However, each vector instruction can be considered to be a vector of processes which is scheduled on x processors, where x is the number of pipeline stages in the functional unit. When vector instructions are considered, a x-stage pipeline is just a cost-reducing engineering trick to replace x independent processors. The NYU Ultracomputer has a *parallel-serial* control. Each node is a sequence of instructions called task specified by the programmer at the algorithm time. At runtime, the operating system will put all the active tasks in a queue in the shared memory. Whenever a processor becomes idle, it will get a new task from the top of the queue. Each task is executed serially in a processor. In NYU machine, task and process are the same and represent a node in the control graph.

The tagged token dataflow architecture proposed by Arvind [ArGo82] has a *parallel-parallel* two-level control in which the whole program graph is clustered into tasks called *code blocks*, each of which is another dataflow graph. Code blocks can be executed in parallel on the same or different set of processors, called a *physical domain*. There is no process in this architecture. However, each instruction inside a code block is allocated to a processor at compile time based on its iteration and statement number. Each processor contains a matching unit, a fetching unit, a program memory, one ALU, and a data memory (I-structure). Dataflow graph in each processor is executed in pipelined fashion which allows an increase in performance equivalent to the number of pipeline stages as long as there are sufficient number of executable nodes. It is not obvious what is gained by using dataflow model on a single processor that is more complex and costlier than a von Neumann machine of similar performance [Arla83].

The HEP machine has a *parallel-parallel-serial* control. Programmers specify tasks and processes inside a job. Tasks can be running in parallel in the same or different processors called PEMs, each task being allocated to only one PEM. Each PEM contains a task queue, a process queue, and several pipelined functional units each of which has 8 stages except for the division pipe. When a task is initiated in the HEP, a PEM is selected and the task status word (TSW) is stored into the task queue, while the initial process for this task is loaded into the process queue. Process can be created by another process of the same or different task.

All tasks in a PEM will be executed in a round-robin fashion. When a PEM executes a task, it will select a process from the task and send the current instruction of that process to one of the functional units. Afterwards, the PEM will switch to the next task in the task queue. If there are less than 8 active tasks in a PEM, several processes from the same task can be executed in parallel. When an instruction execution is finished, the process will be put back in the process queue and will wait for the next turn. The instruction execution inside a process is performed serially.

Cedar [GLKS84] also has a *parallel-parallel-serial* control. Tasks are represented by a node in a macro-dataflow graph and can be executed in parallel on different clusters of processors. Each task is a high-level structure of processes which is executed in parallel on several processors. Each process is executed serially on a standard von Neumann processor.

6. Scheduling

Scheduling is a function that associates one or more processor with each task in order to achieve high performance of a single program or high utilization of processors in a multiprogramming environment.

Scheduling can be done statically or dynamically. In **static scheduling** tasks are allocated to processors during the algorithm design by the user or at compile time by the compiler. The OCCAM language [Wils83] allows programmers to specify the instruction execution sequence, the channel of communication, and the execution unit. On the other hand, BULLDOG compiler [Fish83], after applying the *trace scheduling* technique to determine all the traces (tasks), performs register allocation and binds operations to specific functional units at compile time. The advantage is that scheduling cost is paid only once if the program is run many times with different data. Secondly, there is no run time overhead. The disadvantage of static scheduling is possible inefficiency in gussing the run-time profile of each task. For this reason, BULLDOG runs each program with a set of data in order to determine more accurately run time parameters.

Dynamic scheduling is done at run time by the machine. It offers better utilization of processors at the price of additional time needed for scheduling. The scheduling algorithm can be distributed or centralized. The NYU Ultracomputer [GGKM83] uses a distributed algorithm in which all tasks are in a queue in the shared memory and each processor takes the first task from the queue and executes it. The task queue is not a bottleneck since Ultracomputer uses its special synchronization instruction called Fetch&Add, which allows simultaneous access from all processors to the same memory location without performance degradation. Such a distributed algorithm allows architectural scalability at low cost with high scheduling-overhead penalty because of the global memory access through the network. On the other hand, Arvind's dataflow machine uses a centralized scheduler called 'manager' to schedule each code block to a physical domain.

The HEP machine uses a self-scheduling technique to balance the execution time of processes in a task. This method is very useful when the number of totally independent processes in a task, such as iterations in a DOALL loop, is significantly exceeds the number of processes allowable in a PEM; moreover, the execution time of each process may be varying widely because of the unpredictable delay of memory access through the network. Self-scheduling allows each process to acquire the next iteration dynamically when it finishes the previous one.

Some machines have more than one level of execution control. Different control levels may use different scheduling schemes. For instance, Cedar and Arvind's dataflow machine use dynamic scheduling for tasks, while processes and instructions are bound statically at compile time.

Different *dynamic* scheduling schemes such as Random Choice (RC), First Come First Serve (FCFS), Least Service Time First (LSTF), etc. can be used [HwSu83]. In case when a task is scheduled on more than one processor, a more sophisticated processor-allocation strategy is needed. In Cedar, for example, maximal number of processors needed by a task is determined at compile time. When the number of available processors at run time is not adequate, the scheduler can either *wait* or *fold* the task on a smaller number of processors. Simulation by Yew and Xu have shown that folding the task will provide a batter performance and processor utilization [YeXu84].

7. Synchronization

When executing a program in parallel, we need to synchronize the execution from time to time. Synchronization can be done either at control level or at data level and can be implemented either through shared-variable or through message-passing methods.

In the **Control-level synchronization**, a program counter is used to synchronize a sequential execution, while in parallel execution of a control graph, synchronization is done by allowing a node to execute only when all its predecessors have finished. All the sequential uniprocessors such as VAX-11 and Motorola 68000, use the program counter method while dataflow machines, such as Dennis's and Arvind's, use control synchronization for executing the flow graph.

The **Data-level synchronization** is used whenever synchronization is needed inside a node. It is very effective when a node represents operation on large structured data. For example, the computation of $\sum_{i=1}^{n}(a_i + b_i) * c_i$ is executed in two steps: vector addition followed by vector multiplication. However, the vector multiplication may start before vector addition is finished as long as we assume that consumer multiplication will not overrun producer addition. In Cray-1, this is called *chaining* and synchronization is accomplished through

synchronous operation (central clock) of two functional units: adder and multiplier. When vector instructions are replaced by general processes that cannot be executed in lock-step manner, a different mechanism must be used.

Different data synchronization primitives exist in different machines. Smith introduced a Full/Empty bit in each memory location in the HEP [Smit78]. Each register or memory word can be used to synchronize two processes in a producer-consumer fashion. This method is very elegant for 'single assignment' languages. However, for language that allow the reassignment of a variable, single Full/Empty bit can only synchronize alternating reads and writes to the same location. Arvind applied this synchronization method to the I-structure of his dataflow machine.

The primary purpose of all data synchronization schemes is to provide an efficient way of preserving a proper order of memory references. The Fetch&Add instruction denoted by F&A (V,e) in the NYU Ultracomputer performs an indivisible operations of fetching the integer variable V and replacing it by V+e [GGKM83]. It allows simultaneous operations on the same memory location by combining requests through switching elements in the connection network. This permits highly concurrent execution of operating system primitives, such as management of a parallel queue. However, the Fetch&Add is a commutative instruction and can not perserve the order in which memory is referenced.

Zhu & Yew introduced a synchronization scheme for Cedar [ZhYe84]. They define a key field for each synchronization variable and use that key as a counter. Each synchronizing instruction will test the key and perform a memory read or write only if the tested condition is satisfied. After this operation, the key is incremented or decremented in order to allow the next operation on the same variable. The counter method of Zhu & Yew is very difficult to preserve the reference order when the counter is updated by two or more overlaping sequences of memory operations. For example, if each processor computes one grid point (Figure 9), then each grid value such as A is read twice by B and C before a new value can be written into A. This is repeated on each r iteration. If processor computing C is delayed by one iteration, then processor computing B will decrement the counter of A twice and allow updating of A out of order. Two errors are made: B gets an old value of A twice while C will compute with a new value of A. A Bit-map synchronization method [Peir83] solves this problem by treating the key field as multiple Full/Empty bits. Each memory operations is associated with one bit in the key. By testing and seting the bits in the key, a proper order can be preserved in this and similar examples.

8. Memory Access

As we mentioned in section 2, each multiprocessor must include a connection network which introduces unpredictable delay in accessing data stored in the shared or distributed memory. Furthermore, since access to certain memory can be blocked temporarily, the arrival

order may be different from the order in which data were requested.

Memory latency problem can be tolerated by using a data-driven, message-passing method so that data will be fetched at the generation time instead of the demand time. All the dataflow machines use this method. Machines such as Cray-1, fetch vector data in a pipeline fashion. After an initial set-up delay, data arrive to the processor at pipeline time. A latency problem still exists when accessing non-vectorized data. High-performance von Neumann processors, such as IBM 360/91, use runtime instruction lookahead to solve this latency bottleneck. However, this introduces an extra complexity in the control unit. The HEP and the NYU Ultracomputer have multiple processes in each processor and perform context switching while one process waits for the data from memory. An extra set of register files is needed for each active process to avoid overhead of saving registers. The Structure Memory Access Architecture [PlDa83] introduced a fetch and an execution processors in each conventional processor. Address generation, data fetch, and operation execution can be overlapped to increase the performance. However, a data dependence problem is not easy to solve between these fetch and execution processors.

9. Summary

In this paper, we identified the issues in solving a problem on a multiprocessor machine. We discussed how to decompose such a problem, schedule and execute it using many processors. We reviewed several commercial and proposed machines and discussed approaches they use to accomplish this.

10. References

[AbGa84] S. Abraham, and D. Gajski, "A Communication Algorithm for a Wafer Scale Integrated Multiprocessor," Conf. on Parallel Processing, Bellaire, MI, 1984.

[ArGo82] Arvind, and K. P. Gostelow, "The U-interpreter," IEEE Computer, Vol.15, No.2, 1982, pp. 42-49.

[ArIa83] Arvind and R. A. Iannucci, "A Critique of Multiprocessing von Neumann Style," 10th Symp. on Computer Architecture, Stocholm, 1983, pp. 426-436.

[ArTh80] Arvind, and R.E. Thomas, "I-structure: An Efficient Data Type for Functional Language," Tech. Rep. TM-178, Lab. for Computer Science, MIT, Cambridge, Mass., Sep 1980.

[Brow84] J. C. Brown, "TRAC: An Environment for Parallel Computing," COMPCON, Spring, 1984, pp. 294-298.

[Denn80] J. B. Dennis, "Dataflow Supercomputer," IEEE Computer, Vol. 13, No. 11, 1980, pp. 48-56.

[Farm84] P.M. Farmwald, "The S-1 Mark IIA Supercomputer," in High-Speed Computations (J. S. Kowalik ed.), Springer-Verlag, 1984.

[FiOD84] J. A. Fisher, and J. J. O'Donnell, "VLIW Machines: Multiprocessors We Can Acturally Program," COMPCON, Spring, 1984, pp 299-305.

[Fish83] J. A. Fisher, "Very Long Instruction Word Architectures and the ELI 512," 10th Symp. on Comp.

Arch., Stocholm, June 83, pp 140-150.

[Flyn72] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," IEEE Trans. Computers, Vol C-21, No 9, 1972, pp. 948-960.

[Fox 84] G. C. Fox, "Concurrent Processing for Scientific Calculations," COMPCON, Spring, 1984, pp. 70-73.

[GaRo84] D. Gannon, and J.V. Rosendale, "Parallel Architectures for Interative Methods on Adaptive, Block Structured Grids," Elliptic Problems solvers (G. Barkoff, Ed.), to appear.

[GGKM83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, M. Snir, "The NYU Ultracomputer," IEEE Trans. Computers, Vol. C-32, No. 2, 1983, pp. 175-189.

[GLKS83] D. Gajski, D. Lawrie, D. Kuck, and A. Sameh, "Cedar," COMPCON, Spring, 1984, pp. 306-309.

[HwSu83] K. Huang, and S. P. Su, "Priority Scheduling in Event-driven Dataflow Computers," TR-EE 83-36, School of Elec. Eng., Purdue University, Dec. 1983.

[Kuch78] D. J. Kuck, "The Structure of Computers and Computations: Volume One," John Wiley & Sons Inc., New York, 1978.

[Kung82] H. T. Kung, "Why Systolic Architectures?," IEEE Computer, Vol.15, No. 1, 1982, pp. 37-46.

[PaKL80] D. A. Padua, D. J. Kuck, and D. L. Lawrie, "High Speed Multiprocessor and Compilation Techniques," IEEE Trans. Computers, Vol. C-29, No.9, 1980, pp. 763-776.

[Peir83] J-K Peir, "An Efficient Synchronization Method for Multiprocessor Systems," Cedar Document No. 27, Lab. for Advanced Supercomputers, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Dec. 1983

[PlDa83] A. R. Pleskun, and E. S. Davidson, "Structured Memory Access Architecture," Conf. on Parallel Processing, Aug. pp. 461-471.

[PSSP84] Y. N. Patt, R. G. Sheldon, M. Shebanow, C. Ponder, and W. Hwu, "A Comparison of Several Evolving Supercomputer Architectures," 4th Jermsalem Conf. on Information Technology, May 1984.

[Seit84] C. L. Seitz, "Experiments with VLSI Ensemble Machines," J. of VLSI and Computer Systems, Vol. 1, No. 3, 1984.

[Smit78] B. J. Smith, "A Pipelined Shared Resource MIMD Computer," Conf. on Parallel Processing, 1978, pp. 6-8.

[Snyd82] L. Snyder, "Introduction to the Configurable, Hoghly Parallel Computer," IEEE Computer, Vol. 15, No. 1, 1982, pp. 47-56.

[Trel82] P. C. Treleaven, D. R. Brownbridge, and R. P. Hopkins, "Data-Driven and Demand-Driven Computer Architecture," ACM Computing Surveys, Vol. 14, No. 1, 1982, pp. 93-143.

[YeXu84] P. C. Yew, and Q. X. Xu, "Simulations and Analysis for a Multiprocessor System with Multiprogramming," 1st Conf. on Computers and Applications, Peking, China, June 1984.

[WaGu82] I. Watson, and J. Guad, "A Practical Dataflow Computer," IEEE Computer, Vol. 15, No. 2, 1982, pp. 51-57.

[Wils83] P. Wilson, "OCCAM Architecture Eases System Design - Part I," Computer Design, Nov.1983, pp. 107-115.

[ZhYe84] C-Q Zhu and P-C Yew, "A Synchronization Scheme and Its Applications for Large Scale Multiprocessor Systems," Conf. on Distributed Computing Systems, San Francisco, May, 1984, pp. 486-491.

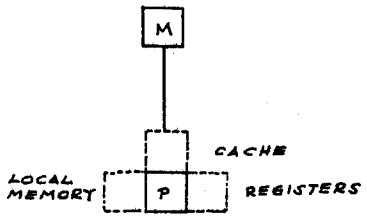


Figure 1. Von Neumann Model

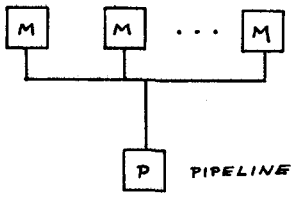


Figure 2. Vector-Machine Model

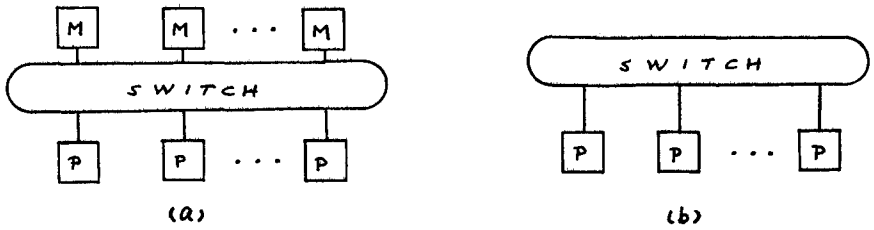


Figure 3. Multiprocessor Model: (a) Shared Memory; (b) Message Passing

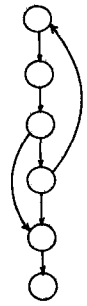


Figure 4. Serial Model of Computation

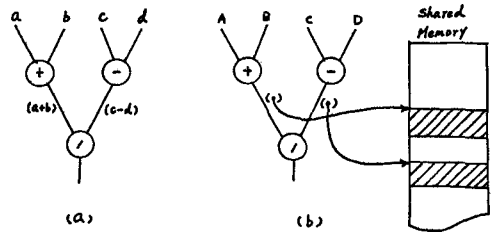


Figure 5. Parallel Model of Computation:
(a) Data-driven; (b) Control-driven

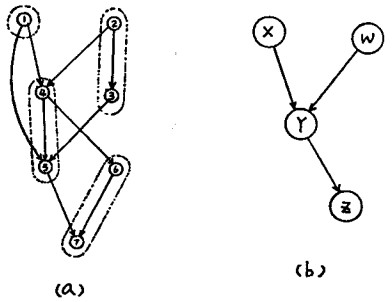


Figure 6. Control Graph:
(a) Fine Granularity; (b) Crude Granularity

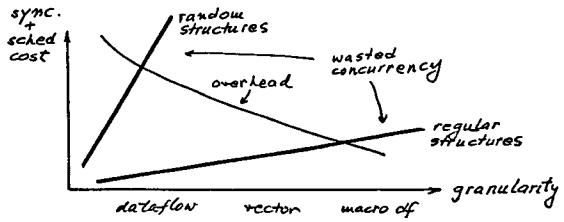


Figure 7. Concurrency and Overhead Plots

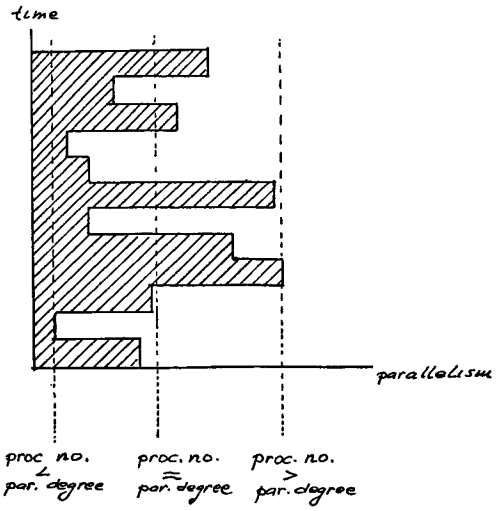


Figure 8. Program Profile with Respeat to Parallelism

```
DO r = 1, TIMES
   DO i = 1, N
      DO j = 1, N
         A(i,j) = ((A(i+1,j) + A(i,j+1)) / 2.
      ENDO
   ENDO
ENDO
```

(a)

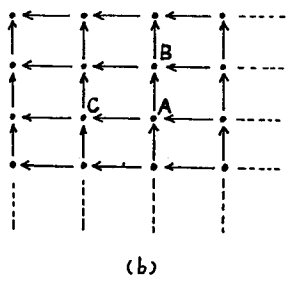


Figure 9. A Finite Difference Problem:
(a) Source Code; (b) Data Passing

QUESTIONS AND ANSWERS

Q: Can your CEDAR structure handle list-processing problems more-or-less efficiently?

T. Brody

A: Present processor is not a LISP machine. If we replace it with a LISP processor and figure how to decompose LISP programs, I believe it will run efficiently.

Q: What are the kind of computational problems which are well suited to the architecture of CEDAR?

R. Brower

A: We believe that a large spectrum of problems are suited for CEDAR, since CEDAR provides fast access to shared memory, good process synchronization, static and dynamic scheduling, restructuring compiler and macro data flow model of computation.

Q: How much overhead do you expect for fetching data and load module to a particular cluster?

K. Miura

A: Such overhead is inevitable in any multi processor architecture. Multi-programming may solve such problems for CEDAR.

Q: Can one avoid creating new languages and rather define a set of extensions to an existing language (FORTRAN) for partitioning, scheduling, etc.?

T. Nash

A: When I say "new languages" I mean mostly extensions to existing languages to support scheduling, synchronization, etc.

Q: Experience, though limited, indicates that not a large number of extensions are needed to control parallel or multi-processor operations from FORTRAN. But the specific nature of this extension is currently somewhat dependent on the specific architectural structure. Therefore, specifying a completely general set of FORTRAN.extension for parallel systems is probably presently impractical - but it is tantalizingly close.

C. Maples

A: I agree.

Q: When will CEDAR be operational and which physics and engineering problems will you attack with it to demonstrate the effectiveness of the approach?

M.J. Levine

A: CEDAR may be operational in 1986 or 1987 if someone would support it. We really have not done enough work, yet, looking into applications.

EXPERIENCE WITH SCIENTIFIC APPLICATIIONS ON THE MIDAS MULTIPROCESSOR SYSTEM*

Creve Maples
Lawrence Berkeley Laboratory, University of California Berkeley, California 94720

Introduction**

Most experts agree that the speed of standard or 'serial' computers is approaching fundamental physical limitations imposed by signal propagation (the speed of light) and heat dissipation. This is evidenced by the fact that the fastest modern serial computers are only 2 to 3 times faster than the CDC 7600 introduced in 1968. Although it is possible to increase present serial processing speeds by a factor of 10 to 50, such increases will require new technological advances (gallium arsenide, Josephson junction, etc.) and will probably be relatively expensive.

The Cray-1 computer, introduced in 1976, attempted to circumvent these obstacles by performing various identical operations in parallel (vector processing). Suppose, for example, a problem required that two sets of 50 numbers be added, by pairs, and the 50 results were, respectively, to be multiplied by a third set of values. A vector machine would use 50 separate addition units and multiplication units and perform the entire operation in two steps rather than 100. For the portion of a problem that can be organized in such a manner, this technique yields substantial increases in speed over the traditional serial machine. If, however, it is necessary to examine the result of each addition, for example to determine an appropriate normalization factor, the performance of the vector processor would diminish considerably. Software effort in both the United States and Japan is being directed towards the development of compilers which attempt to reorganize programs in order to attain more effective utilization of vector machines. The extent, however, to which problems are amenable to this approach varies considerably and a great many important physical problems appear to be essentially non-vectorizable. The advances in micro-electronics during the past decade have made new approaches to computing feasible. This involves using multiple computers, or processing elements, collectively on the same problem. Unlike the vector approach, such processors are not constrained to perform the same operations at the same time and, may, for example, be working on different facets of a calculation simultaneously. Whatever speed individual serial computers ultimately achieve, it is clear that multiprocessor machines could potentially extend this speed by factors of hundreds or thousands. Although a number of manufacturers now provide systems which incorporate, or can be expanded to include, several processors, such systems are designed primarily to run separate problems and not to work collectively on the same program. Some new systems (such as Denelcor's HEP, CDC's Cyber Plus, Cray's X-MP, and ELXSI's System 6400) are beginning to offer coordinated processing capability. The potential power of multiprocessing computers, however, still remains virtually unexplored within the commercial

*This work supported by Director, Office of Energy Research, Div. of Nuclear Physics, Office of High Energy and Nuclear Physics and Nuclear Science of the Basic Energy Science Program, U.S. Dept. of Energy, Contract No. DE-A03-76SF0098.

**Some of the material in this presentation has been extracted from papers appearing in the proceedings of the 13th International Conference on Parallel Computing (1984).

marketplace. This is primarily due to the difficulty in architecturally organizing many independent processors so that they may work productively towards the same goal. The main problem areas in this respect are in multiprocessor control, coordination, and inter-processor communication.

Although many designs for such multiprocessor systems exist, few have actually been constructed. The reason for this is both the expense involved in constructing such a system and the fact that many proposed designs are directed at specific classes of problems and hence may not be commercially viable. The new industrial consortium, the Micro-electronic and Computer Corporation, currently involving over a dozen corporations, is vitally concerned with the lack of development in this area. A recent MCC report stated that "To date, attempts to improve performance through highly parallel structures have been relatively disappointing. We believe the major reason for this lack of progress is the high real and personnel cost to build and evaluate parallel structures." This fact, coupled with newly expanding computer applications in both business and personal computing, make high-risk, high-priced ventures appear unnecessary.

Contention and problem decomposition are two areas which can pose significant difficulties in the successful design and application of multiprocessor systems. Contention between processors for memory (or bus) access can drastically affect the performance of a system. As an example of this, Digital Equipment Corporation's announcement of the shared memory, two processor VAX 11/782 (Feb. 1, 1982) indicated that system would only provide between 60 to 80% speedup over the single processor system. This loss of performance was due to contention even though the two processors were not intended to work on the same problem.

The use of global memory versus hierarchial, partitioned, or local memory is the subject of considerable discussion. Some projects, such as the S1 System[1] and NYU's Ultra Computer,[2] believe that a general computer must permit any processor to access any memory at any time. Although this is extremely desirable, it leads to problems of contention and conflict, where two or more processors want to access the same location simultaneously. Conflict can occur if even one of the processor accesses is a write operation. The situation rapidly deteriorates as the number of processors increases. A simulation of the expected performance of the S1 system on the hydrodynamic test code SIMPLE is shown in Table 1. An 8 processor system is expected to perform at 48% efficiency. The equivalent of 2 processors is expected to be lost due to global memory conflicts and one due to interprocessor line transfers. The authors conclude[3] that there "is little to gain by running this problem on more than 8 processors." Other projects, such as the Cosmic Cube,[4] avoid this problem by not having any global memory. In this case,

Table 1.
Simulated Performance of S1 on SIMPLE*

No. Processors	*Speedup*	*Efficiency*
1	1.00	1.00
2	1.77	0.89
4	2.93	0.73
6	3.56	0.59
8	3.84	0.48

*T. S. Axelrod, et al.,
Lawrence Livermore Laboratory, Livermore, CA[3]

Table 2.
Ware's Model of Multiprocessors

no. processors (P):	Relative Speedup (Efficiency)			
	2	8	16	100
% serial code (α)				
1 %	0.99	7.5 (0.93)	14.0 (0.87)	50 (0.50)
5 %	0.95	6.0 (0.74)	9.1 (0.57)	17 (0.17)
10 %	0.91	4.7 (0.59)	6.4 (0.40)	9 (0.09)
20 %	0.83	3.3 (0.42)	4.0 (0.25)	5 (0.05)

$$S(P) = \frac{1}{\alpha + (1-\alpha)/P} = \text{speedup}$$

$$E = S(P)/P = \text{efficiency}$$

however, significant amounts of interprocessor communication can severely degrade the system's performance.

The efficient decomposition of a problem can, independently of the multiprocessor structure, be crucial to the performance of the system. This can be seen in the Ware Model for parallel systems. In this simple model, it is assumed that either one processor or all processors are operating, and that all effects of interprocessor communication can be neglected. In Table 2 alpha represents that fraction of code (in terms of time) that cannot be executed in parallel. Then Table 2 illustrates the effect of this parameter on performance as the number of processors increases. A very small amount (1%) of serial computation obviously has little effect on the efficiency of systems with only a few processors. In a 100 processor system, however, 1% of serial code would lead to a 50% performance efficiency, or the loss of 50 processor equivalents. This illustrates the necessity of developing highly parallel problem decompositions in order to be able to utilize multiprocesors extensibly. Table 2 also illustrates that performance on a few processor system is relatively insenistive to the amount of serial code. This is important since many tests are currently being carried out in 2 to 4 processor environments. Such tests need to realize very high performance efficiencies (better than about 98%) if the approaches are to be successfully extended.

The results of some actual multiprocessor tests are shown in Table 3 for the 2 processor CRAY X-MP and four processor ELXSI System 6400. The test results were reported by Melvin Scott of Sandia National Laboratory.[5] The code SPEED consists of 5 independent kernals exerpted from Sandia's mathematical library and from large user codes. It is designed to measure the ability of the machine to

Table 3.
Speedup Factors on CRAY X-MP and ELXSI 6400*

Code	*Machine*	*Number of Processors*		
		2	*3*	*4*
SPEED	X-MP	1.995	-	-
	ELXSI	1.994	2.74	3.03
BENNEU	X-MP	1.556	-	-
	ELXSI	1.995	2.85	-
SUPORT	X-MP	1.735	-	-

*Melvin Scott, Applied Mathematics Division, Sandia National Laboratories, Albuquerque, NM[6]

perform floating point calculations. BENNEU is part of a particle-in-cell code. It was used for testing because it required the separate tasks to share common memory. The program SUPPORT is a large code which solves two-point boundary-value problems using methods of superposition and orthonormalization. These tests illustrate all the effects of contention and code on efficiency. As expected, efficiency drops rapidly as the number of processors increases. The difference in relative performance between the X-MP and ELXSI on BENNEU is most probably due to the architectural structure of shared memory. Unfortunately no information could be obtained that indicates specifically the causes of the performance loss.

System Overview

Focusing the power of many independent processors so that they may be effectively applied to single problems or applications is not easy and is the subject of a great deal of current research. The **MIDAS** system, under development at the University of California's Lawrence Berkeley Laboratory, is based on the concurrent operation of multiple asynchronous processors. The control architecture is a hierarchy of computer processors, organized in a general tree-structure (as illustrated in Figure 1) and integrated with independent 'intelligent' mass storage and interactive systems.[6,7] Within the present 3-level system, multiple processing elements are organized into clusters, each of which is controlled by an auxiliary computer. Every cluster combines central processing units from commercial computers with independently developed specialized processors and a specially designed communications system.[8] Multiple clusters are in turn controlled by a computer referred to as the Primary. The three processing levels, therefore, consist of a Primary Computer, Secondary Computers, and Multiple Processor Arrays. Interprocessor communication can be handled in a variety of ways which

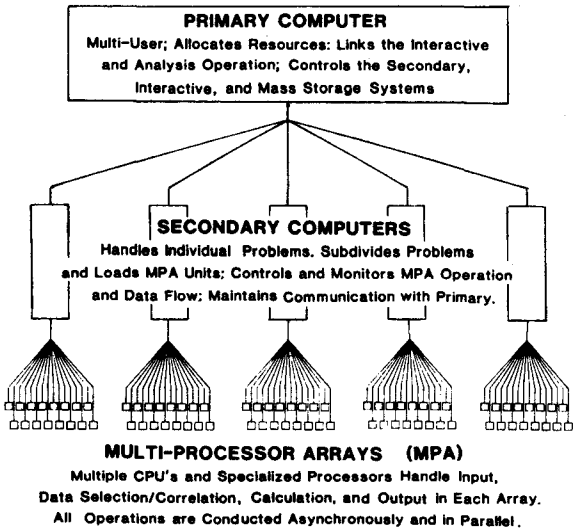


Figure 1 MIDAS processor organization showing the three levels of operation and control.

will be outlined in the following material. The discussion will primarily focus on the operation and performance of a single subsystem at the second and third levels.

A. Processor Organization

Processors within the MIDAS architecture are organized into groups or clusters called Multi-Processor Arrays (MPA). Each MPA consists of a variety of independent processors, multiple busses, a high-speed switching network, and a set of independent switchable memory blocks. At present the processors include an Input Processor, and Output Processor, an array of independent CPUs, and a set of units called Zero Processors (ZP). An MPA is, in turn, controlled and monitored by a commercial mini-computer called a Secondary. This two-level processing structure is referred to as a Distributed Subsystem and forms the basic processing unit of the system. A single subsystem containing 11 independent processors is illustrated in Figure 2. From a control perspective, this architectural organization is similar to that proposed by the Cedar project at the University of Illinois.[9] In the Cedar design a cluster of 8 to 16 processors is controlled by a single cluster control unit (and multiple clusters, by a global control unit). The two designs, however, have significant differences with respect to communication, control, and the type of processors utilized.

The Multiprocessor Array on MIDAS, shown as part of Figure 2, currently consists of eight general purpose CPUs referred to as Programmable Arithmetic Modules (PAMs). Each of these units is, in fact, a standard commercial CPU with dedicated memory, capable of handling scientific calculations in general, and

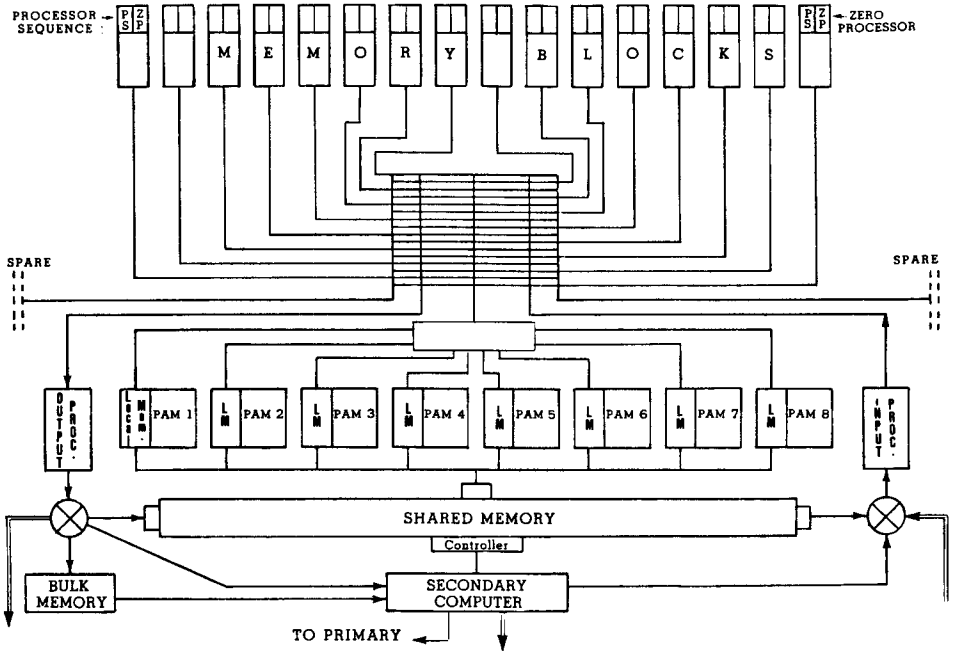


Figure 2. A Distributed Subsystem containing eleven independent processors.

floating point operations and Fortran codes in particular. For the initial development the ModComp 7870 CPUs were selected. These processors support 64-bit floating-point hardware, pipelined operation, and up to 4 Mbytes of local memory. The CPUs are, for comparison, roughly 15% slower than the DEC VAX 11/780. Ultimately a single MPA is expected to contain between 24 and 32 of these independent processors, each of which is perhaps four times the speed of the present units.

The design facilitates the easy integration of specialized processing elements. Two such elements, the Input and Output Processors are shown in Figure 2. These are specialized pipelined devices designed to handle information flow into and out of the cluster. They operate independently at a 200 nsec. clock cycle on two separate, external, 20 Mbytes/sec. I/O-busses (32-bit data, 8-bit control). These processors may, depending on their programming, select or reject information (filtering); expand or compress data (format); manipulate data (mask, shift, etc.); or route specified information as required by other processors in the cluster. Due to the pipelined structure, these operations are all performed at bus speeds. The need for specialized handling of I/O operations is also recognized in the Cedar architecture. That design, however, includes a separate processor cluster which is specialized for I/O operations instead of, as in case of MIDAS, utilizing special-purpose I/O processors within each cluster.

The Secondary CPU is responsible for supervising the operation of a Multiprocessor Array. Each has dedicated disc drives and can, therefore, compile, assemble, and link programs. To facilitate supervisory functions, the console-control functions of all the standard CPUs (in the MPA) are interfaced into the Secondary, as shown in Figure 3. This provides it with the complete capability to monitor and control third level processors. It can run, halt, resume, or single-step each PAM independently or collectively, and can monitor or modify selected registers or memory locations. Examples of effectively utilizing this capability will be discussed in the section on Performance.

The function of the Secondary is both varied and problem dependent. It can directly control processing at the third level of the system, function as an intermediary between the Primary Computer and the third level, and/or directly participate in the calculation itself. A Secondary could, for example, co-ordinate its computation with third-level processing in a master-slave relationship, using these processors to perform needed calculations as appropriate, and in parallel. In this mode it can obtain and act on results as they become available and perform or delegate further processing, as required, until the problem is completed. Alternatively, it may only be necessary for the Secondary to set up a problem for actual execution entirely on the third level. A data analysis problem is an example of the latter mode of operation. Initially the task would execute serially in the Secondary to perform the setup and initialization of the problem (e.g., calibration, etc.). Thereafter independent data events would be analysed in parallel at the third level. In such cases the Secondary might, during processing, be employed to monitor and/or control the overall performance kinetics within the third level, and to handle any abnormal conditions which might occur (informing the Primary Computer, if necessary). Additionally, it may dynamically allocate processor functions as the requirements of the executing problem change.

The top level, and overall master, of the system is a Primary Computer whose main function is to handle system control and user communication. It implements instructions in a job control language which permits the user to define the problem

requirements and to allocate necessary resources. Interactive control of a task's execution is also provided at this level since the Primary can access the instantaneous results of the calculations ongoing at the lower levels.

B. Memory Configuration

Currently four classes of memory are included in the Distributed Subsystem design shown in Figure 2. These include local memory, switched memory, shared (or global) memory, and bulk memory. Each of these has different attributes and collectively can be employed both to minimize contention and to facilitate interprocessor communication. The utilization of these memories will be discussed in the section on Multiprocessor operation. Local memories, as indicated in Figures 1 and 2, are dedicated to a single processor and are not directly accessible by other units. Figure 4 illustrates the schematic layout of memory from the perspective of a processor (or program). The Secondary also possesses its own local memory. In addition, however, it has access to both the global shared memory and the bulk memory units which will be discussed.

Each of the sixteen independent switchable Memory Blocks, shown in Figures 2 and 3, has a dedicated memory bus and may contain up to 256 Kbytes of memory. A 5 x 16 crossbar switch allows any memory module to be dynamically attached to any of the five processor busses shown. Since information transfer between a memory module and a CPU (PAM) is considerably faster than the cycle time of the CPU, it was possible to time-multiplex 8 independent memory-CPU connections on the same bus with essentially no degradation of access time. Time-multiplexing these connections was an implementation, not an architectural, decision. Functionally the multiplexed unit operates as a 12 x 16 crossbar switch. Any Memory Block may thus be attached to any processor at any time. Switching a memory module between available processors requires about 50 nsec. Once a processor-memory connection occurs, there is no functional distinction between the switched memory and the processor's local dedicated memory. The memory module is accessed by standard load and store instructions, rather than by I/O commands. Thus from a programmer's point of view, the switched memory is simply a particular common block. This use of bank-switched memory units in

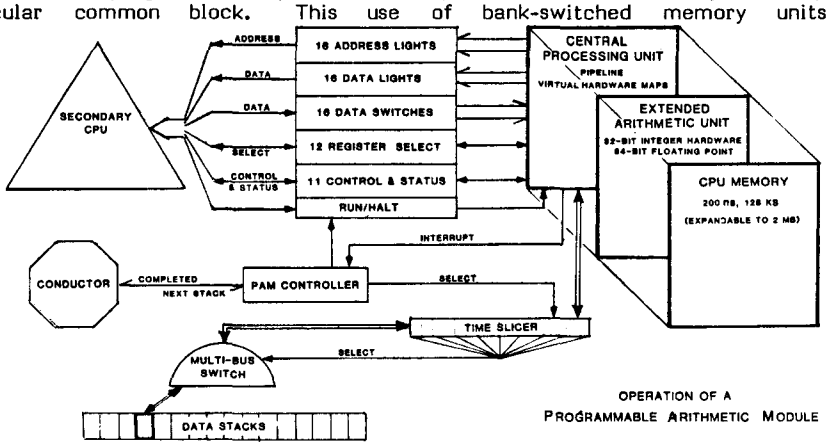


Figure 3. Schematic representation of the control and monitoring interconnections between the Secondary CPU and a third level PAM.

multiprocessor configurations is similar to the S-1 Multiprocessor architecture under development at Lawrence Livermore Laboratory.[1] The present S-1 design employs 16 memory banks and 16 processors, and any processor may access any memory at any time. Unlike the MIDAS design, the S-1 processors use these common memory banks both for program operation and data storage, although access is enhanced by using individual cache memories. Simultaneous memory access conflicts, which can result from this scheme, are identified by special hardware circuits.

MIDAS also provides a global, or cluster-wide, shared-memory unit (Figure 2). Access to this memory is given on the basis of a demand queue. For store operations longer than a single word, a processor may lock out other processors until all memory updates are completed. Since heavy utilization of the global shared memory can slow the parallel operation of the processors, it should be used judiciously. The serious consequences of memory contention problems which can arise from over-dependence on shared memory was discussed in the Introduction. The programs currently running on MIDAS (and described in the section on Performance) have thus far not required the use of global memory.

An independent bulk memory unit, with a 32 Mbyte capacity, is also available for data storage. CPU (PAM) access to this unit is indirect in that information must be transferred via the switchable memory modules. This mode of accessing bulk memory is quite efficient with respect to CPU utilization since a PAM continues operation immediately after releasing a memory module, and is not forced to wait until the data transfer to bulk memory is complete. The bulk memory has dual ports and can be utilized either in a standard DMA transfer or in an address-incrementing (+1) mode.

C. Communication

A Distributed Subsystem is connected to the rest of the processing environment by four separate data paths, as shown in Figure 2. Two independent busses, as indicated previously, are used to handle high-speed data flow into and out of the Multiprocessor Array. The remaining links connect the Secondary to the Primary Computer and to a global switching network. This switching network is controlled by the Primary and serves to interconnect clusters and the global mass storage system. Between the two levels of the subsystem, separate communication channels connect all processing elements of the MPA with the Secondary Computer.

Communication between units within an MPA may be handled in a variety of ways. Information may be broadcast to all processors via the shared memory.

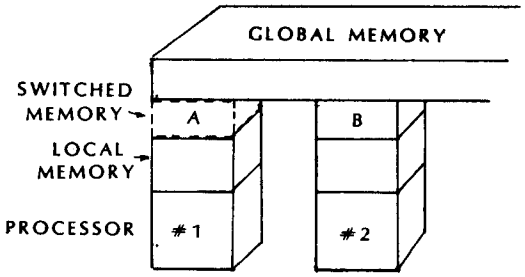


Figure 4. Memory layout from the perspective of a processor or program.

Specific processor communication (particularly control information or imperative commands) may be sent via the Secondary. Finally high-speed communication can be obtained by utilizing controlled access to the switched Memory Blocks. This communications mechanism can take the form of either circuit switched or packet switched operation and will be discussed.

A standard technique for handling interprocessor communication is by utilizing a common memory that all CPUs (third level in this case) have equal priority access to. Typically there is some queuing discipline manifested in hardware and/or software that ensures memory coherence. What this usually implies is that CPU A may not alter the contents of a memory location that CPU B is in the process of reading. Under these conditions, the system is termed tightly coupled. Such systems may exhibit severe degradation of performance in the event of a high rate of requests from all CPUs to common memory. Attempts to alleviate this problem through intermediate private cache memories must first solve the difficult cache coherency problem, and thereafter may still retain a degradation of performance as cache misses increase with total memory requests.[3] Problems that wish to employ this mode of communication must be decomposed in such a manner that this bottleneck does not represent the kinetic step of the solution. Mechanisms such as replace and add[2] can be employed to minimize conflict and to ensure load balancing.

D. Specialized Processors

MIDAS offers the user the ability to employ specialized hardware processors which can assume functions of code that would otherwise need to be executed in the CPUs. The ability of the Input Processor to preprocess the incoming data stream provides a simple example of this capability. In many analysis problems, for instance, this information may consist of compressed data containing only non-zero values. Reconstructing such compressed data is typically accomplished by using descriptive information contained within the data. In standard programs this information must first be decoded and then the data expanded into an array in its original form. In MIDAS, however, the expansion algorithm may, if well defined, be executed at bus speeds by the Input Processor and the equivalent code deleted from the program. In this example the reconstructed information would be placed directly in a switchable memory block at appropriate locations. The time required to complete analysis on a given body of information would thus be shortened by that fraction of the time previously associated with executing the deleted code. Analogously, sorting, filtering, shifting, masking, and similar operations may be programmatically handled in parallel on separate processors operating at high speeds.

This example illustrates an important feature of the MIDAS design - that specialized hardware processors may easily be accommodated within a more general computing structure. Such devices can effectively be accessed, if required, as program subroutines. Utilization of these processors is thus flexible and codes may be both simplified and speeded up by their use. They may be independently programmed and their application controlled or defined by a threading sequence, which may be as complex or as simple as the particular problem demands. The future inclusion of additional hardware processors (such as boolean logic units, array processors, track reconstruction processors, and other specialized devices), coupled with the ability to dynamically redefine the threading sequence on data-dependent conditions, provides exceptional performance potential.

Multiprocessor Control

The distribution of a problem among processing elements is currently handled by specific constructs within the program and by interaction with the user (via job control structures, etc.).[10] In order to achieve an efficient and flexible operational environment and to permit fault-tolerant error recovery, a functionally distributed operating system structure was developed.[11] System functions were distributed among specialized hardware devices (including microprocessor control units) as well as to the software in different processors. Since there are frequently several ways to delegate specific system responsibilities, often at the discretion of the user, a high degree of flexibility is maintained and a certain degree of fail-safe operation made possible.

The switched memory provides an example of distributed functions and multiprocessor control. The Secondary Computer is responsible for overseeing the operation and utilization of the switched memory (including the handling of failures). It, however, is too slow to directly control the high-speed, asynchronous memory switching required in the MPA. The actual switching of memory modules is handled by a special hardware device termed the Conductor. The Conductor is functionally controlled (and even programmed) by the Secondary. It can, when requested, supply the Secondary with detailed information on system activity.

A. Macroscopic Data Flow

Each memory module is equipped with both zeroing hardware and a directory indicating the processor sequence or destination (Fig. 2). Thus when a module is released by a processor, the directory pointer is incremented and the memory is switched to the next class of processor specified. If all processing elements of the required class are busy, the memory will remain unattached until one becomes available. The ability of each Memory Block to carry with it an independent processing sequence permits the Multiprocessor Array to function in what might be described as a 'macroscopic data flow' mode of operation (for a discussion of data flow, see, for example, Ref. 12).

When MIDAS operates in this manner, system software prohibits a memory module from being simultaneously accessed by more than one processor. A processor, therefore, has exclusive access to the attached memory until it relinquishes the module (or until the supervisory CPU forces a relinguish). This avoids conflicts and processor contention problems. In this approach, a particular Memory Block is transferred from one type of processing element to another in a manner and at a speed dictated by the information it contains. Note that in this mode of operation, Memory Blocks are switched to the first available processor of the type required, not in general to a particular processing element. This is important since the number and classes of processors available may vary dynamically. Reassignment of processor function during problem execution may be carried out either by the Secondary or by a processor itself and facilitates load balancing on the system.

The exact manner in which this cluster of processors and memory blocks co-ordinate their activity is flexible and may be defined through a user-specified threading sequence. The following limited example of a data analysis problem serves to illustrate one such sequence and some of the capabilities of the various processors. Execution of a data analysis program would begin in a Secondary

Computer. During this setup phase the program obtains the basic descriptive information on the problem (from the user or elsewhere), performs any requisite initialization, and establishes all calibration information needed for subsequent analysis. Up to this point execution of the code is serial. After completing initialization, in this example, exact binary copies of the program (as it exists in memory) are downloaded into each CPU in the Multiprocessor Array. Execution of the program on these CPUs then proceeds both asynchronously and in parallel.

The functions of input and output operations for the Multiprocessor Array are assumed by the Input and Output Processors respectively. Information is carried by switchable memory blocks. Zeroing memory in any data stack may be carried out by special hardware Zero Processors. A threading sequence determines the processor order in which a memory block is switched through the system. In this example, a user specified threading sequence of Zero Processor, Input Formatter, CPUs, and Output Formatter designates that all memory blocks will be connected in a cyclic manner to each of these processor classes (in this example all CPUs are assumed to be identical). Thus each block would first be attached to a Zero Processor to be cleared, and then switched to the Input Formatter.

The Input processor is designed to receive external data whose description, characteristics, and origin are specified initially at the job control level. This processor may simply store this data directly into attached memory blocks or, alternatively, process the data in a specified fashion prior to storage. The filled block would, as defined by the threading sequence, then be attached to any available CPU requesting data. After connection the memory block appears to the CPU (and resident programs) as directly addressable memory, thus permitting instant access without any of the attendant dead time that input or output usually implies (a memory block switch requires about 50 ns).

After completing the calculations, an analysis code typically stores the results back into its current memory block and the program requests more data. This memory block (with results) would, as specified by the threading sequence, then be attached to the Output Formatter. This processor may block or unblock information, as required, and transmit the results to to a variety of user (or code) specified output devices. On completion of this final process, the entire threading sequence is repeated. All processors in the sequence are, of course, operating independently and in parallel. The actual memory block/processor cycle and co-ordination is controlled by hardware and supervised by a microprocessor called the Conductor.

B. Circuit and Packet Switched Operation

Operation of the system can also take the form of a circuit-switched network in which interprocessor connections can be made or broken as necessary to accomodate exchanges of information. In this environment the switched memories play a static role in that they serve as communication pipelines. The processors play the active role and switch between specific Memory Blocks to establish (buffered) communication channels. In 'data-flow' operation, the system software prohibits multiple processors from simultaneously attaching to the same memory. Exactly the opposite is true in this case, since any number of processors may be attached to the same Memory Block. Access to the memory is controlled by system routines which select and partition the memory according to the global interconnection requirements requested. A dedicated region of a particular Memory Block is established to handle communication between two specific processors. This

region is, in turn, divided into two parts to separate read and write operations - the write area for processor A is the read area for processor B and vice versa. Utilizing this mechanism, it is possible to create most connection topologies. Partitioning a Memory Block into four regions (8 parts) permits four processors to be connected in a square lattice (with a minimum communication bandwidth of 5 Mbytes/sec). Six such Blocks could then be used to form a cubic topology such as employed in the Cosmic Cube.[4]

The memory partitioning is, of course, transparent to the user. A connection topology is supplied to the Secondary. System handlers which utilize this topology are then loaded with the problem. If a processor wishes to communicate with another, the sender's ID, the receiver's ID, and whether the operation is a read or write, is supplied in a call to the system routines. These routines determine the appropriate Memory Block, the address within the block, and, if necessary, request a system interconnect to the block. In this mode of operation Memory Blocks themselves are thus invisible to the user. The system support for the mode of operation is, at present, not complete.

The system can also be utilized in a packet-switched mode of operation. In this case the Memory Blocks are used to carry information between processors and, more importantly, between processor classes. After placing information in an attached memory, the processor would store the destination code in the memory sequence directory associated with the Memory Block (Figure 2). A system call would release the memory which would be directed by the system hardware to the first available processor of the class specified. If none were available, the block would remain queued until satisfied. If all processors are different classes, then information can be directed to a specific processor and two way exchanges are possible.

Single Cluster Performance

The original development of the MIDAS project began in the fall of 1979. A prototype was operational in January 1982 which consisted of 4 CPUs, 8 memory modules, input and output processors, a bulk memory unit, and the high-speed switching hardware (Conductor). This system was used to test the basic switching network, the communication system and the control capabilities. A single complete cluster, similar to the configuration shown in Figure 2, was completed in February 1983. This system has been used for performance studies, software development (system, language, and application), and to investigate the application of real problems to a multiprocessor structure. Although some modifications[10] of the original programs were required to operate in a parallel environment, these changes generally were not extensive.

To date a variety of codes have been run on the MIDAS system, with requirements varying in characteristic from I/O intensive to CPU intensive. In the following we describe two general classes of problems which serve to illustrate important areas of programming for this system. The first is a scientific data analysis program. This example illustrates the programmed use of the bank switched memories for problems that have high I/O requirements and how the inclusion of hardware processors within the threading sequence may enhance solution performance. The second area deals more with a class of problems that may be loosely grouped under the heading of Monte Carlo codes. These examples will serve to show how coordinated calculation between the Secondary and the third levels may be achieved.

A. Analysis Problems

One class of problem examined involved scientific data analysis. These problems are usually characterized by requiring frequent input of information, utilizing moderately heavy integer and floating point calculation, and requiring access to large global arrays. Interprocessor communication requirements are nil and output requirements may vary from little to significant. Typical results of one such test are shown in Table 4. This illustrates how the problem execution time and external I/O requirements vary with the number of processors. If processor contention is negligible and each processor is able to contribute its complete capability towards the solution of the problem, the relative speed would simply equal the number of CPUs employed. As indicated, the speed increases observed in the problem tracked exactly with the number of CPUs used in MIDAS. This result was, with one exception, obtained in all the analysis problems examined.

The single exception consisted of a gamma-ray analysis program written in a language called EVAL.[13] This program primarily sorted large volumes of data and required only logical and integer operations. The test results for this problem are given in Table 5. Using more than 6 PAM units failed to produce any significant increase in performance. Analysis of this situation indicated that the commercial disc controller used in the test was not able to supply information fast enough to keep up with MIDAS's processing capability. This controller was unable to sustain a continuous data transfer rate of greater than 640 Kbytes/sec, and six processors were sufficient to completely handle this rate of information transfer.

The error bar indicated in Table 5 represents the uncertainty (or lack of reproducibility) of the execution time due to random factors (e.g., disc latency, head position, etc.) in the measurement. These factors become more important as the amount of disc I/O increases. The fact that relative speeds are slightly greater than n, the number of processors, is not considered statistically significant although it does suggest that the pipelined I/O processors in MIDAS may become more efficient as the rate of data transfer increases. Possible restrictions in handling highly I/O-intensive problems were anticipated, and will be alleviated when the construction of a specially designed Multiported Programmable Controller is completed. This unit is part of the planned parallel Mass Storage Subsystem. The controller features 3 independent channels of look-ahead, dual-track buffering and is

Table 4

MIDAS Phase 2 - Relative Performance

Problem: Average CPU and I/O Mix

	Time (sec.)	Relative Speed	I/O Rate (KB/sec.)
1 PAM	372	(1)	34
2 PAMs	186	2.0	69
3 PAMs	124	3.0	104
4 PAMs	93	4.0	139
5 PAMs	74	5.0	174
6 PAMs	62	6.0	209
7 PAMs	53	7.0	243
8 PAMs	46	8.0	277

expected to sustain transfer rates of over a Mbyte/sec. per drive.

One Fortran analysis program, typically requiring about an hour of CDC 7600 CPU time per 1600 BPI tape of data, was adapted to the MIDAS system. This code was used to perform the initial analysis of data collected by the LBL/GSI Plastic Ball.[14] The program analyzes up to 3000 parameters which are measured every time a relevant event occurs within the spherical system. This analysis essentially includes the reconstruction of the physical occurrence, involving the determination of particle identities, energies and spatial co-ordinates within the ball. It is heavily dependent on floating point calculations and was used to benchmark the current MIDAS system. Details of Fortran extensions implemented on the MIDAS system and the software modifications necessary to adapt this program have been discussed elsewhere.[10] Three tests were conducted with this program. In the first case the code was converted with minimal changes, and specialized processors were not used. Under these conditions MIDAS executed the program at 70% the speed of the CDC 7600. (The CDC execution times measured only CPU seconds and excluded both system overhead and I/O time; the MIDAS times were total processing time, including I/O.) This code was then modified slightly (about 10 Fortran lines) to utilize the hardware zeroing processors and buffering capabilities of the architecture. This modified code executed at 87% the speed of the CDC 7600. The final modification involved using the Input Processor to carry out the formatting and expansion operations (on the initial compressed data) that originally was performed in the program. This function is done in parallel with the CPU operations, and the corresponding code was deleted in the program. Under these conditions, MIDAS will perform about 16% faster than the CDC 7600. In each of these cases the relative MIDAS speedup equaled the number of CPUs utilized.

B. Monte Carlo Simulation Problems

There has, in recent years, been considerable discussion as to whether Monte Carlo programs could be efficiently converted to parallel operation on SIMD or vector architectures. The Monte Carlo approach is a technique, not a program, and whether a program utilizing this technique is amenable to SIMD (or MIMD) decomposition depends primarily on the application itself and how the technique is

Table 5

MIDAS Phase 2 - Relative Performance

Problem: I/O Intensive

	Time (sec.)	Relative Speed	I/O Rate (KB/sec.)
1 PAM	130	(1)	99
2 PAMs	61.4	2.1	210
3 PAMs	41.2	3.1	314
4 PAMs	30.6	4.2	420
5 PAMs	25.5	5.1	510
6 PAMs	21.0 ±0.5	6.2 ±0.2	620
7 PAMs	(20.3	6.4	640)*
8 PAMs	(20.3	6.4	640)*

*Performance limited by commercial disc controller

employed. It is usually true, however, that Monte Carlo techniques tend not to run efficiently on vector processors.

Several Monte Carlo programs were investigated and adapted for the MIDAS system. The results for three such problems will be briefly examined. The first used Monte Carlo techniques to study reactions in a many-body system. The distinctive aspect of this problem was that calculation produced relatively frequent bursts of information at random intervals which had to be stored on disc. In an asynchronous environment, frequent random output requests from the different processors could lead to instantaneous bus contention or delays. This difficulty was avoided in MIDAS by utilizing the switched Memory Blocks to derandomize the requests. The actual output was handled by the Output Processor from completed Memory Blocks, and in parallel with CPU activity. The CPUs simply switched to an empty block and continued operation. Test results showed the problem executed with 100% efficiency on the single cluster system (8-fold speed up for 8 processors). Since each cluster contains its own independent I/O busses and Output Processors, the architecture would be extensible for this problem.

The second case involved a simulation of the decay of equilibrium and non-equilibrium dinuclear systems. In contrast, this problem involved essentially no I/O activity. A distinctive feature was that it required frequent random access to large arrays. Although the problem itself was easy to decompose into independent parallel code, the requirement that all processors have full access to a single global memory could not be partitioned - either in terms of time or locality. This is due to the nature of the Monte Carlo technique itself and meant that each processor would frequently, at random intervals, need to address and update random memory locations. Classic problems of memory contention and conflict arise when more than one processor simultaneously attempt to access or update the same location. Indeed the intelligent "fetch-and-add" capability of NYU's Ultracomputer project[2] was specifically designed to avoid such conflicts.

These problems were avoided in MIDAS by using the Bulk Memory unit (Figure 1) to store the global arrays. Contention was resolved by using the switched memory in each processor to buffer multiple address requests, and conflict was eliminated by using the Output Processor to serialize the actual memory operations. Since this is a 20 Mbyte/sec. pipelined processor, it is capable of simultaneously reading a 32-bit word of information from a switched Memory Block, processing information (e.g., routing, reformatting, calculating address offsets, etc., if required), and outputting a word of information, every 200 nsec. Information in the bulk memory unit can be updated efficiently since this device has a read-modify-write cycle. Thus in the MIDAS implementation of this problem, contention was avoided, CPUs were never delayed waiting for pending requests, and the system was again able to operate at 100% efficiency and delivered an 8-fold speed up with 8 processors.

The third problem of this type calculated the eigenvalues of many electron systems by utilizing a Monte Carlo approach to the solution of the time-dependent Schrodinger equation. The problem had essentially no I/O requirements and could be decomposed so that memory accesses could be localized to each processor. This problem required neither the switched memories nor any of the specialized auxiliary processors. It was decomposed into a master/slave configuration with the Secondary Computer as master. All communication was carried out between the master and respective slave units.[10] Details of the implementation of this problem have been described elsewhere. It is mentioned here because the initial

implementation of the problem required periodic synchronization of all parallel units by the master. With this condition, test results indicated the system performed at 95% efficiency with eight processors (or a speedup equivalent to 7.6 processors).

A test case of the calculation was performed for H_3, an important transition complex whose eigenvalue determines the rate of free radical exchange in the reaction $H + H_2 \rightarrow H_2 + H$. On a VAX-780 this calculation took 434 hours of CPU time to reach an error bar on the eigenvalue of 0.0004 hartrees - a level of confidence that is needed to make thermodynamic estimates on rates. The same calculation, implemented as indicated above on MIDAS, using 8 CPUs on the third level, took 67 hours to reach the same statistical significance, a factor of 6.5 times faster. This speedup is consistent with previously benchmarked estimates of the speed of individual MIDAS processors (ModComp 7860) versus the VAX-780, i.e., the 7860 is about 85% the speed of the 780 for this code. A similar calculation for the N_2 molecule, which is currently in progress, is estimated to require approximately 1000 hours of MIDAS time to reach needed statistics. The addition of more memory to the processors (currently in progress) should permit the problem to be handled in such a way that only one global synchronization will be necessary. In this case the efficiency should be essentially 100%.

Future Directions

Test results indicate that the I/O capability of a single cluster can support more than the present 8 processors. The number of processors can easily be doubled or tripled by duplicating the current time-multiplexed handler on the two unused busses (Figure 2). New and faster CPUs will also be used to replace the current processors. The addition of other special-purpose processors, including array processors, is also under investigation. These devices can perform specific calculations, or algorithms, which are not amenable to parallel approaches (either vector or multiprocessor). Such processors can easily be incorporated on a vacant memory bus (Figure 2) and accessed through a switched memory from code in a manner analogous to a hardware subroutine.

The performance discussed thus far has been for a single cluster. Plans are underway to develop the 3-level structure which will be able to accomodate between 5 and 10 multiprocessor subsystems (as illustrated in Figure 5). By controlling the second-level switching network, the Primary can then use multiple clusters on individual problems in a similar fashion to the way a Secondary uses multiple processors. The subsystems could either be working on independent problems or different aspects of the same problem, depending on the setup conditions specified by the Primary, and like individual processors within a subsystem, they can be flexibly employed in any parallel or pipelined configuration. A full three level system could support up to 270 processors. Using more than about 10 subsystems might require adding a fourth level to the system. The Phase 3 effort will, in addition, require the development of a multi-bussed, parallel-processor mass storage environment and a high-speed, interactive system.

Summary

The objective of research on the MIDAS project is to demonstrate the viability of a multiprocessor approach to computing and to develop a general purpose, extensible architecture which can be used to address the growing computational requirements of the scientific community. To be successful in this endeavor, however, requires more than simply designing, or even constructing, new hardware structures. To achieve high performance on future systems will require software

approaches which can exploit parallel architectures as fully as possible. A critical issue, therefore, is to understand the functional requirements of a large class of applications. The requirements must be critically examined and, in many cases, new approaches to old algorithms investigated. For highly parallel systems to be effectively utilized, they must be flexible and adaptable to specific application requirements. In particular, they will probably need a variety of control and communication mechanisms to facilitate load balancing of a problem and to minimize bottlenecks in performance.

The effective utilization of highly parallel architectures, however, raises many new questions. Traditional operating system structures must, for example, be re-examined. Fault-tolerant environments, with error recovery capability, become increasingly important as the number of components and processors increases. Language extensions to support parallel operations are obviously required, and ultimately new languages are needed to explicitly exploit parallel constructs. A new generation of development and debugging tools will be necessary in order to examine program performance and operation in an asynchronous parallel environment. Even areas such as mass storage must be re-examined to accomodate the high-speed performance implicit in the new systems. Finally, new computer architectures must take into account the growing demand for highly-interactive computing environments. The architecture itself should reflect this capability in order to minimize its impact on system performance.

The development of the MIDAS structure has, to some extent, required

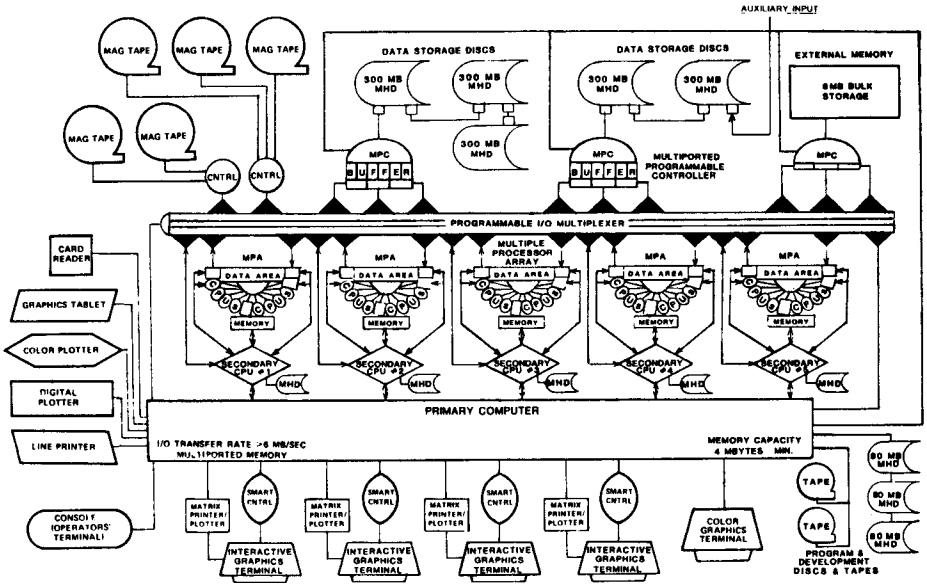


Figure 5. A three-level structure containing 5 multiprocessor systems.

investigation into all of these areas. The current 11 processor cluster supports a distributed operating system and has demonstrated the ability, under program control, to recover from processor failures. New tools have been developed to facilitate program development and debugging. A structured environment permits highly interactive code to directly interact with running problems in such a way that neither system nor problem performance is degraded. Difficulties in conventional mass storage technology have been encountered and parallel approaches to the problem are being developed.

A variety of existing applications programs have been investigated and language extensions developed to both Fortran and XPL. The system performance for these problems (which had I/O requirements spanning several orders of magnitude) was found to be approximately equal to the CDC 7600. The problems examined initially exhibited a high degree of inherent parallelism. In these cases the system performed at 100% efficiency, achieving n-fold speedup for n processors. A few problems with lower inherent parallelism exhibit less than n-fold speed up. These problems are being critically analyzed to determine which approaches can achieve maximum parallel efficiency and the architectural implications of those techniques. Based on detailed performance measurements with a variety of real data analysis programs (and on simulations projecting these results to systems with a large number of processors), we find that the MIDAS architecture will perform at 100% efficiency and is completely extensible.

The MIDAS project should not, however, be viewed as a particular architecture, but rather as a system which is evolving both in terms of performance and applications as the results of experimental tests and investigation dictate. Work, in collaboration with a number of scientific research groups, continues on applying as diverse a mix of problems as possible on MIDAS. The system's evolution will, to a great extent, depend upon achieving success in such disparate areas as AI applications, economic modeling, tomographic imaging, hydrodynamic modeling and lattice-gauge theory. Codes in all of these areas are currently either being closely examined or actively run. The main questions addressed are of fundamental importance to any system that is parallel in nature, i.e., how do the problems decompose and what mechanisms are available to effect parallel problem solving. The main objective is to provide a structure that will not only run these problems, but is extensible and capable of providing hundred and thousands of times today's computing power. To this end, a collaborative research effort is also being conducted with Digital Equipment Corporation on the development of Applied Multiprocessor Architecture. Such joint efforts are important if we wish to bridge the gap to the next generation of computing.

References

1. L. Curtis Widdoes, Jr. and Steven Correll, "The S-1 Project: Developing High-Performance Digital Computers," Energy and Technology Review, Sept., 1979; Ronald D. Levine, "Supercomputers," Scientific American, 118 (Jan. 1982).

2. Allen Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAulifle, Larry Rudolph and Marc Snir, "The NYU Ultra Computer - Designing a MIMD Shared Memory Parallel Computer," IEEE Transactions on Computing, C-32, 175 (1983).

3. T. S. Axelrod, P. F. Dubois, and P. G. Eltgroth, "A Simulator for MIMD Performance Prediction - Application to the S-1 MKIIa Multiprocessor," Proceedings of the 12th International Conference on Parallel Processing, 350 (1983).

4. Geoffry C. Fox, "Scientific Calculations with Ensemble Computers," Proceedings of the "Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics," Padova, Italy, 115 (March 1983).

5. Melvin Scott, presented at the Conference on Experiences in Applying Parallel Processors to Scientific Computation, Glendenen Beach, Oregon, March 12, 1984 (unpublished).

6. Creve Maples, Daniel Weaver, Douglas Logan, and William Rathbun, "Performance of a Modular Interactive Data Analysis System (MIDAS)," Proceedings of the 12th International Conference on Parallel Processing, 350 (1983).

7. Creve Maples, William Rathbun, Daniel Weaver, and John Meng, "The Design of MIDAS - A Modular Interactive Data Analysis System," IEEE Trans. on Nuclear Science, NS-28, 3746 (1981).

8. W. Rathbun, C. Maples, J. Meng, and D. Weaver, "A Fast Time-Sliced Multiple Data Bus Structure For Overlapping I/O and CPU Operations," IEEE Trans. on Nuclear Science, NS-28, 3875 (1981).

9. Daniel Gajski, David Kuck, Duncan Laurie, and Ahmed Sameh, "Cedar - A Large Scale Multiprocessor," Proceedings of the 12th International Conference on Parallel Processing, 524, (1983).

10. Douglas Logan, Creve Maples, Daniel Weaver, and William Rathbun, "Adapting Scientific Programs to the MIDAS Multiprocessor System," Proceedings of the 13th International Conference on Parallel Processing (1984).

11. Daniel Weaver, Creve Maples, John Meng, and William Rathbun, "Operating System Considerations in the Multiprocessor MIDAS Environment," IEEE Transactions on Nuclear Science, NS-30, 3980 (1983).

12. Jack B. Dennis, "The Varieties of Data Flow Computers," Proceedings of the First International Conference on Distributed Computing Systems, 430 (1979).

13. Anders Holm, "Construction of Efficient Nuclear Event Analysis Programs Using a Simple Dedicated Language," IEEE Transactions on Nuclear Science, NS-26, 4569 (1979).

14. "The Plastic Ball Spectrometer: An Electronic 4π Detector with Particle Identification," A. Baden, H. H. Gutbrod, H. Lohner, M. R. Maier, A. M. Poskanzer, T. Renner, H. Riedesel, H. G. Ritter, H. Spieler, A. Warwick, F. Werk, and H. Wieman, Nuc. Inst. and Meth., 203, 189 (1982)

15. "Quantum Monte Carlo Study of the Classical Barrier Height for the $H + H_2$ Exchange Reaction: Restricted Versus Unrestricted Trial Functions," P. J. Reynolds, R. N. Barnett, and W. A. Lester, Jr., to be published, Int. J. Quant. Chem., Symp 1984, No. 18 (1984).

QUESTIONS AND ANSWERS

Q: What does the cost estimate include?

W. van Rüden

A: Let me answer in several parts. There is about 20 man years of effort and $350K in the present system. This includes large disc drives, 6250 BPI tape, high resolution color graphics terminal, etc. Each processor, however, is only about $10K. The cost of adding 8 more processors to the system is about $90K. One should also remember that the total cost I quoted of $350K is a 1980 figure. The cost of memory has dropped significantly in this time span and would be much cheaper to duplicate now.

Q: What and how many CPU's are you now using and planning to use in your next phase? What production running has been done on the existing system?

T. Nash

A: We are currently using a total of 10 Mod Comp Classic 2 CPU's with 64-bit floating point and capable of holding 4MB of memory. They are about 15% slower than the VAX 11/780. Eight more processors are on order and will be added to the existing cluster. In about 12 months the current cluster will be stabilized and used only for production. A new cluster will be built at this point with 20-30 processors, each of which will be about 4x the current processor speed - a total increase of about an order of magnitude. We will then begin multi-cluster operation for another order of magnitude increase. What processor we will use is not determined at this point since most of the likely candidates are not currently running. Two possible candidates are obviously the $308/_E$ or the super VAX processors. We have an open mind on this. MIDAS has been doing production running for about the last 9 months. Within the past 3 months it has been devoted 100% to production, 24 hours/day. This cannot continue, however, since we will be upgrading the cluster in the near future. The problems on the system have included a variety of data analysis programs, Monte Carlo simulations, quantum chemistry calculations, multi-dimensional minimization, free search, etc. There is currently a waiting list to get on the system.

Q: When you increase the number of processors, does cost (including extending I/O) go up linearly, or worse?

M. Fischler

A: Better than linear.

Q: What effort was required to modify the operating system to take advantage of the parallel architecture?

A. Brenner

A: 3 man years.

WORK IN AMSTERDAM ON LOCAL INTELLIGENCE

J.Dorenbosch (1), D.Gosman (1), L.O.Hertzberger (2),
D.J.Holthuizen (1), F.Tuynman (2), J.C.Vermeulen (1), Presenter

(1) NIKHEF-H
(2) Facultaire Vakgroep Informatica, University of Amsterdam
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands

1. The FAMP system

Work on local intelligence at NIKHEF-H was started in 1979 with a proposal to design a multi microprocessor system for real-time event selection in high-energy physics experiments. The outcome of this is the Fast Amsterdam Multi Processor (FAMP) system [1] which has the following properties:

- a high degree of modularity, facilitating easy system expansion
- the possibility to realize hierarchical structures of processing elements
- use of commercially available microprocessors

The system consists of modules that mechanically fit into a CAMAC crate. From the CAMAC standard only the CAMAC power supply lines are used. A special backplane, forming the internal system bus, has been designed for communication between modules. A processing cell consists of a CPU module (based on the Motorola MC68000) and other system units like memories and interfaces. Only one CPU module can be connected to the internal bus; more than one processing cell can be fitted into one crate, provided that the internal bus is divided and properly terminated. Processing cells are connected with each other via dual port memories (DPM's) (fig. 1), that are constructed as separate modules. The DPM's can be accessed via the internal system bus and via the so-called external bus, which consists of a 64 lines flat cable. The CPU module has also access to the external bus via a front panel connector.

A three processor FAMP configuration has been used for real-time event selection in a study of inclusive phi-meson production in the NA11/NA32 experiment at the CERN SPS [2]. A seven processor configuration, which will be further extended this year, is installed in the UA1 experiment at the SPS. This last application will be discussed in chapter 2. Because of its general nature and the availibility of good software development facilities, FAMP is also used for other applications than real-time event selection, for example:

- A FAMP configuration with special interfaces is used as a fast data-link between the NIKHEF-H CYBER-173 and NORD-100 computers. The NORD-100 is used as the front-end machine for the CYBER-173. In the near future personal workstations will be coupled to this configuration via an Ethernet connection. The communication software is written in C.
- A FAMP configuration, extended with a video display controller, is used for interactive graphics analysis of experimental data of the NA11 experiment. The physics analysis program, written in FORTRAN, was originally used on a mainframe computer with a TEKTRONIX storage display as graphics device. The graphics routines needed were also written in C.
- The Computer Science Department of the University of Amsterdam will make use of a 5 processor FAMP configuration as a "testbed" system for research on multi processor systems.

For all these applications UNIX® is used as the software support environment because it has become a standard especially for 16 bit microprocessors. To improve the real-time run-time environment the FAMP Distributed Operating System (FADOS) [3] was written for the FAMP multi processor environment. FADOS has been used for the last two applications described above and will be discussed in chapter 3.

® UNIX is a trademark of Bell Laboratories

2 The FAMP system in the UA1 experiment

2.1 Present situation

The UA1 experiment [4] is installed at the CERN SPS proton-antiproton collider. Proton-antiproton collisions take place in the center of the UA1 detector at energies of 540 GeV. The FAMP system is used as part of the muon detection system. Muons that are produced traverse several drift chambers, an electromagnetic and a hadronic calorimeter and additional absorber steel. They are identified by drift chambers that enclose the full detector. Muon tracks are measured in two projections. The drift times are measured by so-called Multiple Time Digitizers (MTD's).
The FAMP system has several functions:
- Read-out of the drift times in the MTD's.
- Second level triggering on muons.
- Monitoring of the behaviour of the muon chambers.

At present a configuration of six slave processors and one supervisor is used (figure 2). Each slave reads the data from several MTD crates with a FRI (FAMP REMUS Interface). The distribution of the MTD crates is optimised to spread the data load evenly over all slave processors. The data are transferred to the supervisor via DPM's (Dual Port Memories). An operating system [1] has been developed supporting the communication between the processors. The data are passed to the Data Aquisition System under supervisor control via a FAMP-ROMULUS Output Buffer (FROB). Communication with the trigger supervisor proceeds via I/O registers in the supervisor system. Switches are provided that can connect a back-up readout system to the MTD crates.

For the second level trigger decision two projections of the tracks are reconstructed. Data from the different projections are read by different slave processors, where track reconstruction is done in parallel. Information on tracks found in the projections are passed to the supervisor, again via the DPM's. The supervisor combines the projections in space. Tracks that point to the interaction point in the center of the detector are assumed to be genuine muons. Events with such tracks are accepted. The total decision time of the second level trigger is of the order of several milliseconds. Most of this time is used to sort the muon data by wire number. Because the event rate has been relatively low until now (a few Hz), no rejection was necessary and accordingly FAMP has only been used to flag events with a muon candidate.

Monitoring of the behaviour of the chambers is performed by background programs that run in the supervisor processor. A sample of the event data is analysed and information is stored in histograms in local memory. These histograms can be inspected via a terminal connected to the supervisor or by the on-line NORD computer.

2.2 Future developments in the UA1 FAMP system

In figure 3 the FAMP configuration as planned for the future is shown. The slaves are equipped with hardwired coprocessors (RM: Reordering Memories) that read the data from a MTD crate and sort the data by wire number. This takes about 20 microseconds. Thereafter the MTD's are available to accept the next event. This double buffering method reduces the deadtime of the experiment at high event rates. It also frees the processing power in the slave processors to perform more elaborate trackfinding algorithms.

In the future system the data will be passed to the data aquisition system via FAMP-REMUS (FAREM) interfaces containing among others a 2 Kwords deep FIFO memory. The data transfer will be interrupt driven via I/O units. Data from the streamer tubes that are added to improve the UA1 muon detection will also be passed to the FAMP system. This will be realized via a dual port memory coupling (not shown in the figure). An

extra slave system will be added to provide an independent back-up readout system. Data from the MTD crates can be passed to it via the switches. In the back-up system data can be reordered under software control; no track finding will be attempted. The monitor task will be moved to an extra processor system running under FADOS. This operating system will be discussed in chapter 3. The monitor processor is connected to the supervisor via a dual port memory. Event data and trigger decisions are passed to the monitor program via the DPM. This opens the possibility to monitor both the behaviour of the muon chambers and of the second level trigger. An MC68000-based system running under UNIX will be connected to this monitoring system to improve program development facilities, to store monitoring results and to act as a user interface.

In figure 4 a comparison, as determined from a Monte Carlo simulation, is presented of the data taking efficiency of the UA1 data acquisition system with and without the use of a second level trigger. For the event rates anticipated for the future (50 - 100 Hz) local event selection yields an improved data-taking efficiency, because no useless data have to be transported.

3. The FAMP Distributed Operating System (FADOS)

3.1. Overview

FADOS [3] is a real-time operating system for multi processor systems operating in close interaction with a UNIX host. Using a host-target approach for software development it creates a friendly programming- and test environment. FADOS supports access to a remote file system on the host running UNIX, inter-process communication (also between processes running under UNIX and processes running in the real-time system) by message passing and interrupt handling by user processes. It provides facilities for the loading of cooperating processes. At present support for debugging is tested. FADOS can be used on any FAMP configuration provided that one of the processors is connected via a dual port memory to a host computer running UNIX. Because inter-process communication is done by message passing, other methods of coupling processing cells, such as Local Area Network couplings, can in principle be supported. The use of message passing also allows communication between arbitrary processing cells which do not have to be coupled directly, i.e. the inter-process communication is independent of the system configuration. The initial version of FADOS was supporting a terminal connection to a Motorola EXORmacs Ⓡ computer with UNIX V7 as operating system.

The FADOS software (fig. 5) consists of a part residing in ROM on the target system and another part on the UNIX host. The ROM contains:
- A real-time nucleus which takes care of message passing between processes. It also includes a driver for a local terminal. The messages which have to be sent to this driver are identical to those for the file server.
- A load program

A debug monitor has been developed, and is at present under test.

On the UNIX host the following software is available:
- A file server which operates under UNIX as an ordinary user program.
- A "command interpreter" operating together with the load programs in the target processors, providing facilities for loading and killing of processes.
- A modified version of the UNIX startup file and a modified LIBC library containing the message sending variants of the UNIX file system calls.

A debugger has been developed and is at present under test.

Ⓡ EXORmacs is a trademark of Motorola

The file server, command interpreter and debugger are comprised in a driver, added to the UNIX kernel.

3.2 Access to the UNIX file system and automatic loading

Important facilities that FADOS offers are complete access to the UNIX file system from any target processor as well as automatic loading of files from the host. This enables users to work with FADOS as if it is an extension of the UNIX system. The access to the UNIX file system is completely transparent for the user because the normal I/O statements can be used in a program written in a High Level Language running under FADOS. This is realised by translating UNIX file system calls to messages to the UNIX host. A file server on the host interpretes these messages and takes the appropriate action. Terminal I/O can be done to a terminal connected to the UNIX host or to a local terminal whose behaviour is similar to that of a UNIX file.

FADOS makes it possible to load a group of independently compiled programs as cooperating processes. When a process is loaded it receives a unique identification from the FADOS nucleus. When the load program loads a group of cooperating processes it sends all identifications to all processes. These identifications have to be used as addresses for the messages.

3.3 Inter-Process communication in FADOS

Cooperating processes communicate with each other by exchanging messages. It does not matter whether these processes have been loaded on the same or different processors. In defining the message passing primitives it has been attempted to provide as much as possible the same facilities as those offered by the ADA® [5] programming language. In FADOS the following primitives are used:

- send: send a message and wait for a reply
- receive: wait for a message
- reply: send a reply

When sending a message it should be given a name. When issuing a receive call, the names of the messages one wants to receive have to be specified. Messages with other names are queued. It is not possible to wait for messages from a specific source. The messages have a maximum size of 512 bytes. After a receive call the size of the message received, the name of the message and the source (to be used in a corresponding reply call) are available in global variables.

3.4 Interrupts

Interrupt handling has been integrated with process communication. This is accomplished by transforming an interrupt into a message to a process. Before a process can receive an interrupt with a particular interrupt vector as message it has to call the nucleus in order to indicate that it wishes to do so. This call has as effect that the process will be awakened after a receive call not only when a message arrives, but also when the interrupt occurs. After the interrupt the process will start to execute immediately.

3.5 Scheduling

The nucleus uses a simple algorithm for scheduling. It maintains a ready queue in which all processes not waiting for a message or a reply are placed. The process at the head of the ready queue will actually run on the processor. When a process is waiting for a message (not caused by an interrupt) or a reply it will be placed at the tail of the ready queue as soon as a message or a reply is received. When a process receives a message caused by an interrupt it will be placed at the head of the ready queue and start to execute immediately. A more complicated scheduling mechanism, where tasks get different software priorities has been developed and is at present under test.

® ADA is a reg. trademark of the US Government, ADA Joint Program Office

4. Future systems

4.1 Local intelligence and data-acquistion systems

The application of FAMP in UA1 illustrates the possible tasks of local intelligence in a data-acquisition system. In such a system the following tasks can be distinguished :

- data transport, data manipulation (sparse data scanning, reformatting, reordering, conversion) and control of buffering
- triggering and on-line analysis
- monitoring (including monitoring of the triggering and data transport processes) and calibration.

From our experience with the FAMP system we believe that it is desirable to perform these tasks locally (fig. 6,7). Our experience with FAMP has also shown that it is attractive to couple processing cells to each other with dual port memories. Moreover, in order to avoid bus contention multiple data pathes within a processing cell may be desirable, causing a demand for multiple-port memories. A communication link with a coordinator which receives the results of the triggering processes and which commands which data should be transported should exist. For monitoring only a subsample of the events is needed; however, access to the original data as well as the processed data should be possible. It should also be possible to modify the monitoring programs without disturbing the data transport and triggering processes. A Local Area Network connection offers good possibilities for communication between the real-time environment and the program development environment, which can also be used as user interface and for storage of monitoring and calibration results.

4.2 Hardware aspects

At present standardized busses, like VME and FASTBUS, are available that support multiple busmasters. In FASTBUS also crate interconnection facilities are defined. For VME and other industrial bus standards no such facilities exist, however multiple information-paths are sometimes defined (e.g. VMX and VMS for the VME-bus). For moderate speed requirements Local Area Networks may solve the problems due to the lack of standardization of crate interconnections. Powerful processors like the MC68000, the MC68010, the NS16032 and the NS32032 are available. Furthermore an entire new generation of processors is appearing at the horizon: e.g. the Motorola MC68020, the Inmos T424, the Zilog Z80000, the Intel iAPX386. It can be expected that the introduction of those processors will soon lead to processor cards for the industrial busses, like VME.

4.3 Software aspects

At present advanced programming environments (UNIX, APSE [6]) are or are becoming available. Many High Level Languages can be supported (C, PASCAL, FORTRAN-77, MODULA-2, ADA). The industrial push for use of High Level Languages on advanced microprocessors causes the development of better and cheaper compilers. Furthermore personal computer systems based on microprocessors with sophisticated and cheap software tools are appearing on the market. Programming in High Level Languages creates some overhead, but with the new generation of fast processors this does not need to be problematic. For FADOS, which is written completely in C, the overhead is estimated to cause a loss less than 25% in execution speed.

The use of multi processor systems for which all processors are contributing to the same real-time task gives rise to the need for a real-time kernel supporting multiple processes running on multiple processors as well as for software controlling the multi processor system. Real-time kernels which support multiple processes running a

single processor, like RMS68K Ⓡ, are available. FADOS has also facilities for inter-process communication between processes running on different processors and is a step in the direction of the integration of a real-time embedded system with a sophisticated programming environment. As such it is better adapted to the needs in high-energy physics experiments. On top of such a system the need for a software layer can be envisaged, that should e.g. take care of data transport across the data-acquisition system, still allowing manipulation of the data, and of making the data available for monitoring, triggering and calibration purposes.

References

1. L.O.Hertzberger et al., Proc. of the three day in-depth review on the impact of specialized processors in elementary particle physics, March 1983, Padova, p. 229. L.O.Hertzberger, Comp. Phys. Comm. 26(1982)79.
2. C.Daum et al., Nucl. Instr. and Meth. 217(1983)361.
3. F.Tuynman, L.O.Hertzberger, to be published.
4. A.Astbury et al., CERN SPSC/78-19 (1978).
5. Reference Manual for the ADA Programming Language, Alsys, ANSI/MIL-STD 1815A, 198.
6. Requirements for ADA Programming Support Environments "STONEMAN", U.S. Department of Defense, February 1980.

Figure captions

fig. 1: Schematic view of a simple 3-processor FAMP configuration

fig. 2: Present FAMP configuration in the UA1 experiment (MTD=Multiple Time Digitizer, FRI=FAMP REMUS Interface, MEM=memory, DPM=Dual Port Memory, FROB=FAMP ROMULUS Output Buffer, TERM=Terminal)

fig. 3: Future FAMP configuration in the UA1 experiment (RM=Reordering Memory, MEM*= Memory with battery back-up, FAREM=FAMP REMUS interface, VFI=VME-FAMP external bus interface, GRAPH=Graphics interface, DISP=Display; see also caption of fig. 2)

fig. 4: Results of a Monte Carlo simulation of the UA1 data-acquisition system (Decision time second level =2 ms, decision time emulator <30 ms, read-out time second central detector buffer =30 ms, double buffering of the data assumed)
curve 1 - no rejection, tape writing speed limiting factor
curve 2 - rejection factor of emulators = 50, no second level reduction, data transport limiting factor
curve 3 - rejection factor of emulators = 10, rejection factor of second level trigger = 5

fig. 5: Schematic representation of FADOS and of some of the information flows:
1 - Commands from debugger
2 - Responses from debug monitor
3 - Stop or kill a process
4 - Debug monitor's confirmation of 3
5 - Start a process (arguments are the name of an executable file, stacksize and priority)
6 - Return process identification of started process
7 - Filenames and data to disk
8 - Data from disk
9 - For a process to be started standard input, output and error files are opened
10- Returned file descriptors

fig. 6: Possible structure of a data-acquisition system (MEM=memory, PROC=processing cell)

fig. 7: Schematic representation of information flows in and tasks of a processing cell

Ⓡ RMS68K is a trademark of Motorola

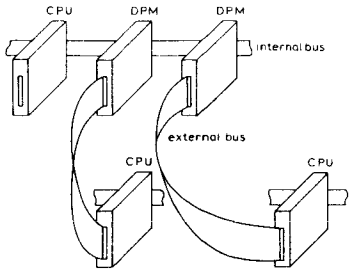


fig. 1

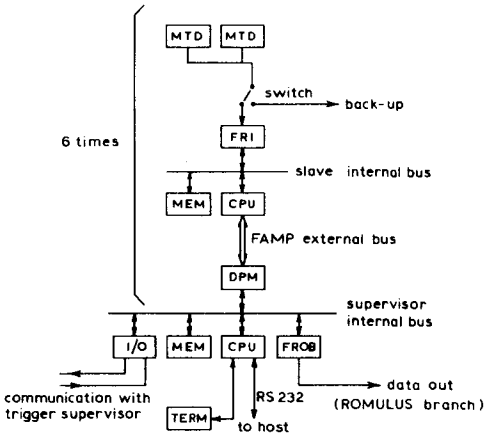


fig. 2

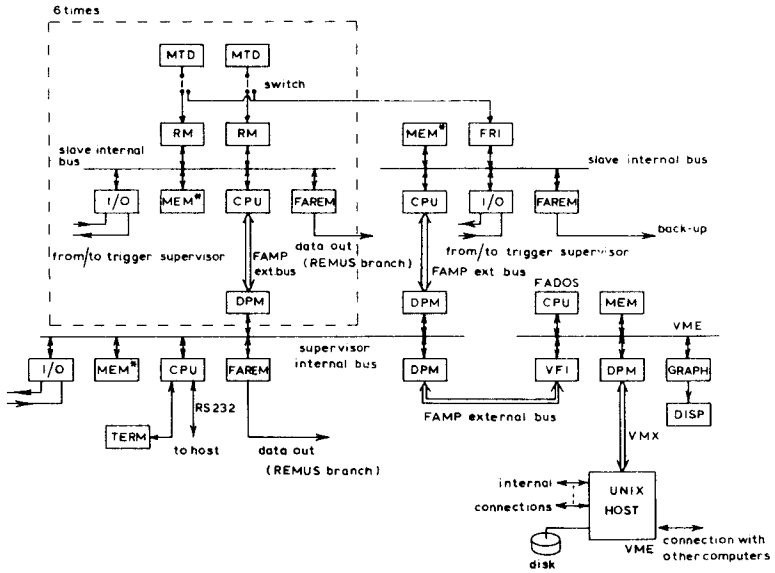


fig. 3

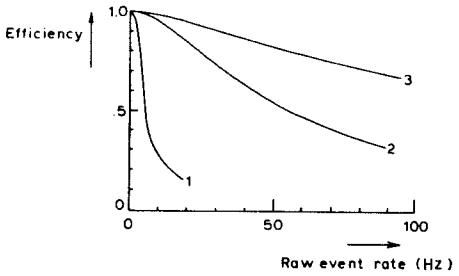


fig. 4

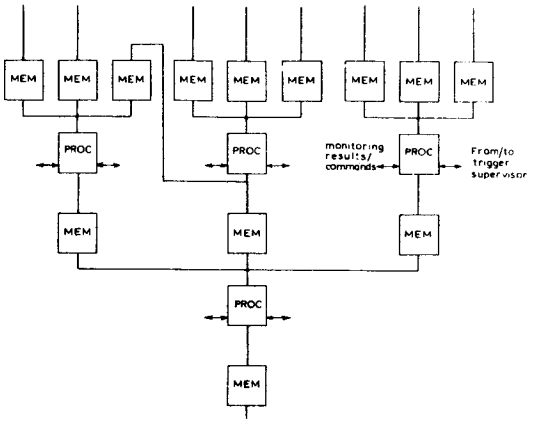


fig. 6

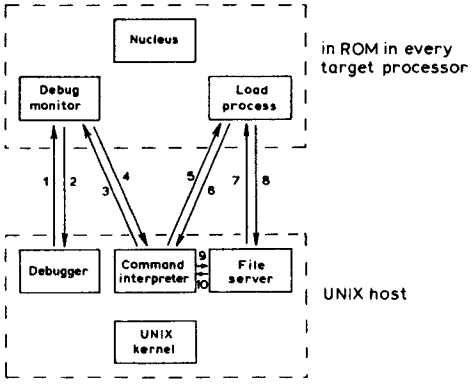


fig. 5

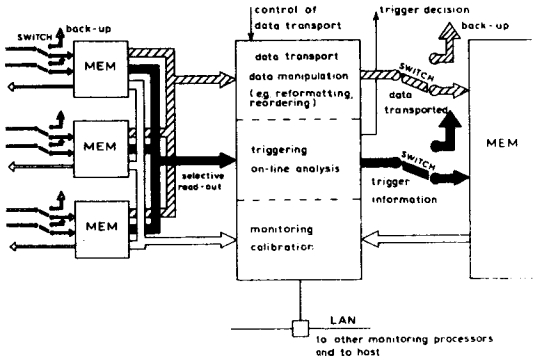


fig. 7

SOFTWARE FOR EVENT ORIENTED PROCESSING ON MULTIPROCESSOR SYSTEMS

M. Fischler,* H. Areti, J. Biel, S. Bracker, G. Case,
I. Gaines, D. Husby, and T. Nash

Advanced Computer Program
Fermi National Accelerator Laboratory
Batavia, Illinois 60510

ABSTRACT

Computing intensive problems that require the processing of numerous essentially independent events are natural customers for large scale multi-microprocessor systems. This paper describes the software required to support users with such problems in a multiprocessor environment. It is based on experience with and development work aimed at processing very large amounts of high energy physics data.

Introduction

We describe here the support and system software that has been developed by the Fermilab Advanced Computer Program (ACP) for users with event oriented problems to be run on ACP multiprocessor computers.[1] Supporting a system of over one hundred individual processors requires a set of efficient, flexible, and simple routines that control the movement of data within the system. To meet these requirements, the routines must be designed with particular types of applications in mind. Specifically, the software described here supports event oriented applications where the problem is naturally divided into a process requiring a single intelligence (such as reading tapes, forming statistics, etc.), and a process done once for each of many hundreds of events, which uses most of the CPU time. In high energy physics event reconstruction, the second process is the actual reconstruction procedure, which does no I/O and which takes at least 99% of the CPU time in most cases.

We are dealing here with a software structure in which a host program feeds events down to a node program, which is replicated on many node processors. The user must provide the information as to how the particular program is to be split into a part to be run on the host and a part for the node. However, the user does not have to explicitly control the detailed logic of data transmission such as deciding which node is finished and should receive the next event. Rather, a set of FORTRAN callable subroutines is provided which are simple to learn to use and which handle the transmission of events to and from the nodes. FORTRAN was chosen because of the strong commitment to FORTRAN in the physics community and the large collection of existing FORTRAN code.

*Presenter

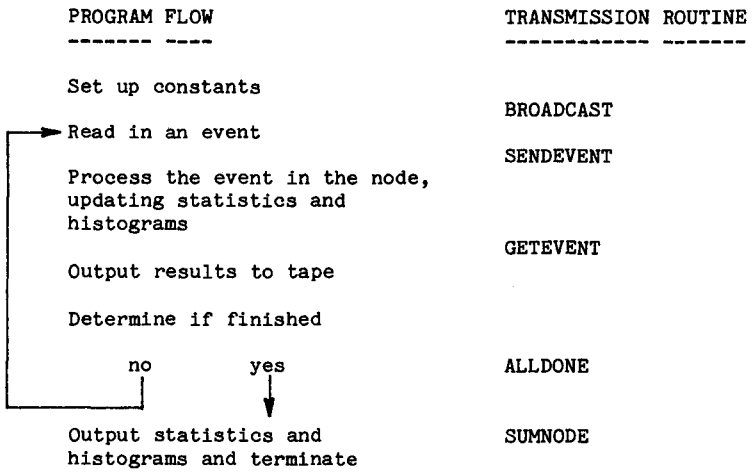
The concepts described here apply to any problem which is separable into many subproblems requiring little intercommunication. These include many lattice gauge theory applications, high dimensional integrals, tracking of particle trajectories through proposed accelerators, as well as problems outside physics such as animation and even (surprisingly) finite element calculations.[2] The system architecture that is natural for such problems is a host computer attached to the I/O devices and to a bus on which many node processors reside.

ACP Support Software

The ACP Support Software allows three modes for transmitting data between host and nodes: Constant Broadcast Mode, in which data, such as calibration constants, calculated by the host is loaded into all the nodes; Event Processing Mode, in which individual event data is loaded by the host into a single node, and the results collected by the host from the nodes as they finish; and Statistics Collection Mode, in which the host retrieves and accumulates results from all the nodes.

A typical program will have the flow shown in Figure 1.

Figure 1



The transmission routines shown in the figure have the following functions:

BROADCAST	Broadcasts constants to all nodes
SENDEVENT	Sends the event to a node
GETEVENT	Retrieves the result from a node
ALLDONE	Will see if all the nodes are finished
SUMNODE	Collects and accumulates statistics from all nodes

In Figure 2, on the following page, is an example that illustrates how a program is modified to take advantage of the routines provided. Note that about a dozen statements have been added to a program which is typically many thousands of steps long.

The user support routines have been gathered into three layers of increasing complexity. This way a user needs to be familiar with only the simplest possible set of routines. A set of Layer 1 routines, including those referred to in Figure 1 will satisfy the needs of many users with programs having basic and simple requirements. Other users may need somewhat more flexibility. Layer 2 routines allow more choices for the programmer (and have, therefore, somewhat longer descriptions to absorb). The Layer 2 routines that correspond to the Layer 1 routines listed above are:

GBROADCAST	Generalized broadcast for inhomogeneous arrays.
SENDBLOCK	Allows multiple block transmissions to nodes of one or more classes.
GETBLOCK	Retrieves multiple blocks of data from nodes of one or more classes.
CHECKNODE	Examines the complete status of nodes.
GSUMNODE	Generalized collection of inhomogeneous data with variable accumulation rules.

For example, if it is necessary to transmit multiple blocks of data for each event, calls to the Layer 2 routine SENDBLOCK replaces the single call to SENDEVENT. Particularly sophisticated users will be able to use Layer 3 routines for direct control of the traffic on the global bus, without having to write their own device drivers or system calls.

Documentation is provided in a complete and extensive "Software User's Guide."[3] A sample of a Layer 1 subroutine description from this guide appears in Figure 3.

Figure 2

Illustration of Modifications Required in User Program Running in Host

```
          ORIGINAL CODE                          MODIFIED CODE
          -------- ----                          -------- ----
1         COMMON/RAW/DAT(20000)                  COMMON/RAW/DAT(20000)
2         COMMON/ANSWER/RESULTS(10000)           COMMON/ANSWER/RESULTS(10000)
3         COMMON/CALIB/C1(100), C2(100)          COMMON/CALIB/C1(100), C2(100)
4         COMMON/STATS/HIST(10000)               COMMON/STATS/HIST(10000)
5         INTEGER DAT, RESULTS                   INTEGER DAT, RESULTS
6                                                LOGICAL LASTEVENT, SEND_DONE,
                                                   ALLDONE, GET_DONE
7                                                INCLUDE '[ACP]ACPUSER.INC'
      C   INPUT, SETUP, CONSTANTS           C     INPUT, SETUP, CONSTANTS
8                                                CALL ACPINIT
9         . . .                                  . . .
10        . . .                                  . . .
11        . . .                                  . . .
                                           C     BROADCAST CONSTANTS TO NODES
12                                               CALL BROADCAST (3, C1, 200, REAL_4)
                                           C     INITIALIZE NODE STATUS VARIABLES
13                                               SEND_DONE = .TRUE.
14                                               GET_DONE = .FALSE.
15                                               LASTEVENT = .FALSE.

      C   EVENT LOOP START                 C     EVENT LOOP START
          .                                      .
          .                                      .
          .                                      .
16      10 CONTINUE                              10 CONTINUE
17                                               IF(.NOT.LASTEVENT.AND.SEND_DONE)THEN
18          CALL READEVENT                         CALL READEVENT
19          IF (ENDOFTAPE) GO TO 20                IF(ENDOFTAPE)LASTEVENT=.TRUE.
20                                               ENDIF

      C   PROCESS EVENT                    C     PROCESS EVENT
21        CALL PROCESS                           IF(.NOT.LASTEVENT)
                                                   CALL SENDEVENT(DAT,20000,SEND_DONE)
22                                               IF(LASTEVENT)THEN
23                                                 IF(ALLDONE(NODE))GO TO 20
24                                               END IF
25                                               CALL GETEVENT(RESULTS,10000,GET_DONE)
26        CALL WRITEEVENT                        IF(GET_DONE)CALL WRITEEVENT

      C   EVENT LOOP END                   C     EVENT LOOP END
27        GO TO 10                               GO TO 10
28      20 CONTINUE                              20 CONTINUE

      C   OUTPUT HISTOGRAMS, ETC.          C     OUTPUT HISTOGRAMS, ETC.
29                                               CALL SUMNODE(4, HIST, 10000, REAL_4)
30        CALL HISTDO                            CALL HISTDO
31        . . .                                  . . .
32        . . .                                  . . .
33        . . .                                  . . .
34        END                                    END
```

SENDEVENT passes a block of data to the first available node and starts that node running. SENDEVENT passes data to block number 1. The data will be passed, unconverted, as 32 bit binary words. (At user option a global parameter can be set at compile time to cause the routine to pass down data as unconverted 16 bit binary words.)

CALL SENDEVENT (ARRAY, LENGTH, SEND_DONE) is equivalent to the following Layer 2 call:

CALL SENDBLOCK(ARRAY,LENGTH,block_number=1,ANY_NODE,ALL_CLASSES,GO).

SENDEVENT (ARRAY, LENGTH, SEND_DONE) Layer 1

Arguments:

	Input only:	ARRAY, LENGTH
	Input/result:	---
	Result only:	SEND_DONE

Return Variables: RETURN_STATUS

ARRAY:

The first word in an array or block of data available to the calling program on which the subroutine will act. This must be the first variable in the common block to be passed to the node.

LENGTH:

In standard usage, this is an integer scalar with the number of 32 bit words of data to be transmitted. Note that a double precision variable is two 32 bit pieces of data, and that a pair of 16 bit integers is a single 32 bit piece. LENGTH is an integer greater than zero, except in GETEVENT and GETBLOCK where LENGTH.LE.0 signifies variable length transmission.

SEND_DONE:

This is a logical variable, returned as .TRUE. if an available finished node was found, and as .FALSE. otherwise. This must be declared a LOGICAL variable in the host program.

The following are return variables available in COMMON/ACPUSER/:

RETURN_STATUS:

An integer variable that is returned to indicate the status of the subroutine's activity. For details see the section entitled, "Reserved Name Parameters and Return Status Variables."

Error Handling

Error handling for a multiprocessor can and should be more sophisticated than on a uniprocessor. Upon detecting an error, the ACP support system provides a description of the error and where it occurred. It also makes available, at user option, a memory dump of the faulting node. A third output is needed in multiprocessing environments which is not needed in uniprocessor computers. Since each node processes a different sequence of events, it is necessary to maintain a history file of which events a processor has done previously. This is made available when an error is detected so that a diagnosis can be made on the development system using utilities that automatically reproduce the errant node's history. The support software on a production multiprocessor only provides information about an error. Analysis of why the error occurred is done on a separate development system (described below) since time on the many-node production system is likely to be at a premium.

The user can specify one of the following levels of action to be taken on detection of an error: ignore the error; print a warning, but continue running; kill the node statistics, but continue running; excise the offending node from the system for the remainder of the run; or abort the run immediately.

Error detection falls in five categories: hardware failures detected by automatic bus, node, and hardware diagnostics; node software errors (divide by zero, etc.); node time out; user defined exceptions; verification exceptions. The last one is available only in a system with multiple nodes, and is a new type of error detection which can be very helpful. The same event is sent to two nodes and the results compared; we call this "verification." Verification enables the system to catch rare software "time bombs," where a logic error in the program running one event causes some area to become invalid, but the invalid area is not used until many events later. Verification will also catch infrequent non-fatal (soft) hardware errors, and enable the studying of soft error rates. This can be inconvenient to do on uniprocessor systems.

The Development System

The development system is available for writing and testing new programs as well as for analysis of errors detected in the production environment. This system will consist of a host and a few samples of each type of node that exist on production systems. The host, a commercial super-mini (VAX or similar), has compilers, a symbolic interactive debugger, file handling, editing, and all the other features of such a computer. The nodes have a node compiler and a node symbolic debugger, when available; otherwise, a cross-compiler for the node is supported on the host. (The production nodes have only a stripped-down "operating system" called the Tight Loop Monitor, which waits for the host to tell them an event is ready to be run and jumps to a program which had been downloaded over the global bus.)

Additional software is provided on the development system to support error analysis. This includes a convenient way to work through the node memory dump available when errors occur, a facility for reconstructing a

particular sequence of events from the history file to duplicate the conditions under which errors had occurred, and support for I/O directly from the node.

There is also a set of quick and simple automatic procedures for users to compile and link the node programs forming node executable images and download node programs under the control of the host FORTRAN program. These are available for use in both the development and production systems. They allow the user to select options both at compile and run time concerning how the system is to behave when handling errors, passing data to nodes, etc.

In conclusion, Fermilab's ACP has developed, in addition to the hardware, user friendly software for using its multiprocessor systems. The same software concepts can be employed over a broad range of problems, and with various implementations of the hardware architecture.

References

1. I. Gaines, et al. "Fermilab's Advanced Computer Program," Proceedings, this conference, FERMILAB-Conf-84/63. Thomas Nash, et al. "Fermilab's Advanced Computer Reseach and Development Program," Proceedings, Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics," Padova, 1983, p. 227. Hari Areti, et al. "ACP Modular Processing System: Design Specifications," Rev. April 2, 1984, FN-402. See also Reference 3.

2. John A. Swanson, talk at Forefronts of Computing Conference, NBS, Gaithersberg, Maryland, June 25-27, 1984.

3. Advanced Computer Program, "ACP Software User's Guide for Event Oriented Processing," Rev. June 18, 1984, FN-403.

QUESTIONS AND ANSWERS

Q: Have you considered separating your arrays in sections and assigning these sections to different users?

R. Poutissou

A: Yes, the nodes have a class number associated with each item and this can be specified in the level 2 subroutines.

Comment: This is possible in the development system but the production systems are intended to be single-user systems.

T. Nash

The Fermilab ACP Multi-Microprocessor Project

I. Gaines,* H. Areti, J. Biel, S. Bracker, G. Case,
M. Fischler, D. Husby, T. Nash

Advanced Computer Program
Fermi National Accelerator Laboratory
Batavia, Illinois 60510

ABSTRACT

We report on the status of the Fermilab Advanced Computer Program's project to provide more cost-effective computing engines for the high energy physics community. The project will exploit the cheap, but powerful, commercial microprocessors now available by constructing modular multi-microprocessor systems. A working test bed system as well as plans for the next stages of the project are described.

Introduction

High energy physics experiments have become more and more complex and are accumulating ever increasing amounts of data. The need for computing to analyze these experiments has expanded enormously. Elsewhere in high energy physics computing problems, such as beam-orbit simulations for the design of the SSC and lattice-guage theory calculations, are also expected to require large amounts of computing time. We can no longer afford enough conventional computers for the overall high energy physics workload. Many experiments have already had their ability to do physics compromised by limitations in the amount of off-line computing power made available to them. With the turn-on of a number of even more complex colliding beam detectors in the immediate future, the problem has become so acute that it has spawned several high level review committees.

In response to this problem, Fermilab has established the Advanced Computer Program (ACP) [1] with the primary mission of developing new approaches to computing that will represent more cost-effective alternatives to conventional mainframes for the compute-bound problems of high energy physics. The ACP's first project is the development of a flexible and modular approach to multiprocessing based on 32 bit microprocessors of near VAX class power. We describe this project, its goals, plans, and status, in the following.

Design Goals and Concepts

One method of providing more cost-effective computing is to design dedicated special purpose processors for particular problems. In the high energy physics community such devices are in common use as trigger processors.[2] Such devices have almost no limit to the increase in cost-effectiveness that can be provided, but suffer from the disadvantage of being relatively inflexible and difficult to program. Changing to a different algorithm requires a large amount of work by system experts.

*Presenter

On the other hand, commercial computer manufacturers and university computer scientists usually focus on designs of fully general parallel processing systems, where large numbers of processors can all be brought to bear on an arbitrarily general problem. Such fully general systems must solve the difficult problems of shared memory, interconnection networks, and synchronization mechanisms. The complexity inherent in the goal of generality implies a long delay in bringing the designs to practical fruition. Furthermore, much of the cost of such systems goes into pieces other than the processing elements themselves, reducing the potential cost-effectiveness.

The ACP project is neither fully general nor dedicated special purpose. Rather, it is attempting to exploit the characteristics of the relatively well understood high energy physics computing problems to design a simple and straightforward architecture that gives near maximal cost-effectiveness for these problems while maintaining the flexibility and programability of general purpose computers. In particular, the most important feature of the high energy experiment computing problems is their event oriented nature. A typical experiment may have tens or hundreds of millions of events, each of which is an essentially independent analysis problem. The natural and trivial parallelism inherent in the problem leads to a multiprocessor solution with no global memory and simple interconnections, but where each processor has sufficient local memory to process a complete single event.

The ACP project will exploit additional characteristics of the problem to yield improved cost-effectiveness. These include the existence of compute-bound kernels (inner loops in the programs which use very large fractions of the overall CPU time), structured blocking in the programs with minimal communication between the blocks, and very long (weeks to months) run times for the same program on different data tapes. This makes it sensible to design special purpose "hardware subroutine" coprocessors for efficient execution of the inner loops of particular types of problems. It is also appropriate to allow for reconfiguring the connection topology and the distribution of memory and special coprocessors for the needs of a particular program with a long production run.

The critical goal of very high cost-effectiveness for the ACP system, therefore, is met by the following features of the design: an extremely simple architecture; small, mass-produced VLSI (and thus cheap) CPUs; and (eventually) from high-speed special purpose hardware attached to the CPUs for particular problems. Another important design goal is modularity, which allows the system to be optimally reconfigured for a given problem and allows the use of newer and faster CPUs and other components without redesigning the entire system. For this, it is important to construct the system out of commercially available VLSI and board level components whenever possible. This reduces initial design effort, can reduce costs and will make it easier to make copies of the system with minimal expert assistance. A third important goal is user friendliness, which is realized by supporting FORTRAN-77 on processing CPUs and making available program development and debugging tools on a convenient host machine.

The ACP system can be summarized in a long-winded phrase, as a flexible, loosely-coupled multi-microprocessor system, with optional customized special purpose hardware subroutine coprocessors. It is broadly applicable to a large class of compute-bound problems which share the important characteristic of being "event-oriented," that is, having a natural simple parallelism inherent in

the problem. These include a number ouside of high energy physics such as process simulation, robotics, animation, and finite element analysis.

System Overview and Phasing [3]

The core of the ACP system is the individual processing node (shown with some optional additions in Figure 1). The node always consists of a processor which supports user software written in a high-level language (FORTRAN-77) and sufficient memory to contain an entire event (at least 1 Mbyte). Optionally, as required for particular problems, the node may also contain additional memory up to 16 Mbytes, floating point hardware coprocessors, special purpose coprocessors optimized for the compute-bound kernel running on that node, and nearest neighbor node communication interfaces for problems requiring fast grid-like internode communication.

Each node lives within a dual bus structure. It is a slave on a global bus over which programs and data are downloaded to all the nodes. The node's CPU accesses its own local memory as a master over a private local bus. Thus, each node can address its own memory simultaneously without any contention on the global bus. High-speed hardware coprocessors may even require a third super-fast bus to process data in memory with a much faster cycle time than that of the local bus.

The software within such a node is simple because the node is a slave on the global bus. The node waits for events to be delivered to it and processes them on command. The primitive "operating system" which runs on the individual nodes must only support the FORTRAN run time environment (but not I/O), trap exceptions, and handle communication with a host CPU through dedicated memory locations. This node software system jumps to the user code when a flag is set indicating the presence of an event. It sets a second flag indicating completion when the user code returns. Further details on ACP work on support software are found in the companion paper, "User Software for Event-Oriented Processing" by M. Fischler et al. [4] .

Arrays of such nodes can be configured in a variety of topologies, depending on the problem at hand. These range from the most simple (Figure 2) where a collection of identical processors are lined up each to receive individual events, to the more complicated arrangement of multiple ranks of processors shown in Figure 3. Other arrangements, suitable for accelerator beam-orbit simulations, are discussed in Reference 5.

We require a CPU node to have the processing power for reconstruction codes of at least 0.5 VAX 11/780 (or else too many nodes are required), and to run high level language programs (specifically, FORTRAN-77). It should use cheap memory technology (high production MOS dynamic RAMs) so that comfortable amounts of memory can be made available in each node. Upward compatability to higher performance parts without major system redesign is also required as is easy coprocessor interface. All of these considerations point clearly to the use of commercial microprocessors for the CPU nodes, provided they can meet the performance goals.

Fortunately, at least six different vendors (AT & T, DEC, Intel, Motorola, National Semiconductor, and Zilog) have announced 32 bit microprocessors with expected performance well above the ACP goals. Three vendors (AT & T, Motorola, and National) already have working 32 bit chips. The ACP group has benchmarked

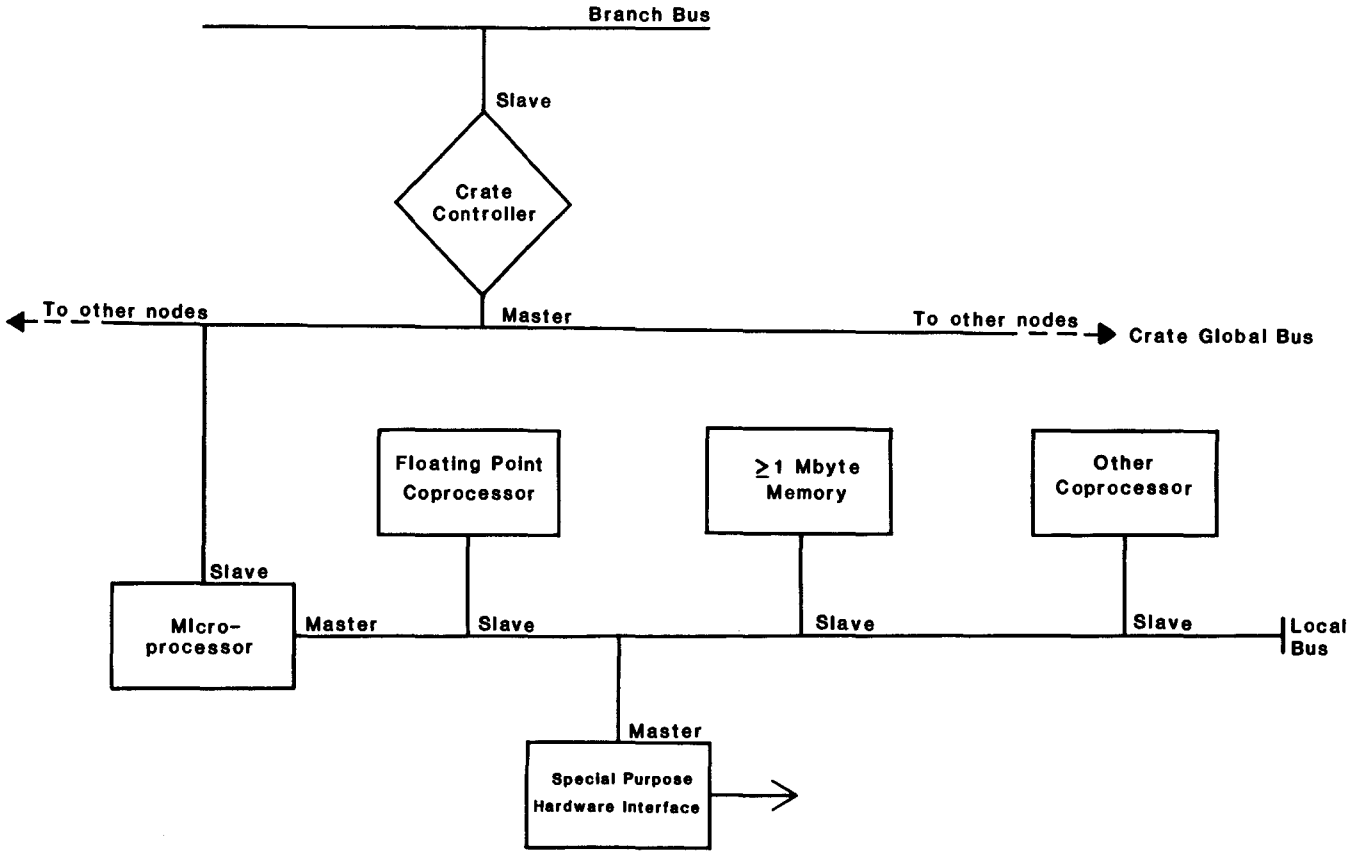


Figure 1. A single processing node.

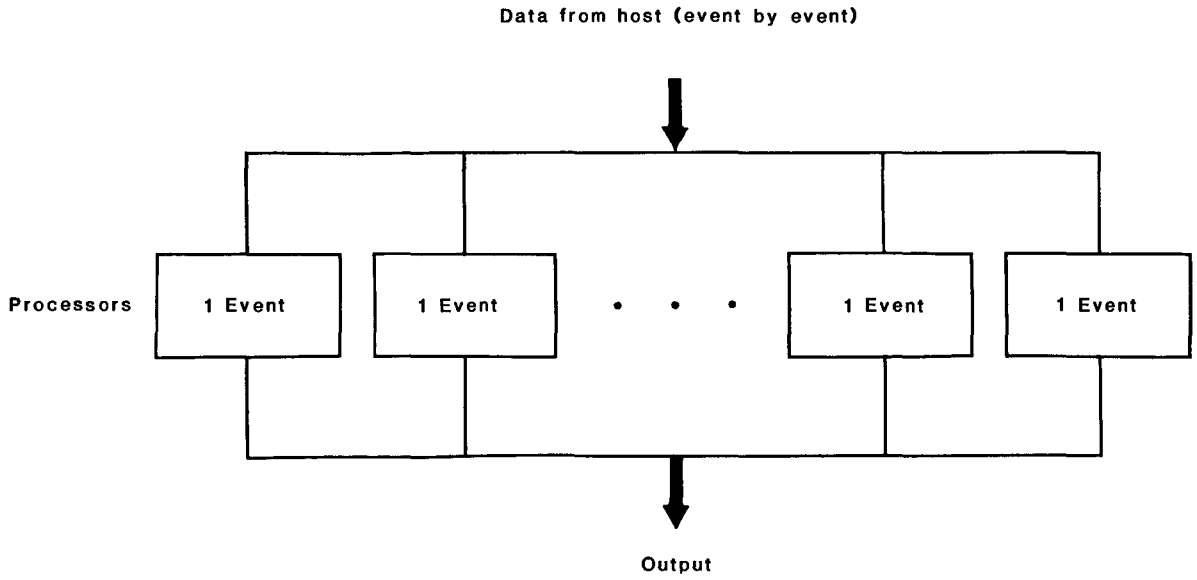


Figure 2. Single rank multiprocessor concept.

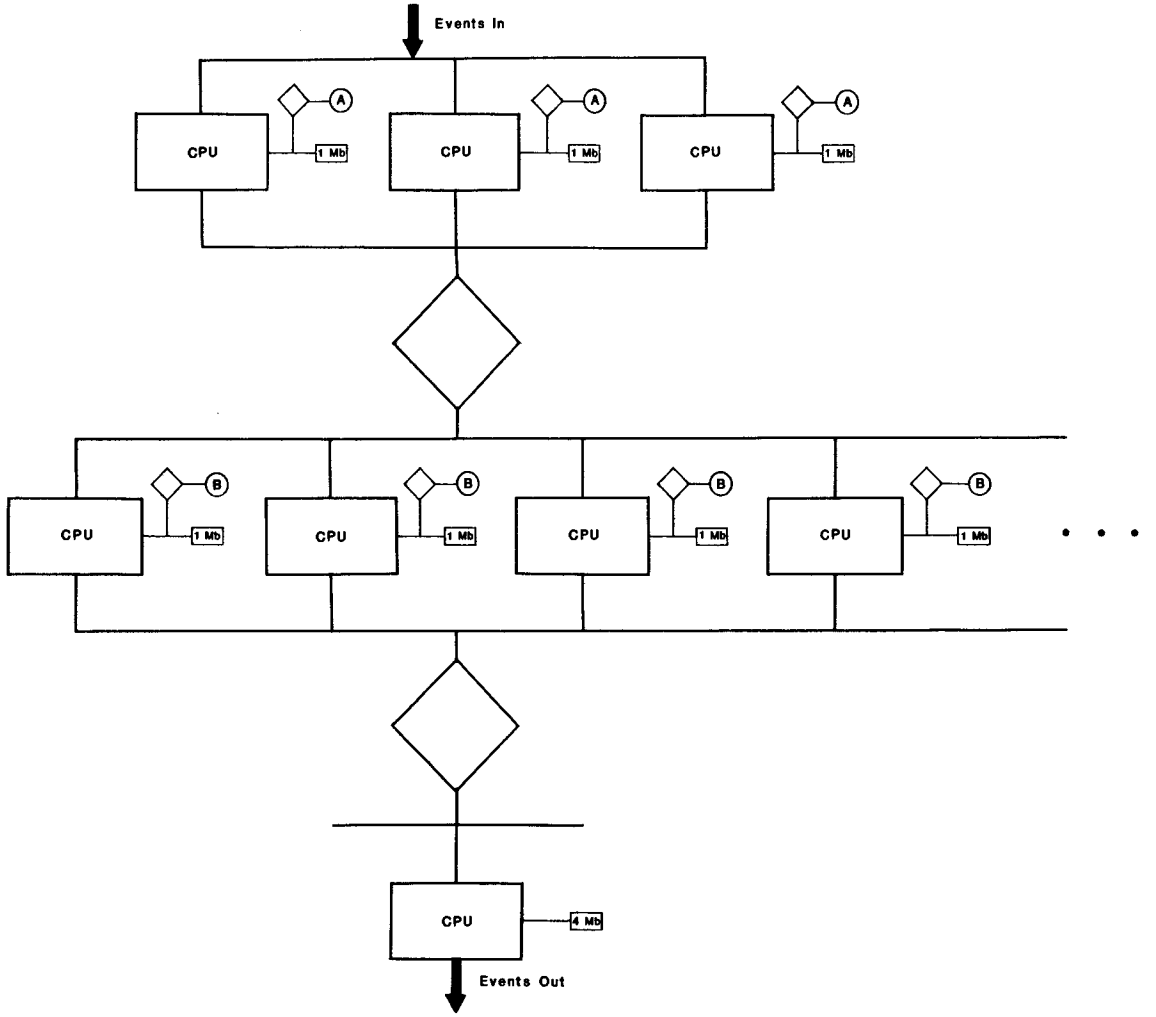


Figure 3. Multiple ranks of processors; different coprocessors and memory capacity in nodes of different ranks.

a full-scale physics track reconstruction program written in FORTRAN on existing 16 bit processors. Performance was measured relative to a VAX 11/780 as 0.1 for the Motorola 68000, and 0.12 for an Intel 80286. Taking into account the performance improvements available in the 32 bit versions of these chips, it is clear that the 32 bit commercial processors to be widely available in early 1985 will easily meet the ACP performance goals. As of this writing, we are in the process of benchmarking the new 32 bit Motorola 68020.

The project is proceeding in three phases, each of which is described more fully in subsequent sections. Phase I, now all but complete, consists of the development of support and error handling software on a test bed system using 16 bit processors and a 16 bit bus. Phase II, scheduled to be complete in summer 1985, is the first full-scale production system, consisting of at least 128 full-performance nodes. Phase III includes the development of special purpose hardware coprocessors, more complex node interconnection and host fuction schemes, and higher performance nodes.

System Components

The components of the ACP system in each of its phases consist of the following items:

1. Crate/bus - The crate must support at least two busses, a crate wide global bus and a separate segmented local bus for each CPU node in the crate. After Phase I, the global bus should support transfers at a rate of 20 Mbyte/sec with an address space sufficient for 16 Mbytes of memory for each node in the crate. The local bus should support memory accesses at a speed sufficient for the processors which will run with no wait states, and should have 16 Mbytes of address space. Only the crate controller needs to be a master on the global bus, while the individual node CPUs are each masters on their own local busses. The local bus should be reconfigurable to allow for different numbers of cards in each node at different times. Optional desirable features are a serial bus for low-speed or diagnostic transfers, and provision for a high speed coprocessor bus. Both MULTIBUS II from Intel and VME/VMX from Motorola are commercial busses that meet these requirements.

2. CPU board - The CPU board should be a commercial 32 bit microprocessor that is a master on its own local bus and can be controlled from the global bus. It must run FORTRAN-77 programs. The initial Phase II system will contain at least two different types of CPUs. It is expected that commercially produced boards will be available at competitive prices. The ACP is a "beta site" for a 68020 board under development by Motorola's Microsystem Division.

3. Memory board - The memory board needs to be dual-ported on the global and local busses, although the arbitration between the ports can be very simple (the global bus can be given absolute priority). It is expected that commercially produced boards will become available.

4. Crate controller - Used as the only master on the crate global bus, the crate controller must be able to do full-speed (about 20 Mbytes/sec) reads and writes to anywhere in the crate memory space. It is a slave to the host on a bus linking crates.

5. Host interface - This must provide data and control paths to allow the host CPU to download programs and event data to crates full of a total of up to 255 nodes. After Phase I, this system must link the host to up to 64 crate controllers. It may include the intelligence to find nodes available for new events and detect nodes with completed events.

6. Development host - A minicomputer supporting multiple users must be provided as a development host. Small numbers of nodes of each variety will be attached to this computer to allow the user to develop and debug programs for use in a multiprocessor environment. The system, most likely a VAX 11/780 running VMS, includes file editors, compilers, symbolic debuggers, etc. for both host and node user software.

7. Production host - Linked to the development host via a network (DECNET), the production host is a single user system supporting running programs on the multi-node system. It provides the user the functions of event input/output and control of the nodes in a transparent manner. The host portion of user programs, as well as system control functions, run in the production host. It is often referred to as the "roots" of the tree-like ACP multiprocessor system (see following discussion).

8. System software - Software components include: development tools (compilers and debuggers); user support subroutines to allow programs to be split into a host piece (which does event I/O and printout) and a node piece (which executes the CPU intensive portion of a user's code simultaneously on many nodes); diagnostic and verification tools; and simulators of the overall system. The system software runs on the development host and various components of the production host as well as on the nodes. This is more fully described in References 4 and 6.

Test Bed System

The Phase I test bed system, now in operation, was built to develop and test the user support multiprocessor software described in References 4 and 6. Since high performance was not required, it consists of low-speed 16 bit hardware. It includes a full software prototype with node "operating systems", user support subroutines, and command procedures for compiling and debugging. Error handling and verification capabilities are presently being developed.

The test bed hardware contains 6 CPU nodes: 5 Motorola 68000s and one Intel 8086 with an 8087 floating point coprocessor. The 8086 has 256 Kbytes of onboard RAM, while each 68000 has a 512 Kbyte memory on a separate card. The system is implemented in a MULTIBUS I crate, with MULTIBUS being used for the global bus. A commercial SAM bus manufactured by SGS Corporation (Milan and Phoenix) is used as the local bus for the 68000s. The 8086 has no local bus since all memory is on-board. The 68000 boards were designed and built by the ACP group, while the 8086 board and the memory boards (dual ported MULTIBUS and SAM bus) are commercial products, as is the crate. All five memory boards can be put on the local bus of a single 68000 to test programs requiring up to 2.5 Mbytes of memory. A VAX 11/780 is being used as the host for the test bed system, with a DR11W UNIBUS DMA interface connecting the host to the crate. An ACP built board interfaces the DR11W to the MULTIBUS and acts as the crate controller.

Both types of processors are supported with FORTRAN-77 compilers and run-time libraries. The 8086 compiler, from Intel, is a cross compiler which runs on the VAX. The 68000 compiler, from Absoft Inc. (Royal Oak, Michigan), is a native mode compiler which runs on one 68000 node. The 68000 software also includes a powerful interactive symbolic debugger which can be used to debug programs running on the nodes.

ACP software on the test bed system is the full complement of routines and utilities described in References 4 and 6. This includes the operating systems on the individual nodes, user subroutines to allow the user to split his program into a host and a node piece, automatic routines to download the user's code into the nodes and handle all host-node communication, command procedures on the VAX to compile and link the users programs for execution on the nodes, and VAX routines to support the 68000 compiler and the run-time system. Several different large high energy physics FORTRAN programs have run successfully on the test bed system. Test users are finding the support software convenient to use. A major reconstruction package was successfully brought up by two physicists with no prior knowledge of the ACP system in a little over two working days.

The performance of the system is limited because of the small number of nodes and the fact that the nodes are low-speed 16 bit processors. However, two important aspects of the test bed system performance that can be investigated are the efficiency of utilization of the nodes and the possibility of bus contention on the global bus. The first issue was checked with a typical reconstruction code by evaluating the fraction of time the individual nodes spend executing user programs compared to the time they spend waiting for events from the host. In all cases this was greater than 90%, and could be made to approach 100% by having the user software double buffer events. The second issue was checked by comparing the performance of the system with all six nodes running to the performance with a single node running. Six times the performance of an individual node was obtained. Similar tests will be carried out on the Phase II system in 1985.

Full-scale Production Systems

The first full-scale production system is scheduled to be operational in summer, 1985. It will consist of at least 128 nodes using full-speed 32 bit microprocessors of at least two types (Motorola 68020 and DEC MicroVAX are the leading candidates at the moment). At the crate level it will use the high-speed 32 bit bus most appropriate for the processor in use in that crate. Clearly, VME/VMX is appropriate for the 68020. Either MULTIBUS II or VME/VMX may be suitable for other processors.

The crates, CPU nodes, and memories in this Phase II system are simply higher speed versions of the existing Phase I components in the test bed system. However, the crate interconnections are necessarily more complex to allow the use of a larger number of nodes. A tree-like system (see Figure 4) will be designed for the Phase II system. The host CPU functions, including I/O and system control, are in the root. The node crates are connected by simple, ACP designed, high-speed branch busses. These multiple branch busses, capable of operating simultaneously at 20 Mbytes/sec each, are interconnected via a bus switch which allows any one of several root masters to be connected to any one of the branches. This will support the highest performance requirements of future data storage devices and on-line high level trigger applications with

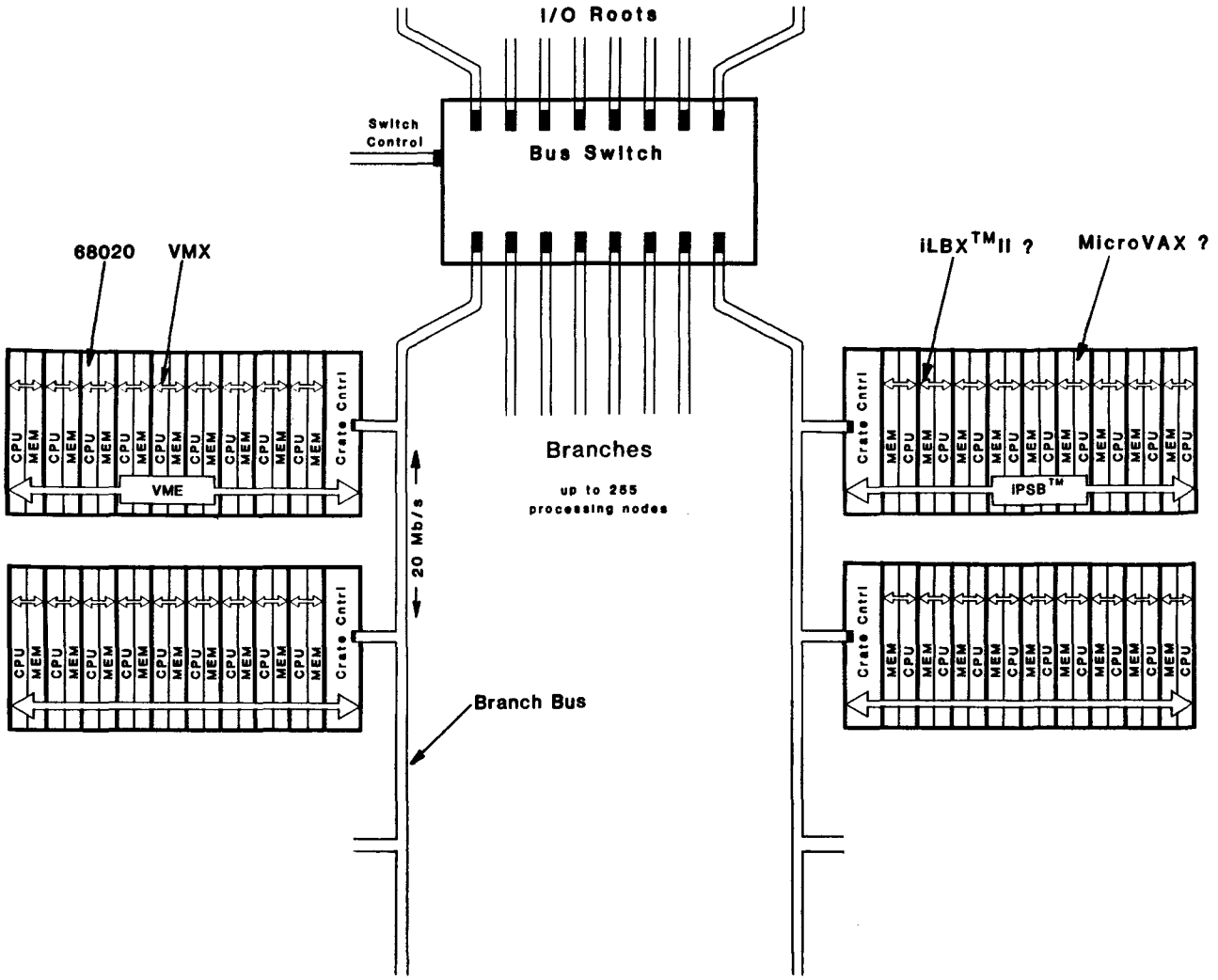


Figure 4. Branches and bus switch of Phase II system.

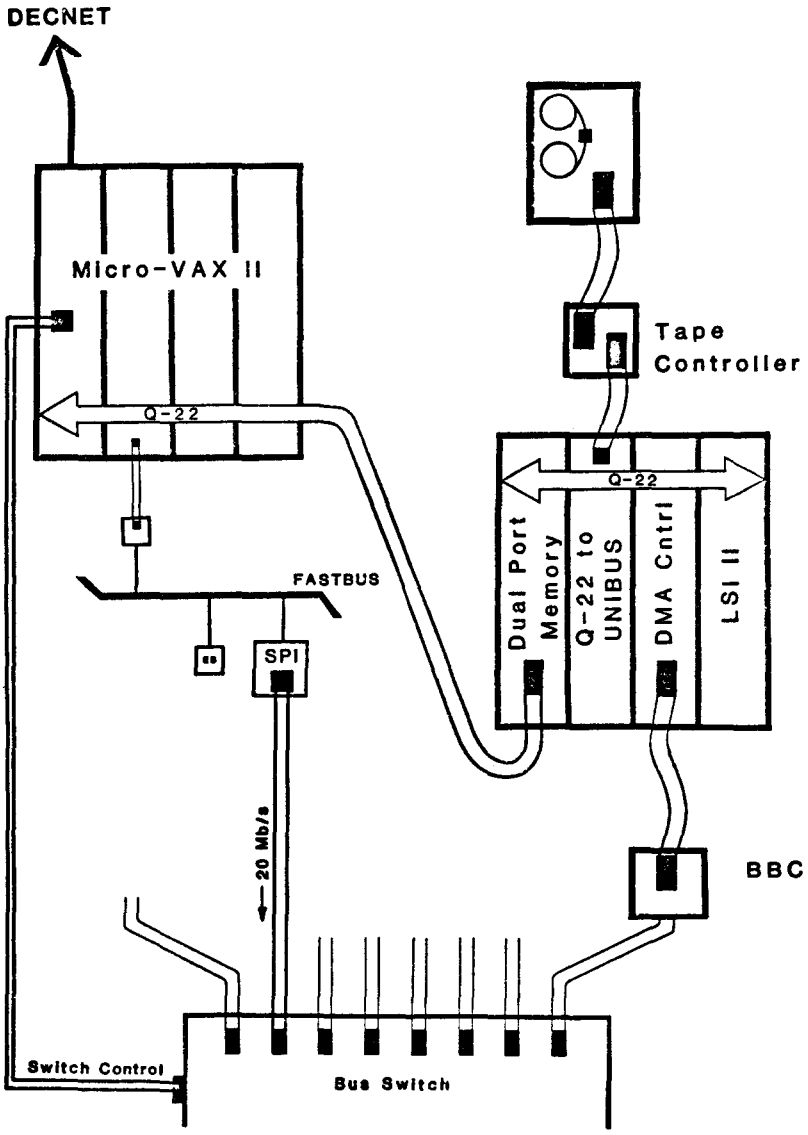


Figure 5. Example of root arrangements for ACP multiprocessors.

over 100 Mbytes/sec data bandwidth capability. The only overhead for this powerful multiple branch system is the relatively low cost of one bus switch, presently estimated at less than $5000. For the simplest implementations, with a limited number of nodes, a single branch bus can be used to connect all crates of nodes with a simple host interface as in the Phase I system.

As soon as the more pressing design issues at the node and branch level will allow, the single host CPU of the Phase I design will be replaced by a sophisticated root with a group of individual CPUs each performing a separate function (see Figure 5). LSI 11s (or similar devices) sitting on a Q-22 bus will act as input and output controllers for tape (or disk) operations. Each is connected through a UNIBUS converter to one or more tape drives and disks. They will operate under the familiar RT11 system to pass data between tape and the nodes through a Q-bus DMA I/O device and a branch bus controller (BBC), the master on the branch bus.

The user's production host software will run on a separate CPU, most likely a MicroVAX running MicroVMS. This CPU sits on a second (global) Q-bus. A memory with two Q-bus ports services its local LSI 11 and the MicroVAX. This allows the user high level software in the MicroVAX to initiate execution of the I/O and node communication primitives in the LSI 11. System control software monitors the status of the nodes, sets the bus switch, and transmits the node address cycle before each block of data cycles. This software also resides in the MicroVAX which is connected to the switch control port.

Also shown in Figure 5 is a root connected to a FASTBUS on-line data acquisition system through a special processor interface module (SPI), which is a master on the branch bus. In this environment, the host MicroVAX is informed by FASTBUS of a ready event and its type. The MicroVAX, under control of user software, sets the switch and transmits the node address just as it does when operating with a tape drive as described above. It then instructs the FASTBUS system to transmit the event over the appropriate root channel. The bus switch can support up to eight such root channels operating concurrently, each carrying up to 20 Mbytes/sec. This can include one or more FASTBUS channels, along with tape or disk I/O channels. This flexible and modular root system provides a cost-effective implementation of host CPU functions for off-line systems, as well as a convenient way to use the same collection of nodes with unchanged user software in both on-line and off-line environments.

In some sense, this has been a description of a Phase 2.1 system since, as already alluded to, the ACP may not have the design resources to develop the components of the root which are not commercially available on the time scale planned for Phase II. Early testing of the first full scale system may take place using a single rather than double Q-bus system, or even a VAX 11/780 as the production host much as has been done for the test bed system. However, the latter configuration would only take advantage of about half of the full data rate capabilities of 6250 bpi tape drives. For this reason, because of the large cost savings, and because of its importance in on-line activities a multiple micro-CPU root will be brought on-line as early as possible.

Conclusion and Future Directions

Phase III, starting in the second half of 1985, will build on the modules developed in Phase II to provide higher performance and more specialized versions of the ACP hardware. This will include implementing the production

host with more cost-effective processors, the incorporation of higher performance nodes, and the development of special purpose hardware coprocessors for a variety of particular algorithms. The flexible bus switch, and a nearest neighbor connection module which may be developed in Phase III, will be exploited to provide more complex node interconnection schemes in both grid-like and multiple rank systems.

A large amount of industry effort, including both minicomputer and semiconductor manufacturers, is converging in the direction of making VAX class VLSI products available at the chip and board level. The ACP is developing the hardware and software structure to take early advantage of this most cost-effective and flexible solution to high energy physics production computing needs. It has demonstrated user support software that makes it relatively comfortable for physicists to take advantage of multiprocessing. In the course of these activities, the ACP is testing multiprocessor architectures and solving system problems, both in hardware and software, that are relevant to many computer research activities outside of high energy physics.

References

1. Thomas Nash, et al. "Fermilab's Advanced Computer Research and Development Program," Proceedings, Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, 1983, p.227.

2. See other papers in the Proceedings of this conference as well as Proceedings of Topical Conference on the Application of Microprocessors to High Energy Physics Experiments, CERN, Geneva, Switzerland, May 4-6, 1981 (CERN 81-07), and Proceedings of Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.

3. Hari Areti, et al. "ACP Modular Processing System: Design Specifications," Rev. April 2, 1984, FN-402.

4. Mark Fischler, et al. "Software for Event Oriented Processing on Multiprocessor Systems," Proceedings, this conference, FERMILAB-Conf-84/64.

5. Mark Fischler and Thomas Nash, "Computing Tools for Accelerator Design Calculations," Report of DPP Workshop, Accelerator Physics Issues for a Superconducting Super Collider, Ann Arbor, December 12-17, 1983, UM HE 84-1, page 113.

6. Advanced Computer Program, "ACP Software User's Guide for Event Oriented Processing," Rev. June 18, 1984, FN-403.

THE 3081/E PROCESSOR*

P. F. KUNZ,† M. GRAVINA, G. OXOBY, P. RANKIN, AND Q. TRANG
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

and

P. M. FERRAN, A. FUCCI, R. HINTON, D. JACOBS,
B. MARTIN, H. MASUCH, AND K.M. STORR
CERN
1211 Geneva 23, Switzerland

1. Introduction

Since the introduction of the 168/E,[1] emulating processors have been used over a wide range of applications[2] including both offline event reconstruction and Monte Carlo applications, and online triggering and filtering.

This paper will describe a second generation processor, the 3081/E. This new processor not only has much more memory space, incorporates many more IBM instructions, and has full double precision floating point arithmetic, but it also has faster execution times and is much simpler to build, debug, and maintain.

Nonetheless, with the 168/E, valuable experience has been gained on how to make efficient use of this kind of processor which, unlike computers or commercial microprocessors, does not run an operating system nor have a direct connection to I/O devices. The 3081/E takes advantage of this experience by maintaining the same style of flexible but simple interface as the 168/E. This paper will also describe how such processors have been and will be used.

---

* Work supported by the Department of Energy, contract DE-AC03-76SF00515

† Presenter

2. The Processor

The architecture of the 3081/E is shown in Fig. 1. The details of the processor have been given elsewhere,[3] so only a brief summary will be given here. The processor has a modular structure. There are four execution units interfaced to two 64 bit wide busses, called the ABUS and the BBUS. The busses each carry 8 bytes of data and 1 parity bit per byte. Also interfaced to these busses are the control and register unit, data memory, and the interface. The control and register unit serves four functions: it contains the microprogram address counter, conditional branching logic, the data memory address logic, and the register files. A microinstruction can transfer two operands simultaneously on the ABUS and BBUS busses from data memory and/or registers to an execution unit. The results from an execution unit are transferred on the BBUS to a register, to memory, or along with a new operand on the ABUS to another execution unit. Instructions are fetched on a third, 32 bit wide bus, the PMD bus (not shown in Fig. 1). There is a single clock which has a cycle time of 120 nsec.

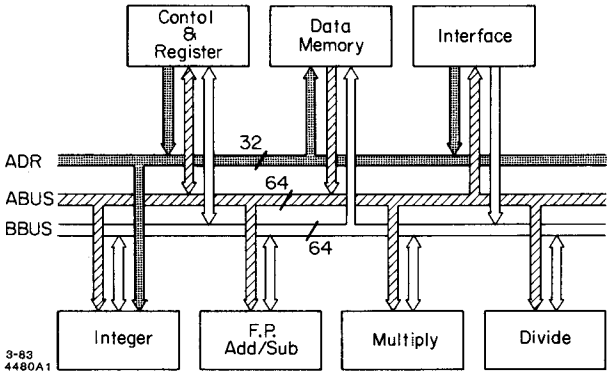


Fig. 1. Block diagram of 3081/E

An important goal of the 3081/E processor project is to produce a processor that is simple, reliable, and easy to debug and maintain. The choice of the modular architecture helped tremendously to reach these goals. The design of the processor is much simpler than the 168/E. The design is much more conservative and uses off-the-shelf multiple source TTL components. Every effort was made to reduce the man-power cost to build, debug, and maintain the processor. FORTRAN simulations have been done of each execution unit which have made a valuable contribution to the designing and in debugging. For example, the Add/Subtraction execution unit with over 200 MSI circuits, had only one design error when it was debugged, and this error was just one signal that had the opposite polarity in the hardware due to

an error in the simulation. The cost of the processor, power supply, and chassis is expected to be under US$ 10,000 excluding the cost of memory.

2.1 MEMORY

Memory is one of the most important aspects of any computer or processor. In the high energy physics field, both the size of analysis programs and the quantity of data per event have grown so that the memory space needed is measured in units of Megabytes.

The memory of the 3081/E is implemented using the less dense but faster static memory circuits. Today they have 55 nsec maximum access and cycle time, come in packages of 16 Kbits, and cost about US$ 5,000 per Megabyte. The speed of memory is important because even with the best of compilers, a processor still obtains one operand (of the two for an arithmetic instruction) from memory over 75% of the time. Thus the speed of a processor tends to be dominated by its memory access time. The fast memory and 64 bit data path to it is also the best solution for online applications which must support FASTBUS I/O rates.

A 3081/E memory board at present contains one half Megabyte of either program or data memory with byte parity. The processor can accept a maximum of fourteen memory boards or 7 Megabytes. It is expected that 64K static memory circuits will be introduced in 1984 so by 1985 they will be reasonably priced. Their use will lower the cost of the processor's memory and make it possible to have a processor with 28 Megabytes.

2.2 EXECUTION UNITS

For high energy physics code, good floating point performance is essential, especially due to the heavy use of trigonometric functions in most analysis codes for solenoidal detectors. Attempts to use commercially available microprocessors with their floating point co-processors have led to disappointingly poor performance.

The following sections give a short description of each of the execution units.

Floating point add/subtract

A REAL*4 or REAL*8 add/subtract is done in 360 nsec, including reading one operand from memory. The floating point compare instruction needs only to generate the condition codes and not a result, thus it is one cycle shorter. This execution unit is also able to do an integer to floating point conversion in 360 nsecs.

Multiply

The implementation of the multiply execution unit has been optimized for single precision execution time. INTEGER*4 and REAL*4 multiplies take 360 nsec including reading one operand from memory. Modern, multiple-sourced (thus cost competitive) 16 by 16 multiplier circuits are used. To implement double precision multiplication in the same way would take a considerable number of circuits, therefore, an iterative technique is used that is reasonably fast. The results of a REAL*8

multiply is available after only 4 internal cycles for an overall time fo 720 nsecs including reading one operand from memory.

Divide

The divide execution unit does division iteratively, 2 bits per cycle which leads to a INTEGER*4 divide in about 2 μsecs and a REAL*4 divide in about 1.5 μsecs.

Integer

All integer instructions except multiplication and division are done in the integer execution unit. This unit handles the four–byte (INTEGER*4) and two–byte (INTEGER*2) arithmetic operations, and also the instructions with one–byte operands (LOGICAL*1 and CHARACTER*n). This is especially important for implementation of the instructions required by the FORTRAN '77 compilers. Both single word (32 bit) and double word (64 bit) shifts by any number of places are done in one cycle. Shift instructions are important for online trigger applications, when packed binary information needs to be expanded to individual words.

Optional units

It is possible to add other execution units to the 3081/E busses. For example, one could add a matrix multiplier/accumulator for lattice gauge theory calculations, PROM based look up tables, or other specialized 'hardware subroutines'. For the moment such devices are beyond the scope of the 3081/E project. They are also less necessary as the processor is already inherently very fast. It will also be possible to upgrade any of the existing execution units, when sufficient technology advances warrant the change, thus achieving higher performance and/or lower cost.

2.3 Instruction Pipelining

The separation of execution units, each capable of operating on its operands internally, allows for instruction pipelining. First there is pipelining of memory address calculation on the control and register board. Secondly, the Add/Subtract and Multiply execution units are capable of pipelining internally. That is, they can accept a new operand pair every cycle, then output the results in the next two cycles. Thirdly, one cycle can send an operand pair to say the add/subtract unit, and the next cycle can send an operand pair to the multiply. Fourthly, in the same cycle an execution unit can output results and another execution unit, or memory, can accept the results, thus overlapping input and output cycles. In addition, the separation of program and data memory and the separate program data bus means that program and data memories are accessed simultaneously.

Pipelining leads to substantial performance improvements in typical high energy physics code. For example, the following line of FORTRAN code:

$$\mathrm{XC} = \mathrm{VIX} * (\mathrm{XA} - \mathrm{XZERO}) + \mathrm{VIY} * (\mathrm{YB} - \mathrm{YZERO})$$

would require 23 cycles without pipelining, but only 14 with the pipelining capabilities of the 3081/E.

2.4 THE MICROCODE AND THE TRANSLATOR

The processor's instruction set is not that of the IBM, but is its own microcode, which resembles that of a Reduced Instruction Set Computer (RISC).[4] One could in principal write a compiler to generate the microcode, as done with IBM's 801 project,[5] instead it is generated by a software program, called the Translator. This program reads IBM object code modules, translates them to object microcode, links them together to form an absolute load module for the processor, thus using the IBM object code as an intermediate language. The source of the IBM object code could be the output of a FORTRAN compiler from any IBM compatible vendor or that of a linkage editor on either the VM/CMS, MVS, or MVT operating systems. For all practical purposes the translator step has little impact on the user. It can be looked on as a modified compile or link step. The user will be no more concerned with the $3081/E$ microcode then he would be about the object code from the compiler.

The microcode requires more memory space then the object code. The expansion factor is three in the worst case of no pipelining, and 1.2 in the case of complete pipelining. Nevertheless, at least 30,000 lines of FORTRAN source code can be accommodated per Megabyte of program memory, and many more lines when pipelining is generated.

The advantage of using a translator is the elimination of the complex hardware that decodes IBM instructions into microinstructions. This hardware, called the I-unit by IBM engineers, can consume well over half the total design and debugging effort of a processor. A further advantage of using the translator with the $3081/E$ is that instruction pipelining will be generated with a full knowledge of the context of each instruction.

2.5 INTERFACE

The interface to the $3081/E$ processor is of the same style as the 168/E's. That is, either the CPU or the interface has control of the internal busses. When the processor is not running, all of the processor's memory is directly addressable through the interface. The processor thus appears as a simple slave device on, say, a FASTBUS cable segment. The transfer rate to or from the processor could be over 32 Megabytes per second if FASTBUS cable segments were sufficiently fast or 64 Megabytes per second if a 64 bit interface bus were used. VME and CAMAC interfaces are also being considered.

There are features to make it easier to debug the processor and/or program. The interface halts the processor if there is a parity error on the ABUS, BBUS, or PMD bus. The interface also has registers to allow one to halt the processor when certain conditions arise in a way similar to the Program Event Recording (PER) registers of IBM mainframes. For example, there is a stop on a Store within an address range, a stop on modification of a certain register, *etc.* Debugging some kinds of program error may be more user friendly on the $3081/E$ processor than it is on a mainframe computer.

2.6 PERFORMANCE

To accurately predict the execution speed of the 3081/E is rather difficult, as, in common with many processors, it will depend on program's instruction mix. The pipelining of instructions makes predictions even more difficult. However, three studies have been made to predict the upper and lower bounds of the expected performance.

The lower bound of processor performance can be estimated by assuming that instruction pipelining never occurs. With this assumption the execution time of each IBM instruction is known. Ten different event reconstruction and other programs were traced while in execution to measure the frequency of instructions executed. With these numbers, the performance of the 3081/E processor would be 0.98 to 1.01 times that of an IBM 370/168.

An upper limit is estimated by assuming that pipelining occurs to such an extent that every instruction takes effectively 1 cycle. With the same samples of code, this implies execution times 2.5 times faster than an IBM 370/168; a figure that can not be realistically expected.

A third measure was obtained by translating an inner loop of one of these programs. The loop consisted of 82 FORTRAN statements containing 32 IF statements. Since IF statements break instruction pipelining, it was important to try a loop with a typical number of them. This loop also consisted of several divides and memory references with a non-zero index register. The calculated execution time for one pass through the loop for the 3081/E is 47 μsecs, while for an IBM 370/168 the time would be 71 μsecs. Thus the processor would be 1.5 times faster for this loop.

One can conclude, therefore, that the performance of the 3081/E will be at least that of an IBM 370/168 for typical high energy physics event reconstruction code, or about four times that of the VAX 11/780, and up to 50% faster under the condition that most of the execution time is spent in floating point loops. The performance of the 3081/E is comparable with a well known array processor. The FPS-164[6] has a theoretical maximum execution speed of twelve MFLOPS, while the 3081/E theoretical maximum is 8.3 MFLOPS. In practice,[7] Lattice gauge programs, implemented in microcode of the array processor, achieve about six MFLOPS, while examples of that same code, implemented in FORTRAN, would achieve four MFLOPS on the 3081/E.

3. Use of Processor

In the high energy physics environment, the use of computing resources could be put into two broad categories. The first consists of the thousands of short jobs to write and debug analysis programs, do alignment and calibrations, do physics analysis on processed events, *etc.* This category includes editing, compiling, generation of load modules, using interactive symbolic debuggers, *etc.* The second category is

the long production jobs on raw data or for generation of Monte Carlo events, or in the online environment running filtering programs or analysis programs. Usually there are adequate computing resources for the first category and the limits on productivity are set by user friendliness of the operating system, response time to small needs of CPU time, the fast access to disk files, printers, graphic devices, and the memory paging of the computer. For the second category, the limits on the number of events that can be processed or Monte Carlo generated are set by the available CPU power. It is this category of processing where the inexpensive powerful emulating processors can play an important role.

As the $3081/E$ is a processor and not a computer, it, like other processors, requires support from a host computer to handle input and output operations to physical I/O devices. When multiple processors are to be used (as is frequently the case since one processor is only a fraction of a mainframe computer), this I/O support must be carefully designed for performance.[8] A multi-processor system consisting of five $3081/E$ processors, for example, will have the CPU power of a 3081K, and its I/O support system must be able to supply the data bandwidth to keep the processors busy. In practice, this means tape drives, disks, and channel rates comparable to those found on mainframe or supermini computers.

Much experience has been gained on multiple processor systems with the $168/E$ in both the offline and online environments. The planned uses of the $3081/E$ will build on this experience by preserving the same style of interface as the $168/E$ which worked well and making a few improvements in areas that only became apparent after much $168/E$ experience. The remainder of this section describes how $168/E$'s have been used and thus how we expect the $3081/E$ will be used. The interface of the $3081/E$ is designed for both the online and offline multi-processor environment. The offline environment will be discussed first as it is easier to understand.

3.1 Model of Offline Event Analysis

Consider the following model of how an event analysis program is structured. The typical program has the following steps:

1. Initialize. Initialization starts with the loading of BLOCK DATA statements into memory and continues with reading constants from disk and perhaps calling some subroutines to calculate fixed arrays that will be used in event processing.
2. Read Event. An event is read from a mass storage medium, usually tape. Checks are made to see that the record is an event record and not some other type of record on the tape.
3. Process Event. Event processing involves unpacking the raw data, generating coordinates, finding tracks, fitting tracks, *etc.* It is important to note that this processing uses much more memory for temporary variables than the initial size of the raw data. At the end of event processing, data is compressed into a block for writing to an output tape.

4. Write Event. The event is written to the output tape and the program loops back to read the next event.
5. Print Job Summary. When the event processing is complete, the program prints a summary of the job in the form of statistics gathered, histograms, *etc.*

Four remarks can be made about this structure. First, only the event processing step is CPU intensive. That is, even if the initialization or summary steps take a considerable amount of CPU time, they are only done once, thus don't really matter for a job that will run for many hours. Second, all the steps except the event processing step are I/O intensive. That is, the event processing step usually only has a few print statements for an occasional error message. Third, the program as shown above was written to run on a single processor. That is, it will process events on the same processor doing the I/O and the events are processed sequentially in the same order that they appear on the input tapes. Fourth, there is a large amount of temporary memory space used in the course of analyzing an event and, typically, a complex interrelation between this space and the program in various stages of processing.

It is therefore natural to move the event processing step to the processors, and leave a skeleton program on the host CPU for the other steps. For a single processor, the original program is modified by:

1. inserting I/O calls to download the processor with program and constant data after initialization is completed and before the first event is read.
2. replacing the processing step with I/O calls to send and receive event data with the processor.
3. and inserting I/O calls to receive the job summary data from the processor after the last event is written to tape and before the printing of job summary is started.

These modifications can usually be made with little difficulty by anyone with some knowledge of the program.

For a multi-processor environment, the program can be further modified so that it reads events and sends them to a processor until each processor has an event, then for each event received back from a processor the host program writes it to tape, reads another event, and sends to the next available processor. At the end of the job, the host program would just receive events and write to tape until all processors are empty. Also the job summary data would be received from each processor, and combined before the printing of job summary.

This model of using multiple processors allows a single host program to make efficient use of all the processors while requiring only a single set of input and output devices working on a single data stream. Letting each processor completely handle an event on its own, from input to output, avoids the difficulty of breaking up the program into stages with each stage being run on a different processor. It also allows

multiple slave processors with a simple interface to be attached to a single bus for transferring data.

When the host computer is an IBM compatible mainframe, then there are the following additional advantages:

1. The initialization step can remain entirely on the host. After initialization is done, the labelled COMMON blocks with the initialized data can be downloaded to the same labelled COMMON blocks in the processors without translation of any kind. Thus the initialization code does not need any modification.
2. The output event data received from the processors can be written to tape directly without translation of any kind.
3. The job summary data can be received from a processor by direct copy from labelled COMMON blocks in the processor to the same labelled COMMON blocks on the host, thus the print summary routines do not need modification and can be called directly.

This model has in fact been realized in the use of 168/E processors at many laboratories and universities.[9] The reorganization of the original program has not been radical, indeed it is logical, and once done it has presented little problem even when, at a later date, major changes have been made to the code. In practice, the host computer may be attached to the processors via another computer with event buffers for further efficiency. The buffers allow event data to be unloaded from the processors as soon as it is ready, and new event data to loaded into processors immediately, thus causing minimum idle time on the processors and overlapping physical I/O with processing. At SLAC,[10,11] and CERN,[12–14] PDP-11s were used for attached 168/E processors as early as 1979. A Nord computer was used at DESY.[15] It is also possible for the tape drives to be on a superminicomputer, such has been done with attached 168/E processors at Toronto[16] and Saclay,[17] with some loss of ease due to differences in floating point formats.

3.2 ONLINE USE

Multiple 168/E processor systems have been used in the online environment in a configuration that closely resembles that of the offline systems.[18,19] Similar online systems are being planned for SLC and LEP detectors.[20–24] In the online environment, the input data comes directly from the detector, being processed by the emulators before it reaches the data acquisition computer. The bus interface to the processor is, for example, FASTBUS. Everything else about the running of 'jobs' is virtually the same as the offline environment.

The 3081/E has many important characteristics for the online environment. Being an emulator of a mainframe, programs can quickly be moved from the offline to the online environment. It also has high I/O data capability to minimize deadtime, fast integer instructions including shifting and multiplies for unpacking

data, large memory space to buffer data blocks from different parts of the detector, memory parity checking, *etc.* The separation of program and data memories helps avoid accidental overwriting of program in complex data acquisition systems. Dual port interfacing, which allows simultaneous loading of one processor and unloading of another is easily accomplished[8,18]. Part of the data acquisition system can plug into the internal busses of the processor as has been done with the 168/E.[25,26]

3.3 Other Uses

It is clear that event orientated jobs fit well into the structure described above. But other types of job, such as simulations based on lattices or numerical integration, can also use such a system. Although one's first inclination is to put one processor per node in a lattice simulation, it has been pointed out by Fox[27] that one processor per node will lead to large inefficiencies in the processor communicating with nearest neighbors. At SLAC, nine 168/E processors have been used by running the entire lattice on a single processor, but having different sets of parameters, such as coupling constants or lattice size, running on different processors simultaneously.[28]

Thus, for this type of job the 'event' is a set of parameters, each processor may work on a single 'event' for hours and the job summary printing is the comparison of the results with different parameters. These kinds of jobs require no hardware or software changes to a multiple processor system that can also run the event analysis jobs, thus various kinds of jobs can be submitted to the system just like one would submit jobs to a batch queue on a computer.

In some cases, some limited I/O capability is desirable, 'limited' is important because if I/O capability becomes very important one probably doesn't have a CPU bound job and such a job would run best on a real computer. I/O capability if it is physically done on a host computer and only virtually done on a processor is mostly a question of the software interfacing and not the processor hardware. For example, limited PRINT statements can be accommodated by the processor writing to a buffer in it's own memory, with the buffer only being read out at the end of processing an event as has been done at Saclay with the 168/E processors. In the other extreme, a processor could run part of the operating system; such is the case with IBM's XT/370 where the 370 processor runs the CMS component of the VM/SP operating system while the 8088 processor of the IBM PC in which it is housed handles the physical I/O by emulating the I/O component of the operating system.

4. Conclusion

The 3081/E project was formed to prepare a much improved IBM mainframe emulator for the future. Its design is based on a large amount of experience in using the 168/E processor to increase available CPU power in both online and

offline environments. The processor will be at least equal to the execution speed of a 370/168 and up to 1.5 times faster for heavy floating point code. A single processor will thus be at least four times more powerful than the VAX 11/780, and five processors on a system would equal at least the performance of the IBM 3081K. With its large memory space and simple but flexible high speed interface, the 3081/E is well suited for the online and offline needs of high energy physics in the future.

The project is being carried out as a collaboration between SLAC and CERN DD division. The work has been divided equally between them. Final debugging should occur at SLAC soon with processors being generally available for use by early 1985.

References

1. Paul F. Kunz, "The LASS hardware processor", Nucl. Instr. Meth. 135, 435 (1976).
2. P. F. Kunz, "Use of Emulating Processors in High Energy Physics", Phys. Scr. 23, 492 (1981).
3. P. F. Kunz *et al.*, "The 3081/E Processor", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*
4. D. A. Patterson and C. H. Séquine, "RISC-1: A Reduced Instruction Set VLSI Computer", *Proc. Eighth Ann. Sym. on Computer Architecture, May, 1981.*
5. G. Radin, "The 801 Minicomputer", IBM J. Res. Develop. 27, 237 (1983).
6. Floating Point Systems, Beaverton, Oregon.
7. Ken Wilson, Private Communication.
8. A. Fucci and K. M. Storr, "Using 3081/E Emulators in On-Line and Off-Line Environments", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*
9. A .W. Edwards, N. A. McCubbin and J. P. Porte, "Preparation of Off-line Programs for the Present 168/E and Recommendations for the Future", CERN DD/83/21, Oct 1983.
10. P. F. Kunz, Richard N. Fall, Michael F. Gravina, J. H. Halperin, L. J. Levinson, Gerard J. Oxoby and Quang H. Trang "Experience Using the 168/E Microprocessor for Off-line Data Analysis", IEEE Trans. *NS-27*, 582 (1980).
11. L. S. Rochester, "Microprocessors in Physics Experiments at SLAC", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 204 (1981).
12. C. Bertuzzi, D. Drijard, H. Frehse, P. Gavillet, R. Gokieli, P. G. Innocenti, R. Messerli, G. Mornacchi, A. Norton and J. P. Porte, "On-line use of the 168/E Emulator at the CERN ISR SFM detector", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 329 (1981).

13. D. R. Botterill and A. W. Edwards, "Experiences using the 168/E Microprocessor Within the European Muon Collaboration (EMC)", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 336 (1981).

14. D. Lord, P. Kunz, D. R. Botterill, A. Edwards, A. Fucci, G. Lee, B. Martin, G. Mornacchi, P. Scharff-Hansen, M. Storr and T. Streater "The 168/E at CERN and the MARK II: An Improved Processor Design", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 341 (1981).

15. T. Barklow, thesis, University of Wisconsin—Madison.

16. Steve Bracker, private communications.

17. Jacques Prevost, private communications.

18. J. T. Carroll, M. DeMoulin, A. Fucci, B. Martin, A. Norton, J. P. Porte and K. M. Storr, " Data Acquisition using the 168/E", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*

19. G. Arnison *et al.*, Phys. Lett. 126B, 398 (1983).

20. A. J. Lankford and T. Glanzman, " Data Acquisition and FASTBUS for the Mark II Detector", IEEE Trans. Nucl. Sci. *NS-31*, 228 (1984).

21. A. Lankford, Paper submitted to these proceeding.

22. The L3 Collaboration, "Trigger and Data Acquisition System of L3", CERN/-LEPC/84-5, LEPC/PR3/L3, January 1984.

23. ALEPH Collaboration, " Data Acquisition and Data Analysis", CERN/LEPC/-84-8, LEPC/M46, January 1984.

24. P. Gavillet, B. Heck and F. Udo, "Proposal for Triggering DELPHI", DELPHI Note 83-3 ELEC.

25. D. Bernstein, J. T. Carroll, V. H. Mitnick, L. Paffrath and D. B. Parker, "SNOOP module CAMAC Interface to the 168/E Microprocessor", IEEE Trans. Nucl. Sci. *NS-27*, 587 (1980).

26. J. T. Carroll, J. Brau, T. Maruyama, D. B. Parker, J. S. Chima, D. R. Price, P. Rankin and R. W. Hatley "On-line experience with the 168/E", *Topical Conf. on Application of Microprocessors to High Energy Physics Experiments, Geneva, Switzerland, May 4-6, 1981.* CERN Microproc. 501 (1981)

27. G. Fox, " Scientific Calculations with Ensemble Computers", *Proc. of the Three Day In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983.*

28. J. E. Hirsch, R. L. Sugar, D. J. Scalapino and R. Blankenbecler, "Monte Carlo Simulations of One-dimensional Fermion Systems", NSF-ITP-82-44.

QUESTIONS AND ANSWERS

Q: How big is (the 3081/E) physically?

A. Charlesworth

A: ~ 15" x 19" rack width.

Q: Can one translate SIN-COS and Assembler Code?

D. Notz

A: SIN, etc., can be translated like on the 1681E because instructions and data are separated. Assembler routines will generate holes in the code which does not matter.

Q: Is SIN-COS also re-entrant?

D. Notz

A: Yes, the runtime package is re-entrant.

Q: How hard is it to write a fully optimized translator? Have you already done so?

D. Kaplan

A: The translator has been written. It is not truly an optimizer but makes a single pass through the code, so it's not so complicated. Full advantage is taken of optimizations done by the compiler. For each 3081 machine instruction, the translator searches forward for the first available cycle in which to put an instruction.

Q: What is the time schedule?

A. Brenner

A: Prototype, summer 1984.
Availability, end 1984

e^+e^- Colliders On and Offline Reconstruction Processors

[La Duree poignardee (Time transfixed), 1939, René Magritte, 1898-1967]

THE 370/E EMULATOR AT DESY

Hanoch Brafman
Weizmann Institute, Hehovot, Israel

Dieter Notz[+)]
Deutsches Elektronen-Synchrotron, DESY, Hamburg, Germany

Abstract

A fast general purpose processor which emulates the IBM 370 code is described.

Introduction

There is an increasing demand for computer power in high energy physics. In the era of the forthcoming accelerators people speak about a data production rate of 400 tapes (6250 bpi) per day. All these data have to be analysed and compared with theoretical predictions. In order to support physicists with cheap and IBM 370 compatible computer power the 370/E Emulator has been developed at the Weizmann Institute[1)]. Emulation is defined as "the desire to equal or surpass a rival". In this sense the 370/E is a computer which is from the user's point of view indistinguishable from an IBM 370.

Description

The main components of the 370/E are shown in Fig. 1. The 370/E consists of 14 boards with the dimensions 39.4 cm * 23.5 cm. The whole processor therefore fits into a box of a typical crate size 45 cm * 30 * 40 cm.

The eight memory boards may contain up to 2 Mbytes of memory. If desired the backplane could be easily increased to give space for 4 Mbytes. From the address space point of view the processor can be equipped with 16 Mbytes of memory. The addressing of memory extends across the boards to minimize boundary conditions during instruction fetch. In contrast to other emulators like the 168/E [2)] data and program code are loaded like at the IBM into a combined memory.

The arithmetic and logic unit is divided into five parts: An integer CPU, a dedicated multiplier, two floating point boards and a control unit. The emulator is a synchronous machine with a basic microcycle time of 150 nsec which is narrowed to 100 nsec during shift, multiply and divide. The cycle time of 150 nsec can be decreased to 120 nsec if memory chips with 55 nsec access time instead of those with 70 nsec are used.

+) Talk given by D.Notz in A Review of Triggers and Special Computing Hardware at DESY and talk given in Guanajuato, Mexico.

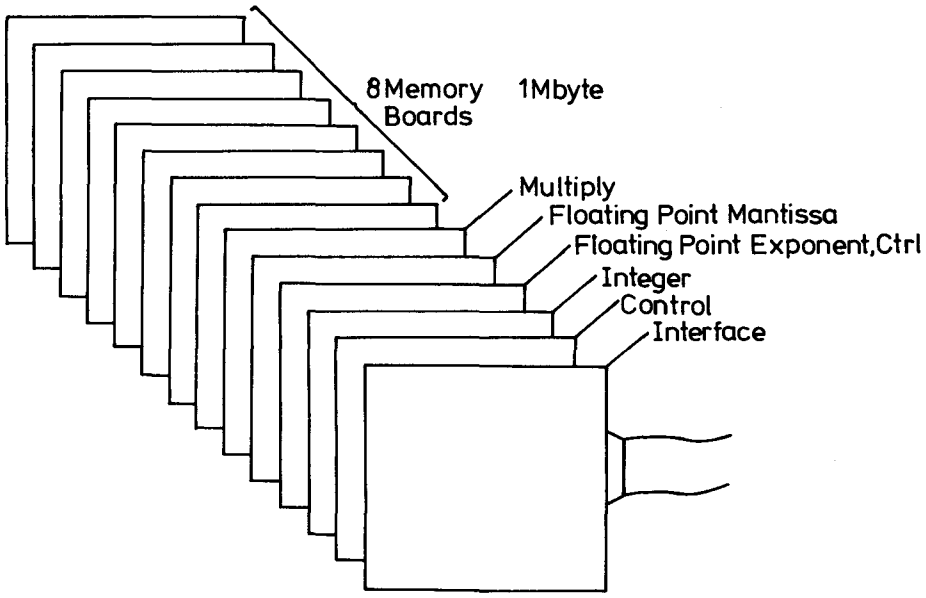


Fig.1 Main components of the 370/E. 2 Mbytes of memory can fit on the eight memory boards.

The design is based on TTL logic making extensive use of the FAST series. In order to achieve high speed the processor contains a multilevel pipeline for prefetching instructions. The pipeline minimizes the contention involved in a combined memory between data and program. It allows to calculate the address of a second operand and fetching it while another instruction is being executed. It also allows microcoding the machine language instruction in preparation to its executive.

The integer CPU contains the 2901B arithmetic and logic unit, a multiplexer for data or addresses and byte oriented right/left shifters. The control unit gets its addresses from the general purpose registers of the integer unit.

The floating point unit performes single and double precision floating point operations. It occupies two boards and includes a dualport register file (29705A) containing the 4 floating point registers, an exponent logic including an alignment encoder and a mantissa ALU with zero detection and encoding logic.

The multiplier performs both integer and floating point multiplication. The basic element consists of a 4 x 4 PROM multiplier.

Input/output of programs and data is performed by the interface. After loading the program and starting it the interface behaves like an IBM channel to local peripherals. File transfer takes place under DMA on a cycle stealing basis. The 370/E emulates SVC, LDSW, SIO and TIO instructions and accepts I/O interrupts.

Input/Output for IBM FORTRAN programs

Via its interface the 370/E is connected to a host computer. At DESY the 370/E is attached by a TMS9900 microprocessor and the online net to the IBM 3081D mainframe. In order to support the user's I/O requests in FORTRAN programs a set of interface routines has been written which makes the online net transparent to the user.

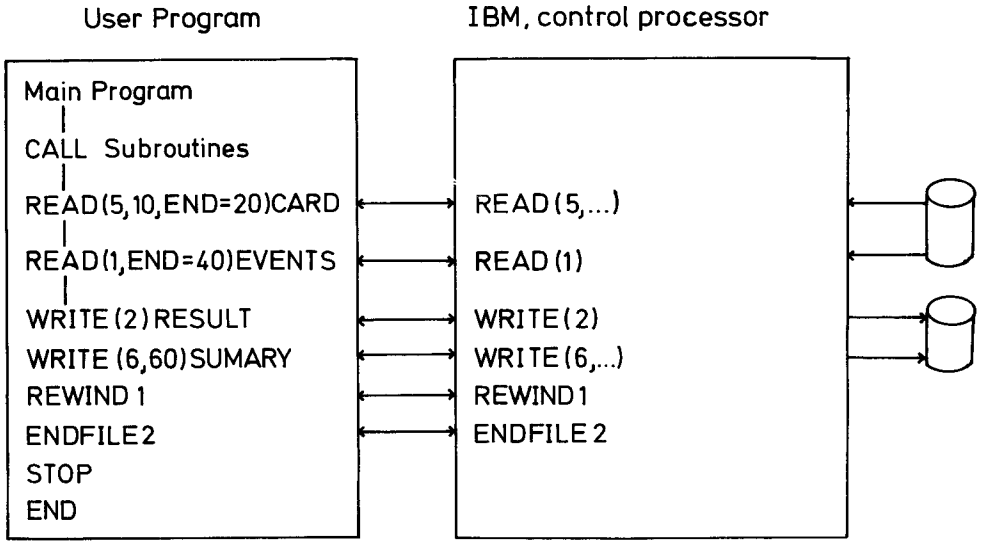


Fig.2 I/O requests of a FORTRAN program in the 370/E are transferred and executed in the host computer.

Fig. 2 indicates the user program on the 370/E written in FORTRAN. All I/O requests to files which reside at the IBM or at another host must be transferred in such a way that the user does not know whether his program runs on the IBM or on the emulator. This is done in the following way: Each FORTRAN program which was generated by the IBM compiler generates a call to #IBCOM for each READ or WRITE (Fig. 3). A lot of parameters like addresses and FORMAT statements are exchanged between the program and the FORTRAN I/O package. #IBCOM then does the formating and transfers buffers to #FIOCS. Here only a few parameters like the unit number, I/O request, buffer address and buffer length is exchanged.

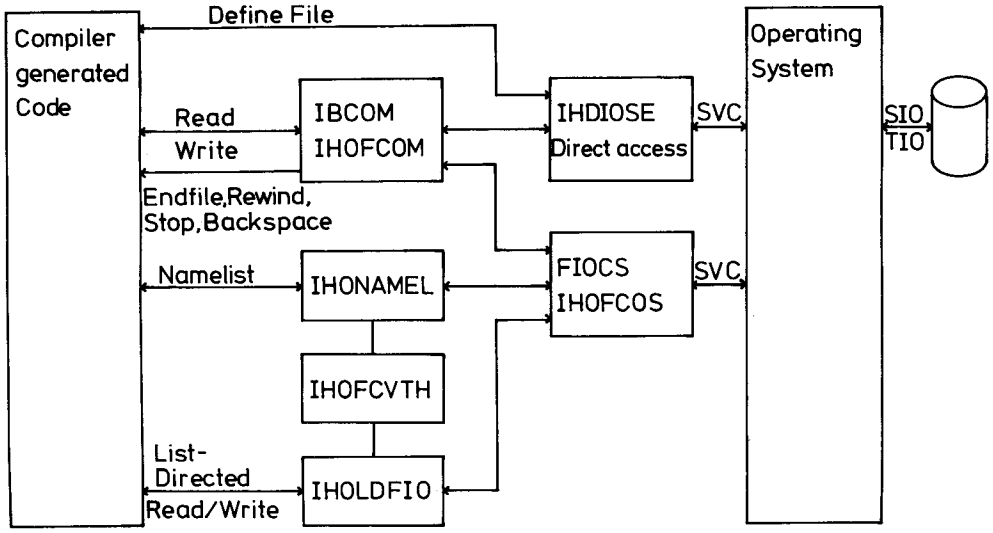


Fig.3 Organization of the IBM FORTRAN runtime library. Each user I/O request gives control to #IBCOM. #IBCOM exchanges buffers with #FIOCS. #FIOCS calls the operating system.

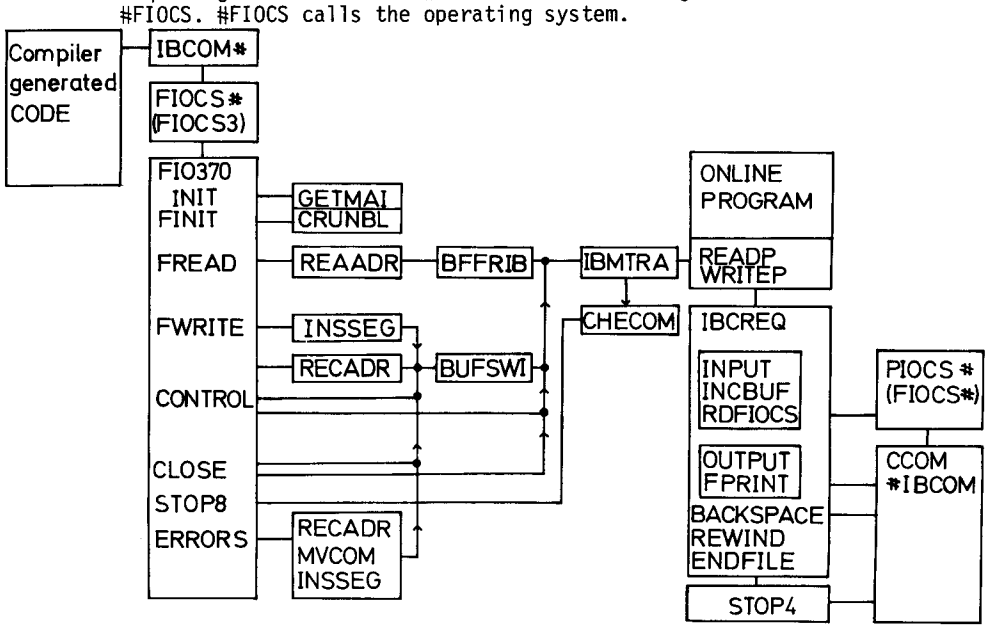


Fig.4 The FORTRAN I/O package for the 370/E. Routines on the lefthand side of IBMTRA run on the 370/E, the other routines on the IBM or on another host (VAX, NORD, LSI11).

In the case of the 370/E the FORTRAN runtime library has been split into two parts[3]: #IBCOM runs on the 370/E processor and #FIOCS runs on the IBM or host. The interface routines are shown in Fig. 4. IBMTRA is the actual transfer routine between the 370/E and the Host. As the information which is exchanged between the 370/E and the IBM is well known the IBM can be easily replaced by a minicomputer like LSI11, VAX or NORD as long as the files are delivered in the IBM format.

Status and performance of the 370/E

The various 370/E installations and their interface techniques are shown in Table 1.

370/E Installations			
Institute		Host and Interface	
Weizmann, Israel	2	LSI 11	
Rutherford, UK	2	VAX 11/780	Ethernet
Imperial College, UK	1	"	"
Birmingham, UK	1	IBM 4341	IBM 2901
DESY, Germany	1	IBM 3081D	PADAC
	1(+2)	VAX 750	PADAC
	1	PDP 11/40	UNIBUS
Bonn, Germany	1	VAX 780	Ethernet
Aachen, Germany	1	LSI 11	
Siegen, Germany	1	VAX 750	PADAC
CERN (DELPHI)	1	VAX 780	Ethernet
CERN (OPAL)	1	"	"
Cornell, USA	6	"	"

Table 1. Installations of 370/E's

At DESY jobs can be submitted to the 370/E from each IBM terminal. The user only has to link his program again including the 370/E I/O modifications (INCLUDE TSOL(SYST370E)). The load module is then loaded to the 370/E. Files reside on IBM disks. Lineprinter output is first stored on disk and then moved to the lineprinter. The user can therefore inspect the 370/E output while the job is running. The 370/E behaves like an IBM, the user does not see any difference whether he runs this job on the IBM or on the emulator. Also error menages like negative SQRT or dumps are generated as on the IBM.

Up to now 1600 jobs with 917 hours (370/E) computer time have been executed on the emulator. The speed of the 370/E processor is 3.8 (Real x 4) to 5.2 (Real x 8 operations) times slower than the IBM 3081D.

Summary

We have described a general purpose processor which emulates the IBM 370. User's load modules are loaded directly without extra translation or linking. The speed of the processor is 3.8 to 5.2 slower than the IBM 3081D. The price is 6000 $ / CPU + 6000 $ / Mbyte memory.

References

1) H.Brafman et al., A Fast General Purpose IBM Hardware Emulator. Weizmann Institute, Dept. of Nucl. Physics., Internal Report, January 1983

2) P.Kunz et al., Experience using the 168/E Microprocessor for Off-line Data Analysis, SLAC-PUB-2418, October 1979

3) D.Notz, The Input/Output Software for the 370/E Emulator, DESY, Internal Report F1-82/01, 1982 and TASSO Note No. 251, March 1983.

A REVIEW OF TRIGGERS AND SPECIAL COMPUTING HARDWARE AT DESY

Dieter Notz
Deutsches Elektronen-Synchrotron, DESY, Hamburg, Germany

Abstract

The DESY computer center, its services and the local area networks will be described. In one example we discuss a trigger processor and how data are taken and analysed in an experiment.

Introduction

The experiments at PETRA and DORIS are now in operation since several years. Only a few trigger processors were added meanwhile and most of the triggers have been presented in the CERN Microprocessor Conference 1981 (Ref. 1). We will therefore describe in this paper how data of an experiment are transferred to the computer center via a network and how they are processed and recorded. One trigger processor which uses an integrated NORD100/E Emulator will be discussed at the end.

DESY Computer Center

DESY has two IBM 3081D mainframes each with 16 megabytes of real memory and 16 channels. The hardware configuration is shown in Fig. 1. Batch programs are processed on both machines. The experiments which send their data via the online net to the IBM are connected to one machine while NEWLIB is running on the other one. NEWLIB is the Editor system written at DESY. It allows to compile and link programs, to submit jobs to the batch queue or to run jobs in foreground for debugging. Both computers have access to the library disks, user's disks, tape units, line printers and plotters.
The DESY Computer Center offers an excellent service for tapehandling. 12000 tapes are stored in the machine room, 50 000 tapes are in the archive. Whenever a user needs a tape he simply defines UNIT=TAPE in his job card //GO.FT02F001 DD DSN=F1CYAA.EX1,DISP=(NEW,CATLG),UNIT=TAPE. The operator then mounts a scratch tape with standard label and write ring. At the end of the job the tape is catalogued by the operating system and the operator removes the write ring. Whenever the user wants access to his data he defines the name of the dataset in his job cards and the catalogue references to the tape number. The correct tape can then be mounted. As the data set name has the user identification in the first six letters the computer center knows who has occupied the tape. When the user does not need the dataset any longer he may delete it. In this case the entry is deleted from the catalogue and the tape will be placed in the scratch poolequipped with a write ring.

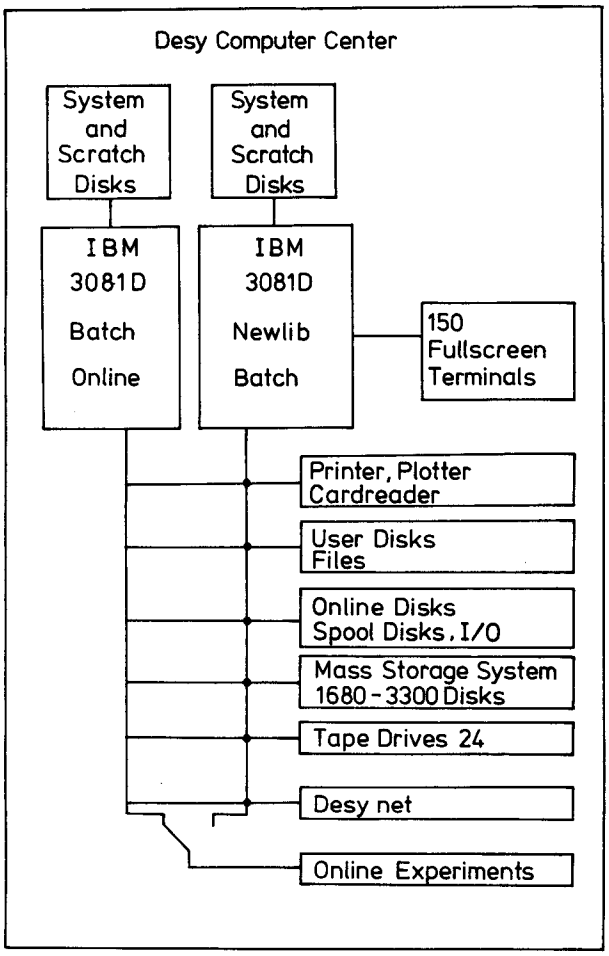


Fig.1 DESY Computer Center. The online experiments are only connected to one computer.

Connections to other Universities

DESY has leased telephone lines to seven other universities or institutions: MPI-Munich, IPP-Garching, KfK Karlsruhe, GSI Darmstadt, University Aachen, University Siegen and Rutherford Appleton Laboratory with SERC net. These lines are used to submit jobs from other institutions to DESY and to transfer files. IBM is installing the European Academic Research Net (EARN) which gives access to Rutherford, Dublin, Paris, Geneva, Madrid, Rome and the US. Inside Germany EARN connects the computer center to more than 10 institutions with IBM computers allowing file transfer, mail and logging on.

Networks on DESY Site

There are several contradicting demands for networks which resulted in the development of four local area networks at DESY.

For the experiments the most important network is the online net. All experimental computers are connected via this net to the IBM. Data are taken by the online computer and then transferred to the IBM disks and tapes. This network is organized like a star and is shown in Fig. 2. Up to 44 computers or microprocessors are connected to the IBM at present. The net is controlled by a TMS9900 microcomputer and seven headers. The TMS9900 sends continuously telegrams to all stations asking the minicomputers for transfer requests. If a computer wants to transfer data the TMS9900 gives an interrupt to the IBM via a 2701 unit. During that time data are already transferred to the header with a speed of 10 µsec/16 bit word. Data transfer between the online computer and the corresponding header is independent of transactions to other headers. When all data have reached the header and the IBM can accept the request data are moved from the header to the IBM via a second 2701 unit with parallel data adapter with a speed of 4 µsec/16 bit word.

In average the experiments may send one to two data blocks per second to the IBM with 32 000 bytes/block.

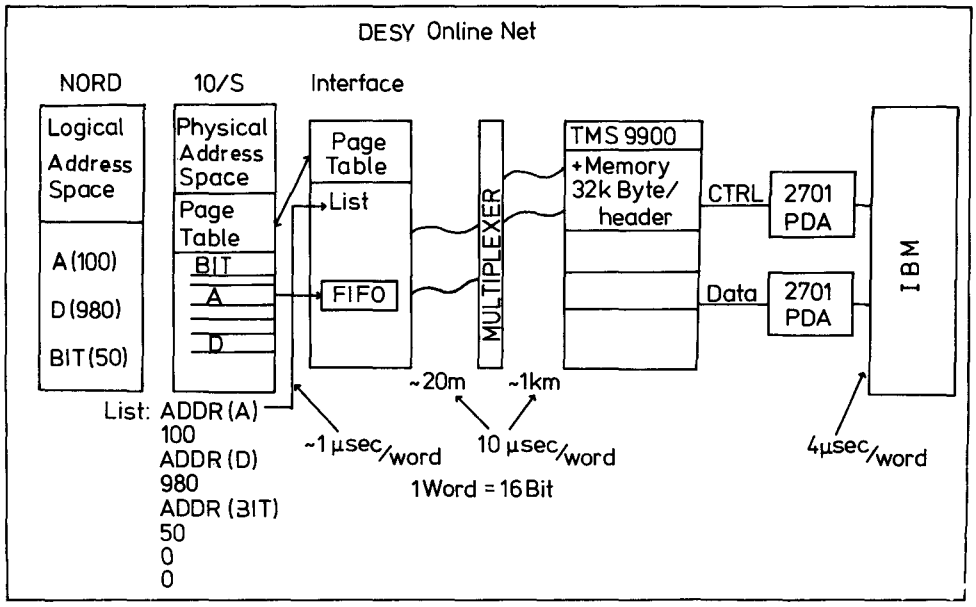
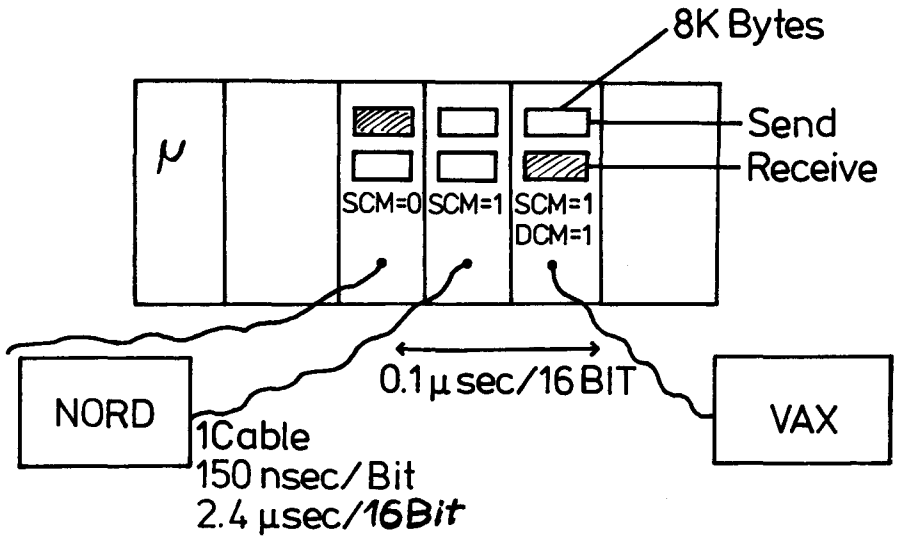


Fig.2 DESY Online Net. The TMS9900 controls the net consisting of 44 minicomputers. The computers are connected via multiplexers and seven headers. The interface gets a copy of the pagetable to map logical and physical address space.

In order to connect several computers within one control room or experiment a fast packet switching system (FPSS) has been developed. The layout of this system is shown in Fig. 3. In the central node of that system a Microprocessor controls several buffers. Each computer is connected by a single coax cable to a pair of buffers, one for receiving and one for sending packets. The transfer rate between computer and buffer is 2.4 µsec/16 bit word and between buffers within the node 0.1 µsec/16 bit word (Ref. 2).



Fast packet switching system.

Fig.3 The fast packet switching system is used to connect several computers of an experiment or control room. Transfers between computers and their buffers in the node are independent of each other.

In preparation is a net for terminals allowing a connection between terminals and several computers.

In addition DESY has a net which connects 25 graphic work stations and the services of the Bundespost to the two IBM computers. This net is controlled by 23 NOVA computers and the speed varies from 0.5 Mbit (up to 1.2 km) to 7 Mbit (up to 300 m).

Data Taking, Filtering and Recording

Fig. 4 shows a typical set up for an experiment at DESY. Data are taken by a PDP, NORD or VAX online computer and stored in buffer or on disk. The online computer monitors the experiment with the help of event displays and

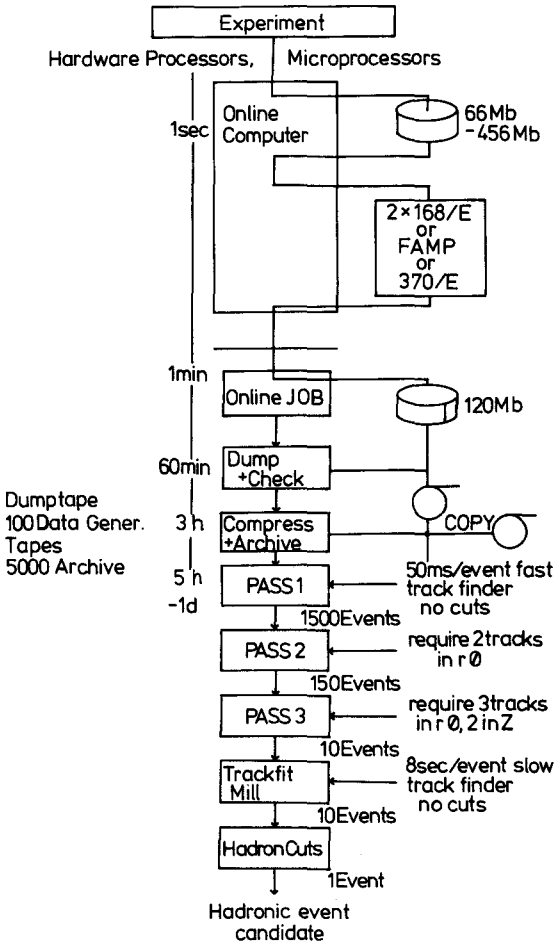


Fig.4 Typical online application. Data are taken, buffered and filtered by the online computer. After transfer to the IBM the first analysis jobs are started automatically.

histograms. Before data are sent to the computer center they are filtered and analysed by fast processors. The JADE experiment uses a NORD50 and a Fast Amsterdam Multiprocessor (FAMP) system with 3 MC68000 processors (Ref.1, Ref.3) while IBM compatible emulators 168/E (Ref.4) and 370/E (Ref.5) are used in the TASSO experiment. The analysed events are then moved to the IBM and written to disks with 100 Mbytes capacity. When the disks are nearly full a dump job copies data to tape and catalogues the

tape number. In addition the dump job activates a check job, a job for event reconstruction and an archive job. As all collaborators have access to the IBM catalogue one can easily check the status of the analysis programs and activate own programs. For normal event rates event reconstruction is finished one day after data taking, track fitting and event scanning is completed after one week.

A Trigger Processor for the Vertex Detector

A small, high precision drift chamber is used in the TASSO experiment as vertex detector. The inner radius is 6.5 cm, the outer radius 15 cm, the active length 58.6 cm and the drift distance varies from 3.5 to 4.5 mm. For track reconstruction in the $r\Phi$ plane three memories are used. One memory is connected with its 10 address lines to the 4 inner layers, the second one in the same way to the 4 outer layers. These memories generate for each possible track combination in the corresponding address a track element number on the output pins. A third memory combines track elements from the innrer and outer layer and generates a trigger signal. The time needed to produce a fast trigger signal is 100 nsec drifttime + 150 nsec in first memory + 300 nsec in the second memory and trigger logic = 550 nsec (Fig.5). This trigger processor does not introduce any deadtime as the repetition rate of PETRA is 3.8 μsec. In addition to the $r\Phi$ trigger the vertex detector is also used to determine the position of the vertex along the beam by using charge division on the wires. The charges which appear at the end of the sense wire try to equalize during the gate time by the internal circuit. Therefore the charges which are measured by the ADCs must be corrected depending on the gate width. In addition one has to subtract pedestal and correct for ADC gains. All these computations are performed by a NORD100/E emulator with the help of look-up tables. The ADCs are addressed like a memory in order to save time for readout. After the gain correction the position on a wire and finally the vertex is computed. The program to find a vertex has a time out of 10 msec. 25 msec are needed to format the data. The total time of 35 msec is comparable to the readout time of a complete event. On a second trigger level the event may be rejected before sending it to disk.

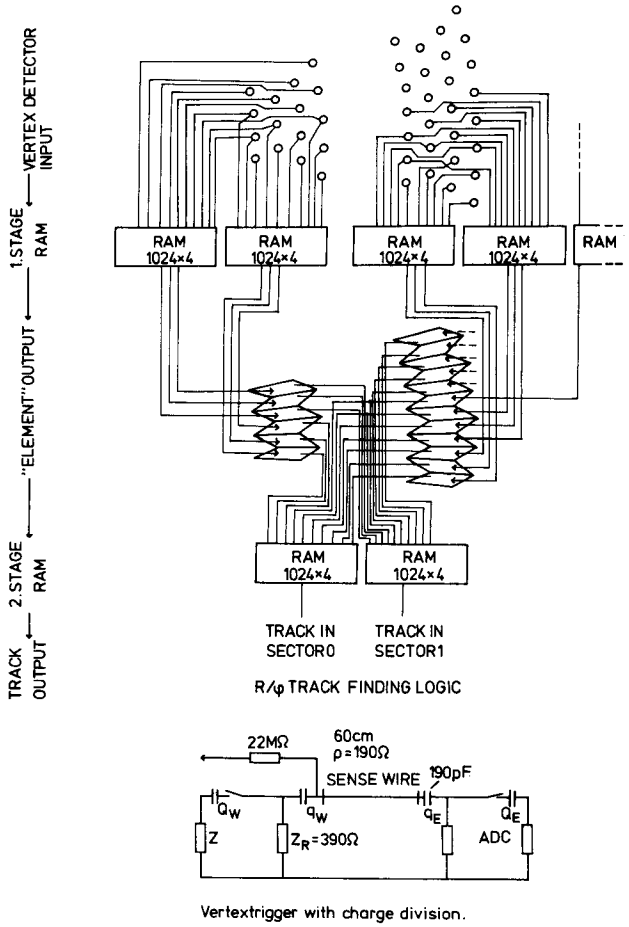


Fig.5 Vertex detector trigger. The fast rΦ-trigger is ready after 550 nsec. In the charge division trigger a NORD100/E emulator determines the result in 10 msec. 25 msec are needed to format the data.

References

1. Topical Conference on the Application of Microprocessors to High-Energy Physics Experiments, CERN, 81-07

2. H.Frese, G.Hochweller, H.Krechlok, R.Schmitz, FPSS A Fast Packet Switching System, IEEE Transactions on Nuclear Science 28, 2377-2379, 1981

3. P.Lankarinen, The JPATH filter algorithm for JADE track trigger. JADE internal note 112 (1984)

4. P.Kunz et al., Experience using the 168/E Microprocessor for Off-line Data Analysis, SLAC-PUB-2418, October 1979

5. H.Brafman, R.Fall, Y.Gal. A Fast General Purpose Hardware Emulator, Weizmann Institute, Internal Report 1983.
D.Notz, Input/Output Software for the 370/E Emulator. Internal Report DESY F1-82/01 (1982). TASSO Note No. 251 (1983), 295 (1984).

QUESTIONS AND ANSWERS

Q: Can you mention the current plans and development projects concerning HERA?

S. Conetti

A: We will do it similar to PETRA and PADAC. The experiments might use VAX with CAMAC and FASTBUS.

Q: Since you already have 2 IBM 3081D's, the one $370/_E$ is a very small increment in computing power. Are you planning to install many more $370/_E$'s, or is there some other advantage to using the $370/_E$?

D. Kaplan

A: TASSO will be happy with 4 $370/_E$'s. Since PETRA will shut down in 2-1/2 years, there is little incentive to add, say, 20 $370/_E$'s to system, unless HERA is built.

Q: How many $370/_E$'s can you put onto your 3081D before the requests for I/O saturates the 3081D?

M. Fischler

A: About 10, thus ~ tripling the 3081D power . But the computer center might not like it.

Q: Could you comment on the relative use of emulators ($373/_E$ and $NSRD/_E$) and single board micro computers (like the Intel 286 reported on by Paolo Franzini in the last talk).

I. Gaines

A: No Answer

THE ARGUS TRIGGER PROCESSOR "LITTLE TRACK FINDER"

H.D. Schulz
Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract : A fast secondary trigger processor has been built that finds and counts circular tracks in the r-φ-plane of a large drift chamber. The trackfinding process takes 12 μs plus 4 μs for each encountered track element. The device consists of a table driven ECL-processor connected to the experiment's computer for table preparation and checking. In this way speed and flexibility can be reconciled.

The "Little Track Finder" (LTF) is part of the trigger system of the ARGUS detector. ARGUS is a magnetic 4π-detector set up in one interaction region of the 5 GeV e^+e^- storage ring DORIS at DESY in Hamburg. Its solenoid magnetic field is centered on the interaction point and the particles generated by an e^+e^-- interaction pass subsequently through the following detector components within the field :

- A drift chamber (DC) with in total 6000 drift cells arranged in 36 cylindrical layers. Half of the signal wires are stretched parallel to the beam line, the other half being stereo wires.
- A cylinder of 64 scintillation counters to measure the time of flight (TOF) of charged particles.
- A lead scintillator calorimeter to measure the energy of showering particles.

Trigger system : Since the rate of e^+e^- bunch crossings is 1 Mhz, whereas the number of good events is expected to be only of the order to 1 Hz a fast selective trigger system is mandatory. The ARGUS trigger is a two step trigger making use of all the above mentioned detector components. The primary trigger relies on fast coincidences between TOF and shower counters and makes use of the total energy deposited in the shower counter system. Its total rate of 50 - 100 Hz is an order of magnitude larger than is tolerable both from the expected rate of good events and the detector readout time of 30 ms. To further reject background events the primary trigger starts a secondary filter, the LTF which finds and counts good tracks in the rφ-plane of the drift chamber and eventually classifies the event to be read out.

Design idea : The operation time of the LTF is severely constrained by deadtime consideration to be less than 50 μs. This rules out a pure microprocessor application, but also a pure hardware solution was rejected because of its inflexibility and clumsiness. To compromise between the speed of a hardware device and the flexibility of a programmable processor, we built the LTF as a table driven ECL-device that is connected to the online computer of ARGUS for software support. The tables that control the operation of the LTF are prepared by running a FORTRAN program on the online computer once for an experiment. The program requires simple physical input information and is flexible enough to cope with different experimental requirements. The constructed tables are then loaded into the memories of the LTF which is now able to do high speed operations without further intervention of the computer. In this way the flexibility of a software solution and the speed of a hardware device could be reconciled.

The layout of the LTF is schematically shown in Fig. 1. There are four types of internal memories :

1) the wire input boards (WIB), one for each drift chamber layer plus one for the TOF-layer. The WIBs are connected to hit-registers within the TDC-modules (LeCroy 4291B) of the DC and TOF-layers and hold the hit information of the actual event. The drift time information is not used since the drift cells are small (18 x 18 mm^2).

2) The mask memory (MM) is loaded from the online computer with precalculated information. It contains up to 2 k masks. Each mask consists of a sequence of words, as many as there are layers connected to the LTF. These words contain the addresses of wires in the WIBs computed such that together they constitute a possible good track in the detector.

3) The contents addressable memory (CAM) of 64 words, preset with fixed information from the computer, contains acceptable hit patterns for good tracks. (To account for DC inefficiencies there may also be good tracks with one or two hits missing.)

4) The increment memory (IM), loaded with calculated information from the computer.

Sequence of operations : Each primary trigger initialises the LTF and strobes the contents of the hit registers into the WIBs. Then a series of mask cycles is executed. The words belonging to one mask are in parallel used to address wires in the WIBs and the read hit pattern is compared to the contents of the CAM. If there is a match, a track counter TC is incremented. If the TC exceeds a preset value, the event is accepted. If the TC does not exceed the threshold and the end of the mask memory is reached a reset signal is sent to the detector electronics.

Operation time : One mask cycle, consisting of 8 subcycles, takes 170 ns. Going through 2 k masks strictly sequential would mean an operation time of 350 µs. This time can be considerably reduced by a simple trick. One notices that one signal wire (or group of wires) participates in many masks. Once it is detected that this wire is not hit, it is of no use to check on all the rest of the masks containing this same wire. With a suitable ordering of masks it is possible to skip over useless cycles by letting the mask address incrementation depend on the result of the last mask cycle. The incrementation control is effected by the contents of the IM. The operation time of the LTF then depends on the number of hits in the DC. For an empty DC the operation time becomes 12 µs, each encountered track element adds 4 µs to this time. The measured mean operation time of the LTF during normal detector running is 20 µs, more than an order of magnitude less than in the strictly sequential mode.

Results : The LTF reduces the trigger rate of ARGUS by an order of magnitude as compared to the primary trigger if one asks for 2 tracks seen in the DC. In doing this it introduces less than 0.2% deadtime into the detector operation. Its track finding efficiency has been measured to be 97%, mainly determined by the DC efficiency. Double counting of one track is reduced to 3% by ignoring the second one of two consecutive matches in the track finding process. The LTF can be easily and thoroughly tested online by computer since all its registers can be accessed via CAMAC. The LTF met its design goals and has proven to be a very reliable tool.

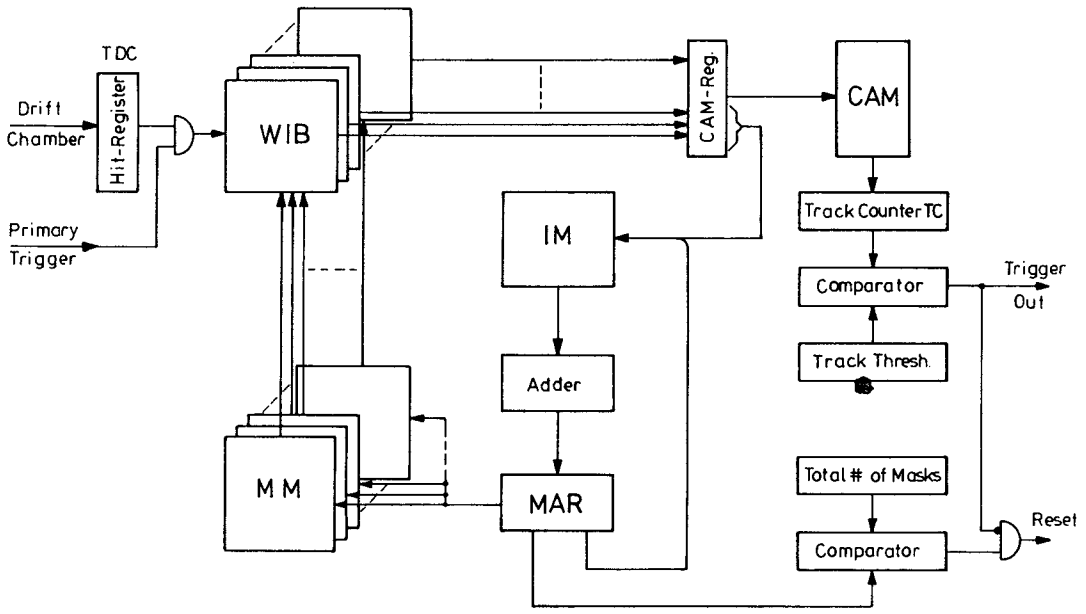


Fig.1 Blockdiagram of Trackfinder

QUESTIONS AND ANSWERS

Q: a) What is the maximum number of wires than can be counted into mask?

b) Can the first-stage region be changed in size?

T. Brody

A: a) Each word with an included wire set has one wire address and three bits for moving in neighbors.

b) It can be moved anywhere, can be of size 1, 2 or 3 words, but must have a bit in the mask-defining word.

Q: How much memory in the processor and what is the duplication cost?

D. Kaplan

A: 32K words (some 12-bit and some 16-bit). It cost 100,000 Deutsch Marks originally; the duplication cost would be less.

The Use of MC68000 Microprocessors in the TASSO Experiment

V. Mertens
Physikalisches Institut Universität Bonn,
Nussallee 12, D-5300 Bonn 1, W.-Germany

Abstract

A MC68000 based microcomputer system for the enhancement of CAMAC data acquisition systems has been developed at Bonn and DESY. Good software support is available including FORTRAN-77 with real-time extensions and PASCAL. The system will be used in various applications in the online system of the TASSO detector at the e^+e^--storage ring PETRA.

1. Introduction

In this paper the hardware and the software of a microcomputer system for data processing in CAMAC based data acquisition systems is described. Its main purpose is the support of online computers in data acquisition and control of experiments. The system is called '68CAMICRO'.

The 68CAMICRO system has a modular structure. It consists of several CAMAC modules which are grouped around a general-purpose processor module based on the microprocessor MC68000. Interface modules make the system applicable to different levels in CAMAC configurations, e.g. it can be employed as Auxiliary Crate Controller for processing in normal CAMAC crates or as a Front-End Processor, sharing the access to a complete CAMAC assembly with the online computer. With real-time FORTRAN-77 and PASCAL good software support is available.

The system will be used in the online system of the TASSO detector /1/ for data reduction, formatting of events and for readout of full events into the online computer.

2. Hardware and Software of the 68CAMICRO System

To date, the hardware consists of three different types of modules. The major goals during the design phase were modularity and expandability of the system. Therefore an important aspect is the introduction of a system-wide interconnection. Modules are linked together by a multipole connection at the front panel of each module thus forming different functional units.

The modules are grouped around a fast general-purpose processor module named 'WS68K' ('WORK STATION based on MC68000 PROCESSOR'). The WS68K is mandatory in each application. Alone or in connection with extension modules, the WS68K can be used as a 'Personal Work Station' for general computing. CAMAC data acquisition is achieved on the lowest level by a WS68K working together with an 'AUXILIARY CRATE CONTROLLER INTERFACE (ACCI)'. This combination acts as an 'AUXILIARY CRATE CONTROLLER (ACC)' for the control of a single CAMAC crate. A 'BRANCH INTERFACE (BI)', which enables the WS68K to access a complete CAMAC branch consisting of up to seven crates is under development. Most recently an 'AUXILIARY SYSTEM CRATE CONTROLLER INTERFACE (ASCCI)' has been developed, which allows the WS68K to gain access to the complete multi-branch CAMAC configuration of an experiment (up to 50 crates) when residing in a GEC Elliott 'System Crate' /2/. The development of an 'UNIBUS INTERFACE' for a direct link to a VAX or a PDP 11 host computer is envisaged.

The WS68K module employs the MC68000 in a 10 MHz version. It offers 256 Kbytes of fast (150 nsec) dynamic RAM with byte-wise parity check and a 192 Kbytes EPROM area for system programs or fixed user programs. In addition, the WS68K provides two RS232 serial ports for connection of a terminal and a host computer. It is built up on a multi-layer PC-board in a single slot CAMAC frame.

The ACCI module allows accesses of the WS68K to all CAMAC modules in the local crate via standard A-2 Crate Controllers. The data paths to modules are 16 bits wide (optionally 24 bits) during both read and write cycles. To the online computer the ACCI appears as an ordinary CAMAC module. The data paths to the online computer are 16 bits wide. A buffer area consisting of 2 x 128 bytes for both directions, made up by First-In-First-Out memories (FIFOs), serves as the communication area. The data exchange via this buffer is supported by dedicated hardware (control and status registers for both the online computer and the WS68K). An intelligent interrupt controller in the ACCI is able to handle 8 interrupt sources:

4 out of 24 possible LAM-signals from modules (selectable via 'jumpers'), 3 NIM front panel inputs and an interrupt from a real-time clock. Two NIM trigger outputs are provided. A watchdog timer supervises the overall operation of hardware and software. The ACCI is also housed in a single-slot CAMAC frame and designed on a multi-layer PC-board.

In order to provide access for the WS68K to all crates in a CAMAC system the ASCCI module was designed. The WS68K/ASCCI combination has necessarily to reside in a GEC Elliott System Crate, where it has the same status as the interface to the main online computer. The ASCCI structure is different from that of the ACCI due to the different use of the System Crate Dataway during system cycles. In addition, the buffer area for communication with the host was enlarged to 64 Kbytes of dual-ported static RAM. The ASCCI occupies two CAMAC slots and has been assembled using the wire-wrap technique.

Software development for the MC68000 can be made either on the TASSO VAX 11/750 or on the target system itself. As cross software we use mainly the FORTRAN-77 compiler 'RTF/68K' /3/. This compiler comprises, with few omissions, the full FORTRAN-77 specification, the extensions of the VAX FORTRAN compiler and special real-time extensions to take full advantage of the architecture of the MC68000 and the hardware structure of the 68CAMICRO system. A PASCAL cross compiler is also available.

The system software consists of a 60 Kbytes stand-alone monitor written in FORTRAN and a run-time library. This library works together with the FORTRAN compiler and is partly written in FORTRAN and partly in MC68000 Assembly language.

3. Implementation of the System in the TASSO Experiment

Fig. 1 shows schematically the application of the 68CAMICRO system in its ACC function consisting of a WS68K and ACCI. In several crates of the TASSO readout system ACCs will be installed to support the main online computer in reading out parts of the detector. One application will be in the pedestal subtraction and data reduction of ADCs. Another application is the formatting of data from the liquid argon electromagnetic calorimeter /1/, including the suppression of data caused by noise and pickup in the readout electronics. A third application will be the efficiency check for the fast 'hard-wired' trigger processor which recognizes track patterns in the central detector /1/.

The most important application of the 68CAMICRO system however is in the main readout stream of the TASSO detector: At the moment, the TASSO experiment employs a coupled system of NORD10/NORD100 online computers /4/. Starting from summer they will be replaced by a VAX 11/750 with data acquisition running under the VMS operating system. Unfortunately, the real-time behaviour of VMS has some drawbacks. In the VAX 11/750 the overhead from interrupt handling, i.e. entering and recovering from interrupt routines, takes about 4 msec /5/. The time required to set up a single CAMAC block transfer (DMA) has been measured to be about 1.7 msec /6/. Since the readout structure of the TASSO detector involves some 50 DMA transfers this would give an unacceptable contribution to the readout time and increase the deadtime of the experiment.

By introducing a WS68K and an ASCCI in the System Crate as Front-End Processor to the VAX one can eliminate these overheads. (fig. 2). In this configuration the WS68K/ASCCI reads in the raw data via 4 branches, formats the events and stores them in the 64 Kbyte buffer area of the ASCCI. Since the event length is typically 5 Kbytes multiple event buffering is permitted. The buffer of events may now be read by the VAX in a single DMA transfer.

4. Remarks on Performance

The new arrangement is expected to improve significantly the performance of the data acquisition system of the experiment. The reasons may be summarized as follows:

The overhead from interrupt handling in the VAX 11/750 of about 4 msec compares to less than 100 μsec in the WS68K. No set-up time for DMA transfers is introduced because the data transfer between the MC68000 and CAMAC modules is done under program control.

The CPU performance of the WS68K has been found to be approximately equal to that of a VAX 11/780, if the features of the FORTRAN compiler /3/ are fully exploited and if only integer arithemetic is used, as in this application /7/.

In addition, two hardware features in the 68CAMICRO system provide for efficient CAMAC accesses. Firstly the complete CAMAC equipment is mapped onto the address space of the MC68000 and is not connected via peripheral I/O-channels. As a result, a CAMAC location is accessed immediately from

FORTRAN programs as an ordinary variable. Secondly during read transfers from CAMAC modules, which form the bulk of CAMAC accesses in the present application, the MC68000 stores the read data under control of the CAMAC timing signal S1. After the signal S1 the microprocessor can decouple itself from the running CAMAC cycle and continue with further activities. This shortens CAMAC read accesses as seen by the MC68000 by roughly 30 %.

An important motivation for the development of 68CAMICRO was the possibility of enhancing existing CAMAC systems at relatively low cost without having to wait for the development of new standards.

Acknowledgements

I am grateful to Prof. Dr. H. M. Fischer for supporting this project and Drs. H. Hartmann, J. Harvey, H. Kolanoski, D. Notz, K. Rehlich and A. M. Rushton for many helpful discussions.

I would like to take this opportunity to thank Dr. H. von der Schmitt for two years of enjoyable teamwork.

References

/1/ TASSO Collaboration, R. Brandelik et al., Phys.Lett. 83B(1979)261; TASSO Collaboration, M. Althoff et al., Phys.Lett. 139B(1984)126; V. Kadansky et al., Phys.Scripta 23(1981)680.

/2/ GEC Elliott, System Crate Catalogue, 1981

/3/ H. von der Schmitt, RTF/68K Real-Time FORTRAN-77 Compiler for 68K Processors - Manual of Compiler and Run-Time Library, Internal Report Bonn University (in preparation)

/4/ T. Barklow and J. Harvey, IEEE Trans.Nucl.Sci., 30(1983)3823

/5/ C. Verkerk, this conference

/6/ F. Gagliardi and A. Vascotto, Considerations on the Data Acquisition Timing on VAXs, CERN, OC/DEC/83-1

/7/ V. Mertens and H. von der Schmitt, Proc. of the Three-Day In-depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25, 1983

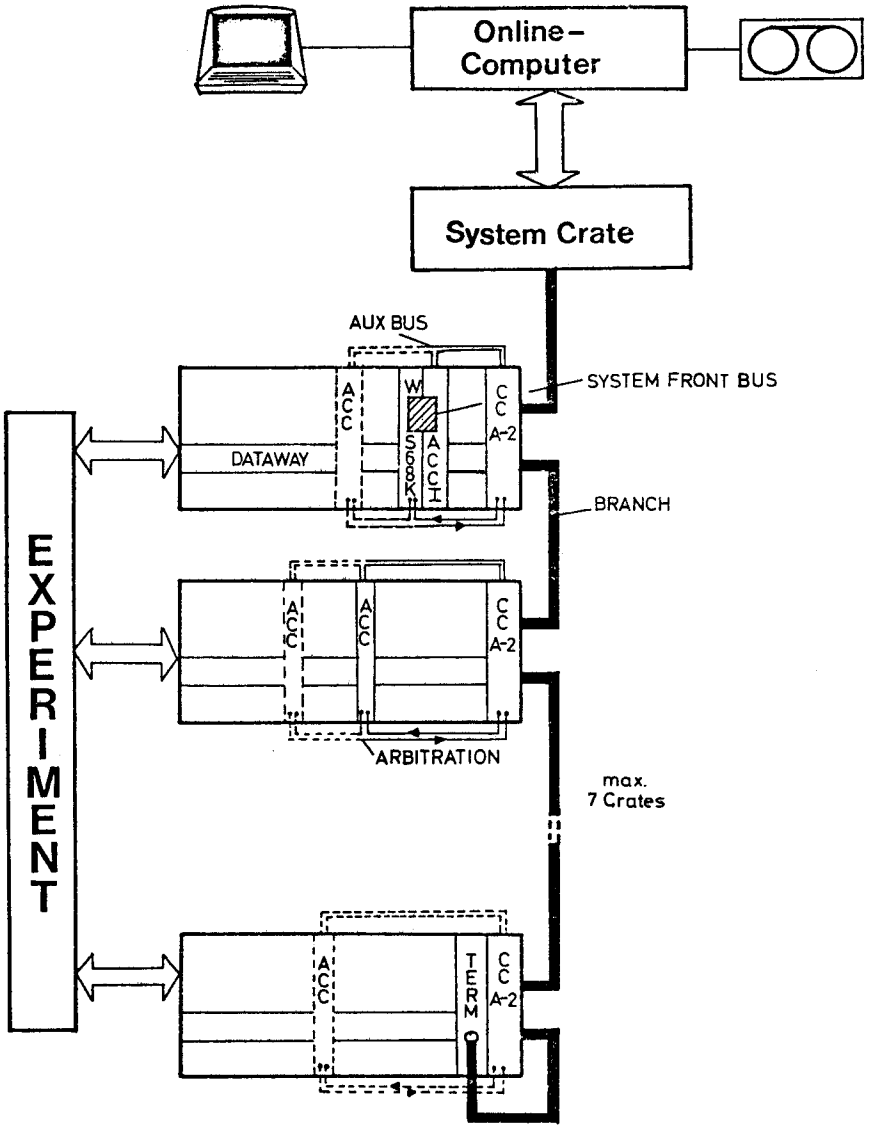


Fig. 1. Application of the system as an Auxiliary Crate Controller in several of the normal CAMAC crates of the TASSO experiment ("ACC" is synonymous with "WS68K/ACCI").

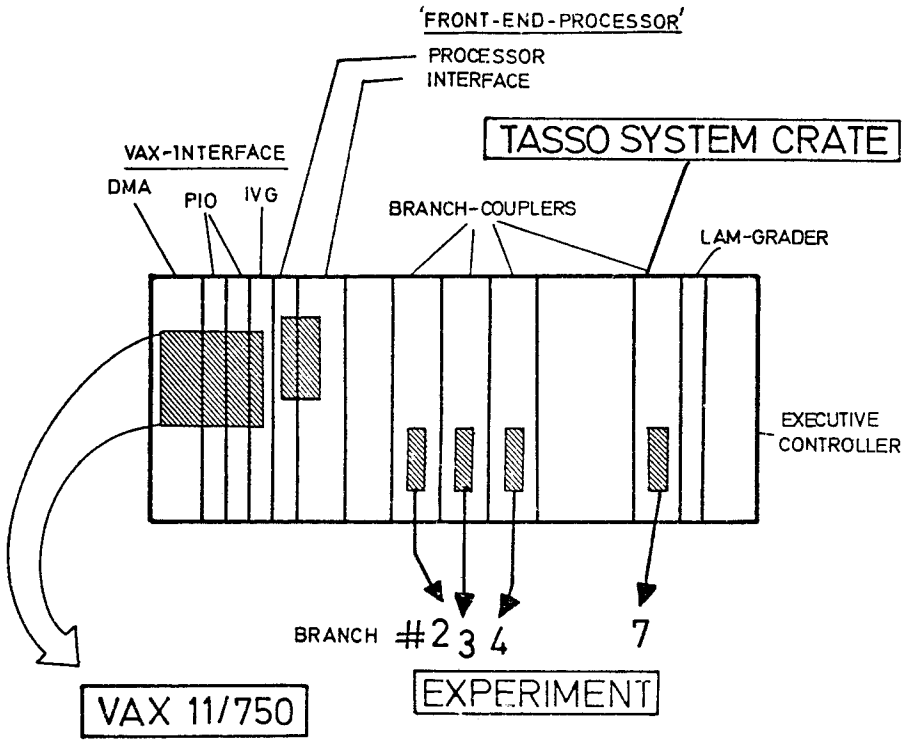


Fig. 2. Application of the system in the System Crate as a "Front-End Processor" to control the main readout stream.

TRIGGERS AND SIGNAL PROCESSING AT CESR

Paolo Franzini
Columbia University

ABSTRACT

In the following we discuss the triggers and signal processing at the Cornell Electron Storage Rings (CESR). Two detectors, CLEO and CUSB, reside at the two interaction regions of the accelerator where they each are exposed to an average crossing rate of 1.2 MHz. The individual strategems of reducing such rates to the acceptable rate of 3 Hz and the subsequent signal processing resulting in data summary tapes are discussed.

1) INTRODUCTION

CESR is an e^+e^- collider operating in the $\sqrt{s} \sim 10$ GeV energy region, where the majority of bound triplet S $b\bar{b}$ states, T's, are produced. Three such states lie below the free flavor threshold, and by observing the photon transitions amongst them, the six χ_b's were discovered by CUSB at CESR [1]. The fourth T, also discovered at CESR, lies above open b flavor threshold and serves as source for the study of the b-flavored mesons (B's) which were explicitly first reconstructed by CLEO [2]. All these activities happened since 1980, while the machine luminosity increased by over a factor of fifty and the detector triggers have continuously evolved in order to reduce data rate without loss of events of interest. Figure 1 shows the schematic layout of the CESR accelerator complex: a LINAC is the source of the electrons and

positrons, and a synchrotron accelerates them to the desired energy after which they are injected into the storage ring. The regions where the electrons and positrons collide are designated by the resident detectors.

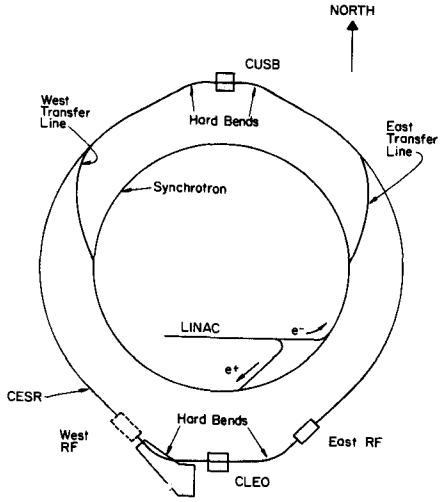


Figure 1 CESR Accelerator Complex

The CESR control system uses three PDP 11/34 connected to a VAX 11/750. It should be noted that CESR can individually set all focussing quadrupoles, as well as correcting sextupoles and steering verniers, thus having enormous freedom in setting tune and choosing lattice. The 11/750 is a recent addition and is used essentially as a clearing house for the 11/34's from which running conditions for the various components of the machine are down loaded and saved. Machine operations, computer assisted fine tuning and orbit corrections are handled by this cluster of computers. Calculations for new lattices which are human vernier assisted, and searches for safe places in the tune plane labyrinth, are done instead on a stand alone VAX 11/780 which the machine physicists share with other CESR developement programs.

Table 1 summarizes the present running environment at CESR, which raises the question of why there is a need at all for trigger requirements at e^+e^- colliders (posed by physicists contemplating the formidable conditions at hadron colliders or fixed target machines).

Table 1: CESR Running Environment

	Peak	Average (for ex. May 8th)
Luminosity	0.03 $nb^{-1}sec^{-1}$	0.01 $nb^{-1}sec^{-1}$
Hadronic event rate	0.75 to 0.12 Hz	0.25 to 0.04 Hz
μ-pair event rate	0.02 Hz	0.007 Hz
τ-pair event rate	0.02 Hz	0.007 Hz
Bhabha event rate	0.2 Hz, and up	0.1 Hz, and up
Total event rate	1.0 Hz, and up	0.35 Hz, and up

The crux of the matter is however, that the crossing rate is 1.2 MHz while the maximum acceptable rate is less than (or equal to) a few Hz. Therefore one needs to reduce the 1.2 MHz rate to $\lesssim$ 3 Hz without losing the above listed real events.

The modus operandi is typically as follows:

1) At each crossing assume there is an interaction, therefore begin signal processing.
2) Meanwhile, generate a trigger.
3) If the trigger is valid, continue signal processing until event is deposited into computer.
4) If there is no trigger, reset (if possible), and resume.

In the next sections we discuss how each detector specifically accomplishes this.

2) CLEO

CLEO is a general purpose detector based on a large magnetic spectrometer. The basic components of the detector, viewed radially outwards from the beam line are: Proportional Wire Chambers (PWC) surrounding the beam pipe, drift chamber (inner drift), superconducting coil, drift chambers (outer drift), dE/dx proportional chambers, Time Of Flight counters (TOF), octant shower detectors, magnetic yoke, hadron absorber, muon chambers. Luminosity detectors and various end cap complements of the abovenamed components are

located at their logical places [3]. The complete analog and digital electronics for CLEO is in crates attached to the detector inside the radiation area. In table 2 we list the detector elements of CLEO, each requiring analog processing and conversion to digital form.

Table 2: CLEO Active Elements

	Active Elements
Tracking	12,000
Electromagnetic shower detector	10,000
Particle identification	10,000
Total number of signals	32,000

CLEO's trigger system includes four major components which allows triggering on charged or neutral particles through track recognition or energy deposition in the detector. The components are: (i) fast track segments from the inner drift chambers and from the PWC's that provide z coordinate information, (ii) fast logic signals from the TOF counters, (iii) analog sums of energy deposited in the shower octants, with variable discriminator thresholds, (iv) precision tracking processors outputs which arrive some 40-60 μsecs later and is used if the fast tracker requests it. Figure 2 shows the block diagram of the CLEO trigger system.

Figure 3 shows graphically the principle of operations of the "Precision Track Processor". Each cylinder of sense wires is mapped into shift registers. These registers are shifted past several positive, and negative, curvature windows to find the track. The "Fast Track Segment Processor",uses fewer cylinders, and a single window to detect track segments.

In figure 4 the whole block diagram of CLEO data acquisition electronics circuits is shown. The 32,000 signals are packed 24 (or 60) to a card and 21 cards to a crate, making a total of 85 crates. Each crate has its own bit

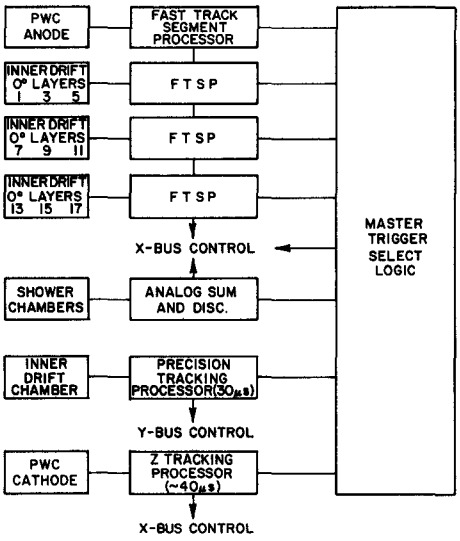


Figure 2 CLEO Trigger System

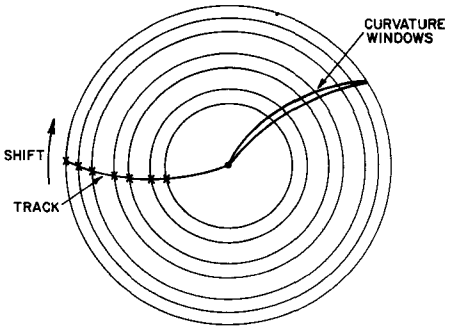


Figure 3 Precision Track Processor

slice μprocessor, ADC, DAC, control and interface logic. The digital outputs for each detector element are gain normalized and empty channel suppressed. Gains can be calibrated. The digital data come out into the safe area via a 16-bit data bus designated as the Y-BUS in the figure. An event thus consists of 2200 data words and 2200 address words, transmitted in 4-8 μsec/word through the Y-BUS driver into an PDP 11/34 which buffers and reformats the event, adding associated information which was transmitted to it through the 8-bit X-BUS (high voltage, fast trigger, monitoring, precision pulser etc.). The latter and its sources, except for its bus driver, also all reside in the radiation area. The PDP 11/34 then sends the event to the VAX 11/750 for on-line analysis, where a series of cuts are applied and dubious events are tagged before being written on tape. The 11/750 also monitors the performance of the detector and histograms bhabha and hadronic events as a function of runs. The more refined analyses and final event reconstructions are performed off-line on the Cornell DEC system-1099, and at various collaborating institutions's computers.

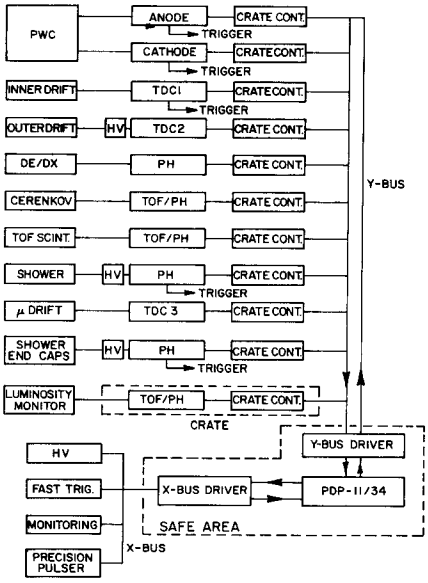


Figure 4 Block Diagram of CLEO Data Acquisition Electronics

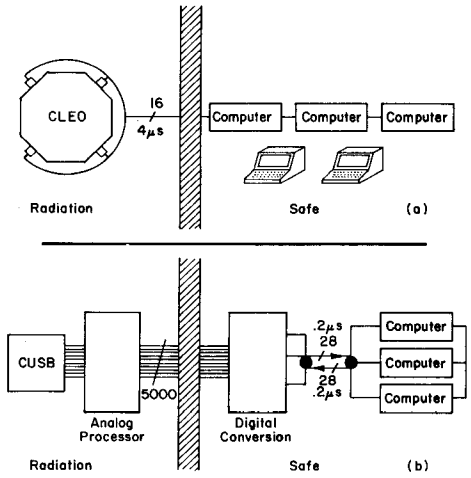


Figure 5 CLEO vs CUSB Signal Transport Comparison

3) CLEO VERSUS CUSB SIGNAL PROCESSING

Before discussing the CUSB data processing proper, I would like to make a digression on the relative merits of analog versus digital signal transport, because CLEO and CUSB each typifies one school of thought, as shown figuratively in figure 5. We have noted that CLEO's electronics are mounted directly on the detector, in a high radiation area. Only digital data is transmitted out of the radiation area, to computers some 300 feet away. This means, incidentally, that to repair any electronics failure, of even a minor sort, requires turning off of the storage ring. Of course, then there are no more analog signals. CUSB, on the other hand, chose to transmit out of the radiation area all 5,000 of its analog signals, differentially transmitted and received, into a nearby safe area where the analog signal processing electronics, the high voltage power supplies and controls, trigger module and

source calibration modules, all reside. There each analog signal is integrated and held for the analog to digital conversion process while awaiting the trigger decision. If a trigger has occurred, the digital information is transmitted at a rate of one word per 0.2μsec (along a 350' transport bus whose length was dictated by the CESR geography) to the computers after digitization.

Table 3 enumerates advantages and disadvantages of digital versus analog signal transport, based on the CESR experience.

Table 3: Digital Versus Analog Signal Transport

ADVANTAGES		DISADVANTAGES	
Digital transport	Analog transport	Digital transport	Analog transport
16 lines.			5000 lines*.
Unlimited dynamic range**.			Dynamic range < few×10^5 **
	Analog eletronics more reliable***.		
	Signals can be viewed.		
	No switching noise on low level signals.	Digitizing noise prevents simple analog event buffering.	
	Faster learning curve.		
	Trigger evolution.		

*For larger number of signals one can use 16 → 1 analog multiplexing.

**Irrelevant in practice.

***After burn-in (before installation) we ran 1.3×10^9 preamp hour without failure.

4) CUSB

CUSB is a nonmagnetic NaI-Pb glass calorimeter, supplemented with inner and interspersed tracking chambers [4]. The 5000 CUSB signals come in the following three catagories:

~350 precision, large dynamic range (10^5:1) charge signals from Central Detector Crystals.

~650 additional calorimeter signals.

~4000 time or charge signals for tracking.

All signals are digitized and transported to the computers, the empty channel suppression option is not used. We have full on-line analysis, this uses ~25% of a VAX 11/780. Approximately 70% of the triggers are saved on tape because the trigger is very efficient. It is based entirely on analog information: (i) the total energy deposited in the central detector is > 0.9 GeV, or (ii) there are three clusters whose energies are > 70 MeV AND the total energy deposited in the central detector is > 600 MeV, or (iii) there are two muons and 120 MeV deposited in the detector. Another unique feature of CUSB is that the central detector is calibrated in real time during data taking. This is performed using advanced μ-processors to track changes in NaI output and photomultiplier gains.

Figure 6 is a block diagram of the signal flow amongst the various components of the CUSB data acquisition system. The digitized output from some forty crates of electronics (containing calibration results, high voltage settings, TDC and ADC information etc.) communicate to one node (the downstairs node) of a data transport system which uses two sets of 28 dedicated unidirectional lines, 16 for data, 8 for address and 4 for control information, to transmit at the rate of 1 word/0.2 μsec to the other node (upstairs node) located some 350 feet away. To the upstairs node are attached a PDP 11/20 which communicates at 3.5 μsec per word, a dual port memory which receives at 0.8 μsec per word, a unibus adapter which links a VAX 11/780 to the PDP 11/20 (data transfer rate 5 μsec/word), and also a 286/310 μ-Processor system which communicates at 1.6 μsec per word. The dual port memory writes only from the upstairs node, and can be read and written from the PDP 11/20.

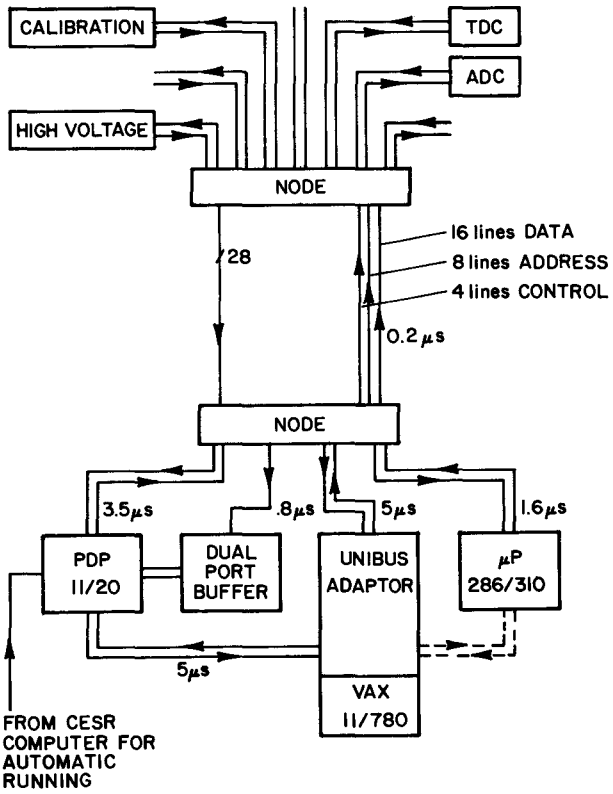


Figure 6 Block Diagram of CUSB Signal Flow

A CUSB run begins when the PDP 11/20 receives information from the CESR computers that stable running conditions have been achieved. A word is sent to a register downstairs which controls the flow of data. The first twenty events are taken with the analog signals disconnected so that we can measure the offsets of the ADC's to check for changes. Then the analog signals are reenabled. While we are waiting for a trigger to occur, the PDP 11 has initialized a buffer in the dual port memory which is large enough to contain an entire event. Whenever an entire event is assembled , the PDP 11, while reinitializing for subsequent triggers, simultaneously transfers the event at the slower rate that the VAX can handle. When the VAX has received the whole

event, it performs preliminary filtering and catagorizes the event as whether it should be written on tape or discarded, and whether it should also be saved on disks for express off line processing. Whenever the VAX sees the beginning of a new run, it also initiates the calibration process, tells the calibration μ-processor that conditions have stabilized so that the latter can independently receive its own kind of triggers and look at its own histograms. When fitted values of those histograms are available, they are transmitted back to the VAX and stored on disk and used to update the calibration constants. The calibration data is collected for 16 crystals at a time and 10 minutes are necessary to collect the data from the 352 crystals of the central detector, corresponding to 12×10^6 words. The calibration system digests 20,000 words/sec in the transfer mode. One CUSB event has 5000 words, which takes 4 ms to transport. At an event rate of about 0.5 Hz, this introduces 0.002 sec/sec of dead time.

5) CUSB CALIBRATION SYSTEM

One of the key to the success of the CUSB calorimeter is its real time calibration using radioactive sources imbedded between layers of the NaI crystals. Figure 7 shows the data path for one channel. Note that the PM signal is sent to two independent ADC's, one of which is 5 MeV full scale to accomodate source signals (as shown at the bottom of the figure from Cs^{137}) and the other has full scale at 2 GeV , typical of the amount of energy deposition in the crystals by real data (for ex. 5 GeV Bhabhas). We have performed these calibrations using an independent PDP 11/20, it takes 3 hours for 352 channels. The VAX 11/780, as we have mentioned, has to do on line analysis and off line computing. Thus it can not perform calibration more frequently than once every 90 minutes without severely degrading response time.

We have therefore implemented a 286/310 μ computer system, whose block diagram is shown in figure 8, specifically dedicated to perform the real time crystal calibration. It takes 19 sec to receive and histogram the incoming 12×10^6 words, 44 sec for fitting the histograms with a polynomial plus a gaussian to obtain peak positions to 0.05%, and to monitor backgrounds. The

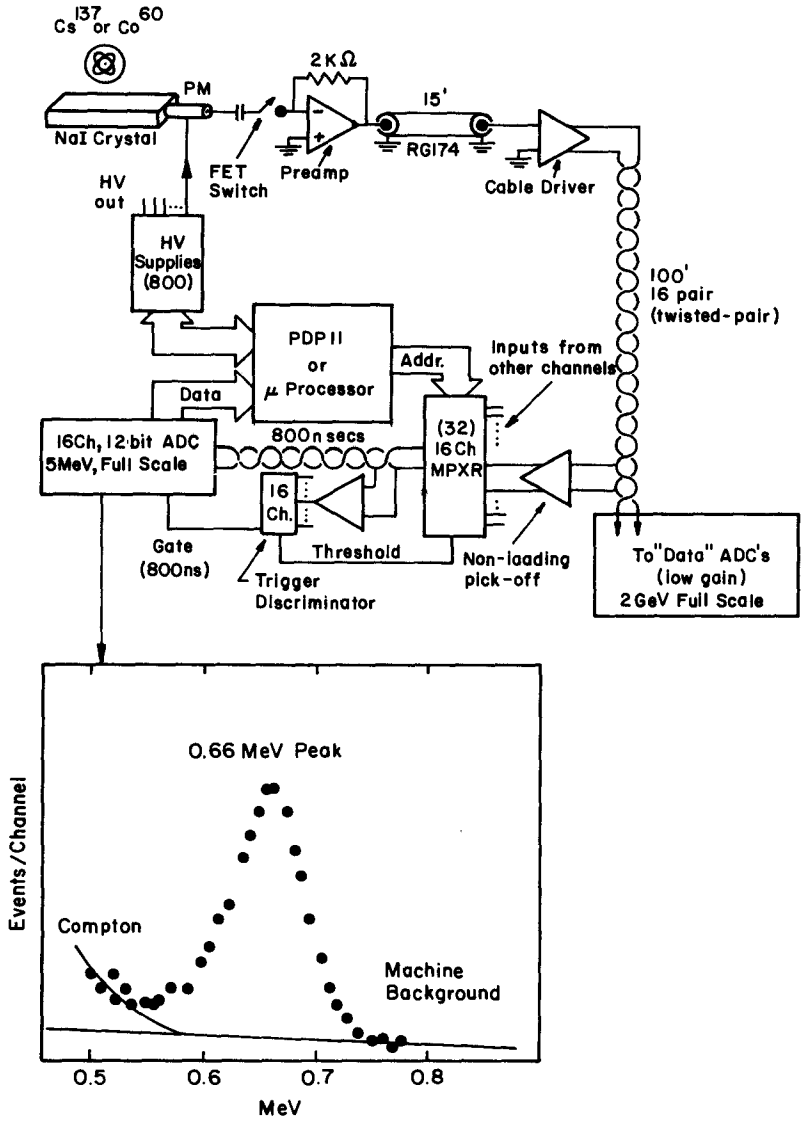


Figure 7 CUSB Dual ADC System For Real Time Calibration
(below) Source Signal During Data Taking

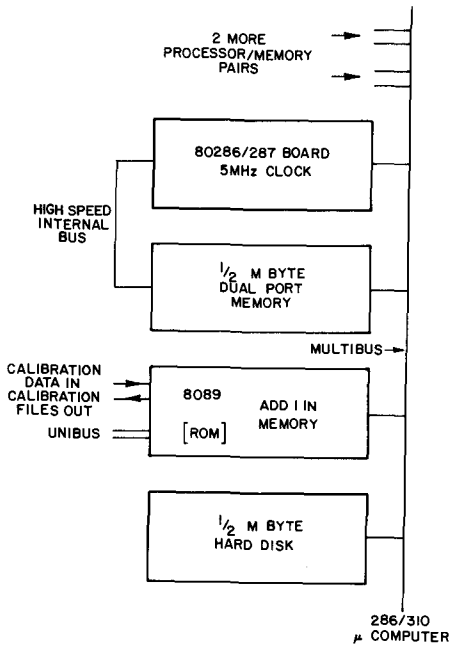


Figure 8 CUSB Real Time Calibration System

required effort to make this change over was small. It took less than one day to transfer the operations to the μ-P. The FORTRAN programs were compiled in the 286/310 using the Intel compiler and operating system. We could also have generated the code using the VAX, but it was simpler to debug this way.

6) CESR PLANS FOR AUGMENTING COMPUTING CAPACITIES

CESR is embarking on a program to increase its machine luminosity by another factor of four. The first factor of two comes from having 7 rather than 3 bunches in the machine, resulting in a crossing rate of 2.8 MHz. The second factor of two is expected from μ-β insertions. Therefore, the peak data rate will be about 4 Hz, with ~ 360 nsec between crossings.

The detectors meanwhile are also being upgraded, CLEO by adding a vertex chamber, implementing dE/dx in their inner drift chamber and eventually replacing their drift chamber, coil and implementing a CsI electromagnetic calorimeter (in ~ 1988). Their computing needs will go up by at least and order of magnitude. CUSB expects to install a Bismuth Germanate (BGO) calorimeter composing of 360 crystals within its NaI array. This combination of more events, more elaborate analysis and more calibration implies at least a four fold increase in computing needs.

Both groups aim to solve part of their computing demands via emulators or micro computers. CLEO has several 370E's under construction. Because of the considerable efforts required to make such devices useful, plans at present are to switch soon to single board μ-computers. CUSB expects that 2 or 3 additional 80286/80287 single board computers can handle the immediate burden and hopes to switch to 32 bit single board micros when they become available.

ACKNOWLEDGEMENTS

The author thanks C. Bebek, D.L. Kreinick, R.D. Schamberger for discussions and J. Lee-Franzini for help during the preparation of the present paper. The author thanks C. Avilez, T. Nash, and R. Donaldson for their hospitality at this interesting conference. The present work is supported in part by the U. S. A. National Science Foundation.

REFERENCES

1. K. Han et al., Phys. Rev. Lett. 49 1612 (1982), G. Eigen et al., Ibid 1616 (1982); C. Klopfenstein et al., Phys. Rev. Lett. 51, 160, (1983); F. Pauss et al., Phys. Lett. 130B, 439 (1983). For a general review of T physics at CESR see: P. Franzini and J. Lee-Franzini, Ann. Rev. Part. Sci 33, 1 (1983).
2. S. Behrends et al., Phys. Rev. Lett. 50, 881 (1983).
3. D. Andrews et al., Nucl. Inst. and Meth. 211, 47 (1983).
4. P. Franzini and J. Lee-Franzini, Phys. Rep. 81, 240 (1982).

QUESTIONS AND ANSWERS

Q: Elaborate on use of the 370/E emulators for CLEO?

A. Brenner

A: Two processors are operative for integers under test. There are difficulties in FORTRAN conversion from DEC to IBM.

Q: What is CLEO's experience with $370/_E$?

A. Brenner

A: No experience running $370/_E$ yet. Two processors are running integer code. Main problem is converting our programs running on DEC machines to run on IBM.

TRIGGER AND DATA-ACQUISITION PLANS FOR THE LEP EXPERIMENTS

Wolfgang von Rüden
CERN, Geneva, Switzerland

Abstract

Four experiments have been accepted for phase 1 of LEP: ALEPH, OPAL, L3, and DELPHI. These experiments have each of the order of 100,000 or more electronic channels producing more than 100 Mbytes of raw data per second. Extensive zero-suppression, data reduction and formating as well as intelligent triggers are needed to reduce the amount of data to be written to tape to 0.1-1 Mbyte per second.

As the various detector components are developed in laboratories distributed all over Europe, in the US and in China and Japan, good communications are essential during the development phase and later for distributed data analysis.

We give a brief overview on the LEP accelerator and introduce the four detectors. We discuss the trigger and the data acquisition for the four experiments and summarize the different techniques used.

1. HISTORY

During the years 1976 to 1979 preliminary discussions started in the European High-Energy Physics community about a follow-up programme for the CERN Super Proton Synchrotron (SPS). Four alternatives were proposed, namely a 10 TeV Proton Synchrotron, a 400 x 400 GeV pp collider, an ep collider, and a large e^+e^- ring. The last option met with the greatest interest and the project was named LEP, the Large Electron-Positron Collider.

In 1979 preliminary studies started in Les Houches, followed by the Uppsala conference in 1980 and the conference in Villars in 1981. The CERN Member States approved the project unanimously in December 1981. At that time collaborations had already been formed, and in January 1982 seven Letters of Intent were submitted to the LEP Committee (LEPC) whose task was to select four experiments. The first session took place in March 1982, where questions arose concerning the feasibility of the projects. Referees were appointed, and in summer 1982 the following experiments were accepted:

L1 ALEPH
L2 OPAL
L3
L4 DELPHI.

All four collaborations submitted detailed Technical Reports in April 1983. In addition, special reports on trigger and data handling were sent to the LEPC in January 1984.

The present talk is based on the Technical Reports [1-4], the reports on data handling [5-8] and on private communications from the collaborations. The reader should keep in mind that we present the plans of the experiments and that some of the information contained in this presentation will be obsolete by the time this report is published.

2. THE LEP MACHINE

As the topic of the present talk is data acquisition, only a very small section can be devoted to the superb LEP accelerator project. Fig. 1 shows an aerial view of the Geneva area with the airport at the bottom of the picture and the Jura mountains in the back. The present CERN accelerators, the PS, the ISR and the SPS, can be seen as well as the planned implementation of LEP, which has a circumference of nearly 27 km (!). The locations of the eight access points are marked in fig. 2. Only the four even-numbered interaction regions will be equipped with experiments during the first phase of LEP; pit 2 has been assigned to the L3 collaboration, pit 6 to OPAL, whilst the assignment for ALEPH and DELPHI is still under discussion.

Figure 3 shows a vertical cut with the Jura on the left-hand side and the lake of Geneva on the right. Most of the LEP tunnel will be in the "molasse", a convenient material for tunnelling. Only a small part, which is under the Jura, consists of limestone and may therefore give some unexpected problems. The ring is inclined by 1.4% and the depth varies between 50 m and 150 m.

The layout of an interaction region is shown in fig. 4. The "straight" pipe will receive the accelerator. The tunnel on the left is for machine equipment, mainly the klystrons for the RF. The experimental hall has a diameter of about 15 m and a length of 70 m. Three access shafts are provided: the leftmost for machine access, the large one in the centre to bring the experiment down, and the small shaft on the right will be used by the experimentalists. Two 40-ton cranes will be installed in the experimental hall to assemble the detector. The detectors (except L3) will be movable from the beam position to the "garage" position for maintenance work. The surface buildings provide storage room for equipment, cranes to bring it down and also space for control rooms, workshops, some offices, etc. Some of the buildings are devoted to machine equipment.

The interaction region number 2, for the L3 collaboration, differs from the others as it is oriented parallel to the machine tunnel. Also, the L3 detector has too much weight to be moved, so it will be assembled directly into its final position.

Table 1 gives a few machine parameters. The particles will make 1 turn every 88 μs and as there are 4 bunches, the beam crossing will occur every 22 μs at each interaction point. According to the present planning, and if everything goes well, the beam for the experiments is expected for the end of 1988.

3. THE FOUR LEP DETECTORS

We give a short description at the four detectors to make the understanding of the trigger and data-acquisition easier. For details, the interested reader should refer to the Technical Reports issued by the collaborations [1-4].

The scenario for the detector construction is very similar for the four collaborations. The various pieces of the detectors will be constructed in institutes spread all over Europe and even to the United States, China and Japan. Therefore, all experiments are confronted with communications problems, which are in particular delicate for the on-line system.

3.1 The ALEPH detector

The ALEPH collaboration, spokesman J. Steinberger, is formed by 25 institutes from 8 countries and counts today some 330 physicists and engineers. The detector [1] is shown in fig. 5. Looking from the beam-pipe outwards, it is composed of an inner trigger chamber (4), the time projection chamber (5), the electromagnetic calorimeter (6), the superconducting coil (7), the hadron calorimeter (8) and finally the muon detector (9). A luminosity monitor (3) surrounds the beam-pipe at each end of the TPC.

3.2 The OPAL detector

The OPAL collaboration, spokesman A. Michelini, is formed by 180 physicists and engineers, coming from 19 institutes in 9 countries. The detector [2] design is more conservative, and the components are based on known techniques, of course on a much larger scale than built up to now.

Starting from the beam-pipe (fig. 6) we recognize the central detector (vertex chambers and JET chamber), surrounded by the solenoid, followed by time-of-flight counters, the electromagnetic and the hadron calorimeters, and the muon chambers.

The simulation of a two-jet event in this detector is shown in figs. 7 and 8 [9].

3.3 The L3 detector

This collaboration, spokesman S.C.C. Ting, is made up of 36 institutes from 12 countries and counts about 360 physicists and engineers. The arrangement of the L3 detector [3] in its experimental hall is shown in fig. 9. From the inside to the outside, we find first a time expansion chamber, followed by electromagnetic shower counters made up of 12,000 BGO crystals. Thereafter come the hadron calorimeter and the muon drift chambers. All these components are surrounded by a very large magnet.

3.4 The DELPHI detector

There are 34 institutes from 17 countries, with 280 physicists and engineers, forming the DELPHI collaboration, spokesman U. Amaldi. The DELPHI detector [4] has the largest variety of detector elements using different techniques. Starting from the beam-pipe (fig. 10), there are a vertex chamber, an inner detector, a time projection chamber, the barrel ring imaging Cerenkov (RICH), the outer detector and the barrel electromagnetic calorimeter. These components are surrounded by a superconducting coil, after which follow a scintillator hodoscope, the hadron calorimeter, and two layers of muon chambers. Going from the centre to the end-plates, we find after the TPC a set of three forward chambers (A,B,C) interspersed by a liquid RICH and a gas RICH, then the electromagnetic calorimeter and finally the end cap hadron calorimeter.

4. THE TRIGGER SYSTEMS

Because of the different properties of the four detectors, the trigger designs also vary in complexity, and response time. The information given here is based on the reports on trigger and data acquisition of January 1984 [5-8]. For more recent information, the on-line coordinators can be contacted:

ALEPH	W. von Rüden, CERN,
OPAL	H. von der Schmitt, Bonn and CERN,
L3	M. Fukushima, DESY,
DELPHI	J. Allaby, CERN.

In the following, we present the basic principles of the different triggers. For the timing and rates, Table 2 provides a comparison of the four detectors.

4.1 The ALEPH trigger

The aim is to trigger on all physics events, whilst rejecting background as much as possible. Three independent conditions have been defined:

1. $\geq$ 2 minimum ionizing tracks;
2. $\geq$ 1 minimum ionizing track and one energy cluster (low threshold);
3. total electromagnetic or hadronic energy above a (high) threshold.

The trigger is performed at three levels: level 1 must respond within 1.5 μs (including cable delays) of bunch crossing, to open the gate of the TPC. This time corresponds to ~ 7.5 cm drift space being lost at the end of the tracks, for those passing through the end-plates. The detector components contributing to this level of triggering are:

- the inner trigger chamber,
- the electromagnetic calorimeter,
- the hadron calorimeter,
- the muon chamber,
- the luminosity monitor.

Figure 11 shows a block diagram of the ALEPH level-1 trigger logic. The 72 segments for the calorimeters are formed by grouping towers of the calorimeter into "super" towers.

After the drift time of the TPC (~ 40 μs), there is additional information about the tracks in the TPC pointing towards the vertex. This information is obtained by a special hardware processor during the TPC drift time using special trigger pads; therefore, no readout is needed to process this information, which is used to generate the second-level trigger. If the decision is positive the readout of the various sub-detector components is started.

The last stage is the event processor, a computer capable of running the reconstruction programme, which will see for the first time a complete event. Its task would be to reject further background events and/or clarify events for the off-line analysis.

A trigger supervisor will assure the proper gating and distribution of the trigger signals to the different sub-detectors; it will also be used to perform consistency checks and to guarantee the proper response of all needed components.

4.2 The OPAL trigger

The OPAL detector will trigger on

1. hadronic jets,
2. charged-lepton pairs,
3. radiative production of $Z^0 \rightarrow \nu$,

and also on two-photon processes and free quarks. As the decision time can be as long as 18 μs (4 are needed to clear for the next bunch crossing), a very sophisticated trigger, including all detector components, can be made. Indeed, information comes from

1. the electromagnetic and hadron calorimeters: energy and topological information;
2. the tracking chambers: vertex, jet and z-chambers;
3. the muon chambers;
4. the forward detector.

This trigger is supposed to bring the 50 kHz bunch crossing rate down to $\leq$ 4 Hz!

4.3 The L3 trigger

For the L3 detector, six different triggers are planned with the following properties:

1. Muon trigger: $\geq$ 1 track pointing to the vertex, $p_T \geq 2$ GeV/c.
2. Energy trigger: the sum of the electromagnetic shower counters (BGO) and of the hadron calorimeter exceeds 20% of the

centre-of-mass energy.

3. Charged-particle trigger: ≥ 2 tracks pointing to the vertex, $p_T \geq 0.3$ GeV/c. If this trigger is too loose, ask for some energy deposit in the calorimeters.
4. Single-photon trigger: defined by an isolated energy deposit in the BGO crystals.
5. Small-angle trigger: requires some energy deposit in at least one side of the small angle calorimefer and a charged track or energy in the central calorimeter.
6. Luminosity trigger: energy deposit in both sides of the detector.

Figure 12 gives details of the level-1 and level-2 triggers. Note, that 4,500 input signals are used to construct the trigger, which might give an idea of the complexity of such a device. The timing and the rates for the different stages are represented in fig. 13.

4.4 The DELPHI trigger

The DELPHI trigger system is based on four levels. For the first level (~ 2 μs) the requirements are

1. ≥ 1 charged track of ≥ 2 GeV/c momentum.
2. An energy deposit in the calorimeters for neutral events.
3. Two coincident track elements from the inner detector, outer detector, TPC end-plates and forward chambers (depending on the detector region).

If these requirements are too loose (rate above 1 kHz), information from the electromagnetic calorimeter, the time-of-flight counters, or the towers of the hadron calorimeter can be used in addition.

At the second level (~ 35 μs) more detector components contribute to the trigger decision:

1. the TPC: - a track points to the vertex;
   - cut on p_T;
2. the HPC: correlate tracks and energy clusters;
3. the muon chambers;
3. the electromagnetic and the hadron calorimeter for neutral hadrons.

Level 3 requires information from the readout system. Each detector will have a microprocessor (GPM [10]) to perform a selective readout. The combined information will be transmitted to the main level-3 processor, likely to be a XOP [11].

Finally, level 4 will use emulators for further filtering of events. Figure 14 gives an overview of the DELPHI trigger system.

4.5 Trigger summary

As each collaboration applies different terms for similar items, and as the reports on trigger and data acquisition are organized in different ways, a

comparison is difficult. Table 2 is an attempt to bring the information together.

The different "levels" defined there do not necessarily coincide with the naming found in the reports: Level 1 includes actions taking place between beam crossings (22 μs). There is a large difference in the expected rates between OPAL (4 Hz) and DELPHI (1000 Hz). Slower detectors such as a TPC will contribute to decisions in level 2. Here the rates of the four detectors are about the same (4-20 Hz). Level 3 (if used) reduces the rates by another order of magnitude.

Finally, the event processors will really see the "full" event and might take decisions based on sophisticated algorithms. In general, emulators or banks of microprocessors are supposed to deliver massive CPU power for this job, at a reasonable cost. The table also shows the expected event size and deduces from that the number of 6250 bpi tapes written per day. Note that for an expected running time of 100 days, each collaboration will produce about 10,000 tapes per year!

5. THE READOUT SYSTEMS

Despite similar requirements, the readout systems of the four detectors have adopted quite different solutions. ALEPH, L3, and DELPHI have chosen FASTBUS as their data-acquisition standard; OPAL will use FASTBUS only at the level of the front-end electronics and base the readout and the processor modules on the VME/VMX bus standard. Using FASTBUS does not mean that the other three experiments would look alike. The flexibility of FASTBUS allows the user to dream up virtually any configuration.

The location and interconnection of the mini-computers is also different for the four experiments, but at least all have decided to use the VAX family of computers running VMS. Ethernet seems to emerge as the preferred solution for the local area network (LAN).

5.1 The ALEPH readout

Figure 15 shows a typical ALEPH sub-detector arrangement during the development phase. The equipment or sub-detector computer is used to develop and understand the detector and to prepare the tools for the monitoring and calibration. The requirements vary from one sub-detector to another; for example, the TPC is the most demanding part, producing ~ 80% of the data per event. Its electronics will fill more than 100 FASTBUS crates. The connection via the LAN to the international network allows close contact between the sub-detectors. In fact, such a sub-detector set-up corresponds in its complexity to today's large experiments.

The connection to the central readout can be seen from fig. 16. During normal data taking the readout is controlled centrally, while the "local" VAX can "spy" on events from its sub-detector. The function of the event builder (EB) is to collect the pieces of an event from the different readout controllers, which are the masters in the front-end crates, and to store the combined information in its memory. Whenever a sub-detector computer needs

data, it might request a copy of the next event from the EB. The main data stream will not be affected by the speed at which a local computer can absorb data.

The overall layout (preliminary) is sketched in fig. 17. The sub-detector computers and the main on-line computer are located in the surface building, at ~ 150 m from the detector. Optical-fibre links are required for the long-distance FASTBUS connection. This solution has the advantage that all computers can be connected via Digital's "Computer Interconnect CI", a high-speed serial link supporting shared disks and tapes via the "hierarchical storage controller HSC50". Also, this link will serve as a fast communication channel between the sub-detector computers and between these and the large VAX. The arrangement in one room gives the additional benefit of a clean "computer floor" and eases the maintenance.

The Ethernet serves to connect terminals via concentrators, to provide a printer/plotter station, and a link to the "slow control" equipment (see later). It will extend down to the counting rooms, where terminals and printer are needed near the electronics. It should also provide the link to the CERN-wide network and through it the access to the international connections.

5.2 The OPAL readout

Figure 18 shows the latest version of the OPAL VME/VMX readout system and the connection to CAMAC. As this information was received just before the Symposium, the reader is asked to contact the OPAL collaboration for more information.

5.3 The L3 readout

Depending on the particular sub-detector, the front-end electronics is placed either on the detector itself and interfaced to FASTBUS, or it is directly built in FASTBUS. Taking the muon chambers as an example, L3 will use the LeCroy 1800 system with the 96-channel TDCs (model 1879) and the readout controller 1821. The overall control is done by a VAX 11/780. For the hadron calorimeter, the ADCs are mounted directly on the detector and a private databus will end in a multiplexor interfaced to FASTBUS.

The BGO readout deserves special attention, as each of the 12,000 crystals will have its own microcomputer (M146805) for readout and control. They are connected in 150 groups of 80, controlled by 150 micros; these are, in turn, organized as 8 groups of about 20. The 8 controllers can be either CAMAC or FASTBUS modules, controlled by a machine with a CPU power equivalent to a small VAX.

Without going into the details of every sub-detector, it can be said that L3 will have, inside each sub-detector, several parallel data streams merging into a buffer memory, managed by an EB (fig. 19). The assembled sub-events are passed through a central EB to the emulators memory; from there they can be transferred (after treatment) to the host computer (VAX 11/790) and/or to an extra emulator for full event reconstruction; they are then further

analysed by another VAX 11/790.

5.4 The DELPHI readout

DELPHI stresses the modularity of the data-acquisition system. Each major sub-detector component will have its own "equipment computer"; all these are interconnected via a LAN (fig. 20). There will be a "OPS" computer, a "large general-purpose computer facility serving the whole experiment", and a "DAC" computer whose task is the transfer of data from the main data acquisition to the tape. Extra computing power for both machines is foreseen in the form of emulators on a separate dedicated LAN. The general LAN will provide the connections to the CERN-wide and European-wide networks.

5.5 Network connections

Good communications between the various home institutes and CERN are vital for all LEP collaborations. At the moment, the network is growing all the time. Most of the machines at CERN are already connected via DECnet, which extends, via the INFN gateway, to Italy, where DECnet has been in use for several years already. For France, Switzerland and Germany, public X.25 networks are proposed. Great Britain has its own scientific network JANET, based on the "coloured book" software and X.25. The network extends to CERN and it might even be possible to run DECnet over JANET to have a uniform connection. The GIFT project at CERN is supposed to provide gateways between incompatible networks.

In the final layout there will be several levels of networking: local connections at the experiment, regional on the CERN site, and the European or world-wide. The example in fig. 21 shows how DELPHI sees the networking, which is representative of the other experiments systems.

6. SLOW CONTROL

This includes monitoring of voltages, currents, temperatures, regulation of power supplies, control of gas systems, monitoring of interlocks, etc. The intention is to use the equipment developed by the LEP control group for the accelerator. It is based on the M6809 microprocessor, the form-factor is a single-height Eurocard using the G64 bus. FLEX has been selected as the operating system. To connect multiple crates, two types of LAN are envisaged: UTI-net [12] or the MIL-1553 standard.

For more demanding tasks, VME-based M68000-type systems are likely to be used. The connection to the main data-acquisition system could go via VME-based Ethernet controllers or via direct interfaces to the VAX.

7. SUMMARY OF PROCESSORS

For the first- and second-level triggers, special hardware machines are proposed, based on look-up tables, coincidence matrices and similar. For the higher-level triggers, some experiments propose to use fast bit-slice

processors such as XOP [11] or M68000-type machines in the FASTBUS standard.

For data reduction and readout at the crate level, the M68000 is the favourite, either in FASTBUS or in VME (OPAL). L3 has a special solution for the BGO readout using one CMOS microprocessor per channel (12,000 units).

At the level of the event processor or event filter three solutions are competing: emulators such as the 370E [13] or the 3081E [14], arrays of microprocessors (Fermilab ACP project) [15] or a set of microVAXs. The last one is very attractive because of software compatibility with the general system. However, the CPU power per dollar might exclude this solution.

All four LEP experiments have decided to use VAX computers as the minis for the data acquisition and monitoring.

For slow control applications the plans are to use the M6809 based on G64 or the M68000 in VME.

Last but not least, work-stations will be needed for on-line and off-line analyses and high-resolution graphics. Apollo is a candidate; LISA II is being evaluated by DELPHI; and there is the hope that DEC might show up with a competitive offer based on a microVAX running microVMS.

8. CONCLUSION

There is much activity at CERN and in the participating home laboratories. We have to find solutions for our communications problems which are caused by the distributed development. A lot of work still needs to be done, in particular in FASTBUS, VME, and for the emulators. The software problem has not been addressed at all in this presentation, which does not mean that it is solved! In fact, the opinions there are as divergent as the hardware solutions.

Fortunately, there are more than four years left, so there should be time to get the work done.

Acknowledgements

I would like to thank all the members of the four LEP collaborations as well as the people from the LEP machine, who provided the necessary input for this talk and helped with drawings and transparencies.

References

[1] ALEPH Collaboration, Technical Report, CERN/LEPC/83-2, May 1983.

[2] The OPAL detector, Technical Proposal, CERN/LEPC/83-4, May 1983.

[3] Technical Proposal L3 Collaboration, CERN/LEPC/83- , May 1983.

[4] DELPHI, Technical Proposal, CERN/LEPC/83-3, May 1983.

[5] ALEPH Collaboration, Data acquisition and data analysis, CERN/LEPC/84-8 (1984).

[6] DELPHI Collaboration, Data acquisition and analysis in the DELPHI Experiment, CERN/LEPC/84-3 (1984).

[7] L3 Collaboration, Trigger and data acquisition system of L3, CERN/LEPC/84-5 (1984).

[8] OPAL Collaboration, Report on data acquisition and data analysis, CERN/LEPC/84-4 and 84-2 (1984).

[9] R. Hemingway, private communication.

[10] H. Müller, The GPM, a general purpose programmable FASTBUS master (F6808), CERN, EP Division, 20 May 1984.

[11] T. Lingjaerde, XOP, DELPHI 82/45 ELEC; A second generation fast processor for on-line use in high energy physics experiments, Proc. Topical Conf. on the Application of Microprocessors to High-Energy Physics Experiments, Geneva, 1981 (CERN 81-07, Geneva, 1981), p. 454.

[12] A.K. Barlow et al., The UTI-NET book, CERN/SPS/ACC/Tech. Note 84-1 (1984).

[13] H. Brafman et al., A fast general purpose IBM hardware emulator, Proc. of the Three Days In-Depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padua, 1983, (University, Padua, 1983) p. 71.

[14] P.F. Kunz et al., The 3081/E processor, same Proc. as Ref. [13], p. 84.

[15] I. Gaines, The Fermilab Advanced Computer Program.
H. Areti et al., ACP Modular Processing System: Design specifications.
T. Nash, private communication.

Table 1

GENERAL LEP PARAMETERS

Machine circumference	26.658,879 m
Average machine radius	4,243 km
Minimum machine radius	4,204 km
Maximum machine radius	4,263 km
Bending radius	3,104 km
Number of intersections	8
Number of bunches per beam	4
Horizontal betatron wave number	90,35
Vertical betatron wave number	94,20
Momentum compaction factor	$1{,}928 \times 10^{-4}$
Harmonic number	31.320
RF frequency	352,21 MHz

Table 2

TRIGGER SUMMARY AND RATES

Beam crossing: 22 μs

TRIGGER	ALEPH	OPAL	L3	DELPHI	Unit
Level 1	1.5	18+4	10	2	μs
	≦ 500	< 4	~ 100	1000	Hz
Level 2	50	N/A	100	35	μs
	≦ 10		~ 10	≦ 20	Hz
Level 3	N/A	10	5	15	ms
		~ 0.4	10	5	Hz
Event processor	~ 2	?	a few	≦ 2	Hz
No. of channels	~ 300,000	86,000	≧ 50,000	~ 150,000	
Event size *)	~ 100,000	140,000	70,000	200,000	bytes
Time per tape	15	30-60	30	4	min
No. of tapes	100	50-25	35-50	≦ 400 !	per day

Expected running time: 100 days/year

* 20 tracks and 20 photons.

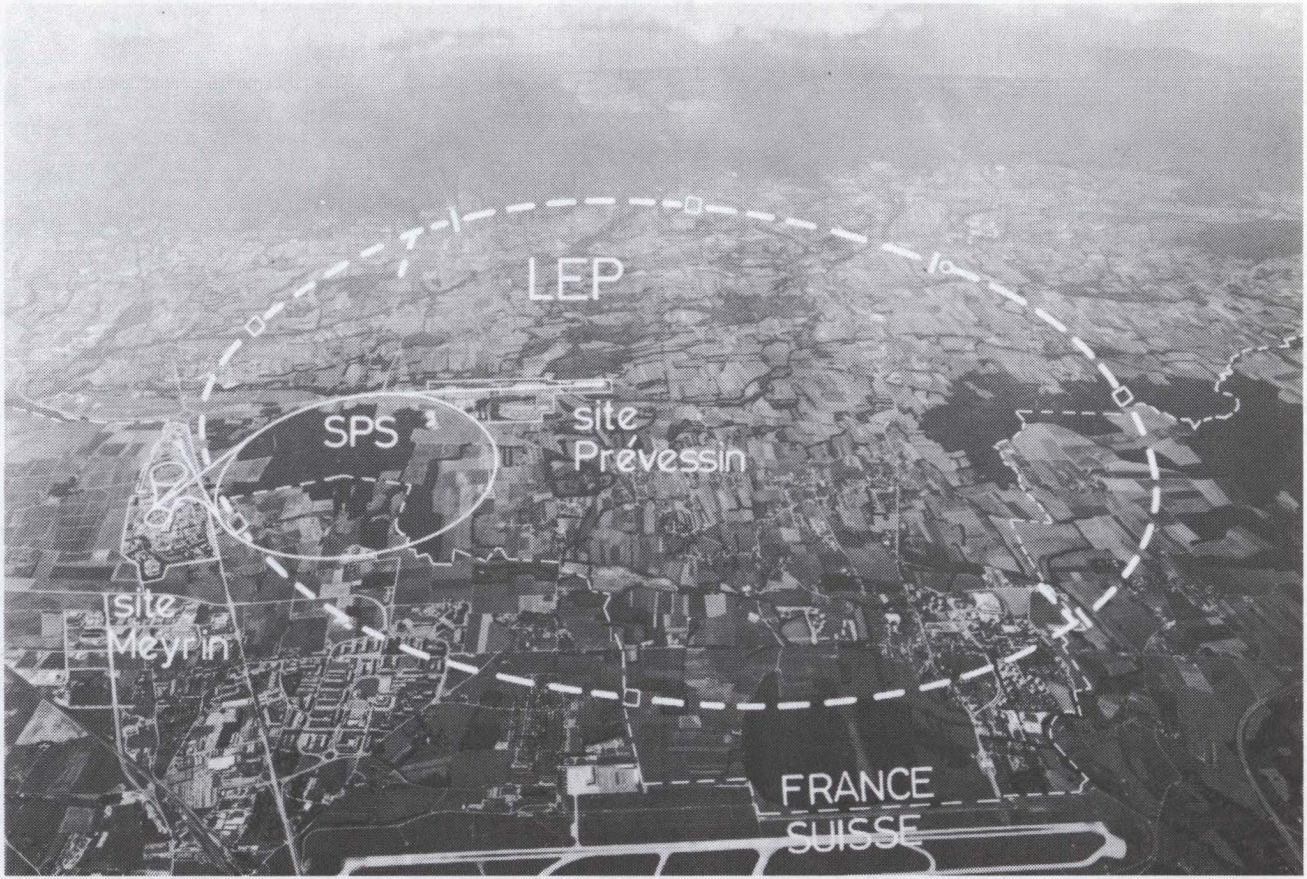


Fig. 1 Aerial view of the LEP site

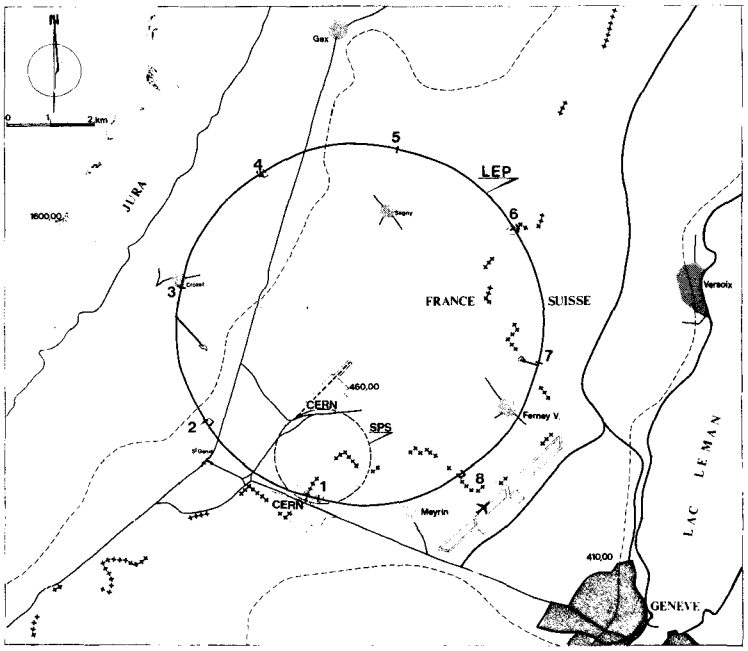


Fig. 2 Location of the LEP access points

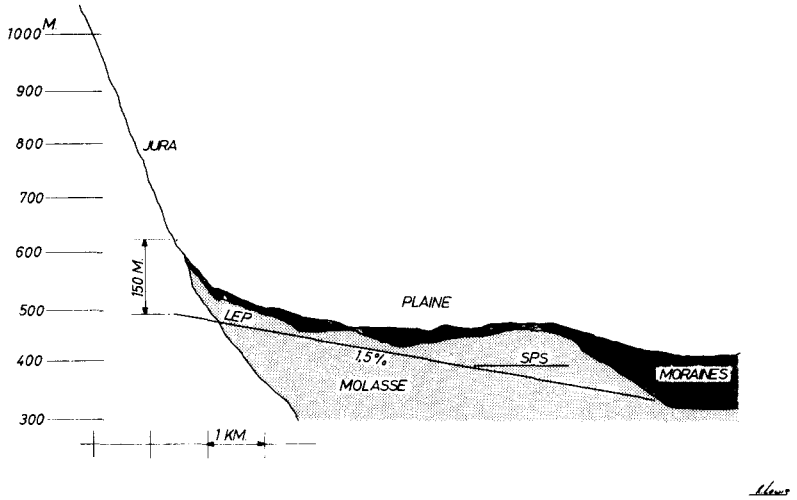


Fig. 3 Simplified geological section

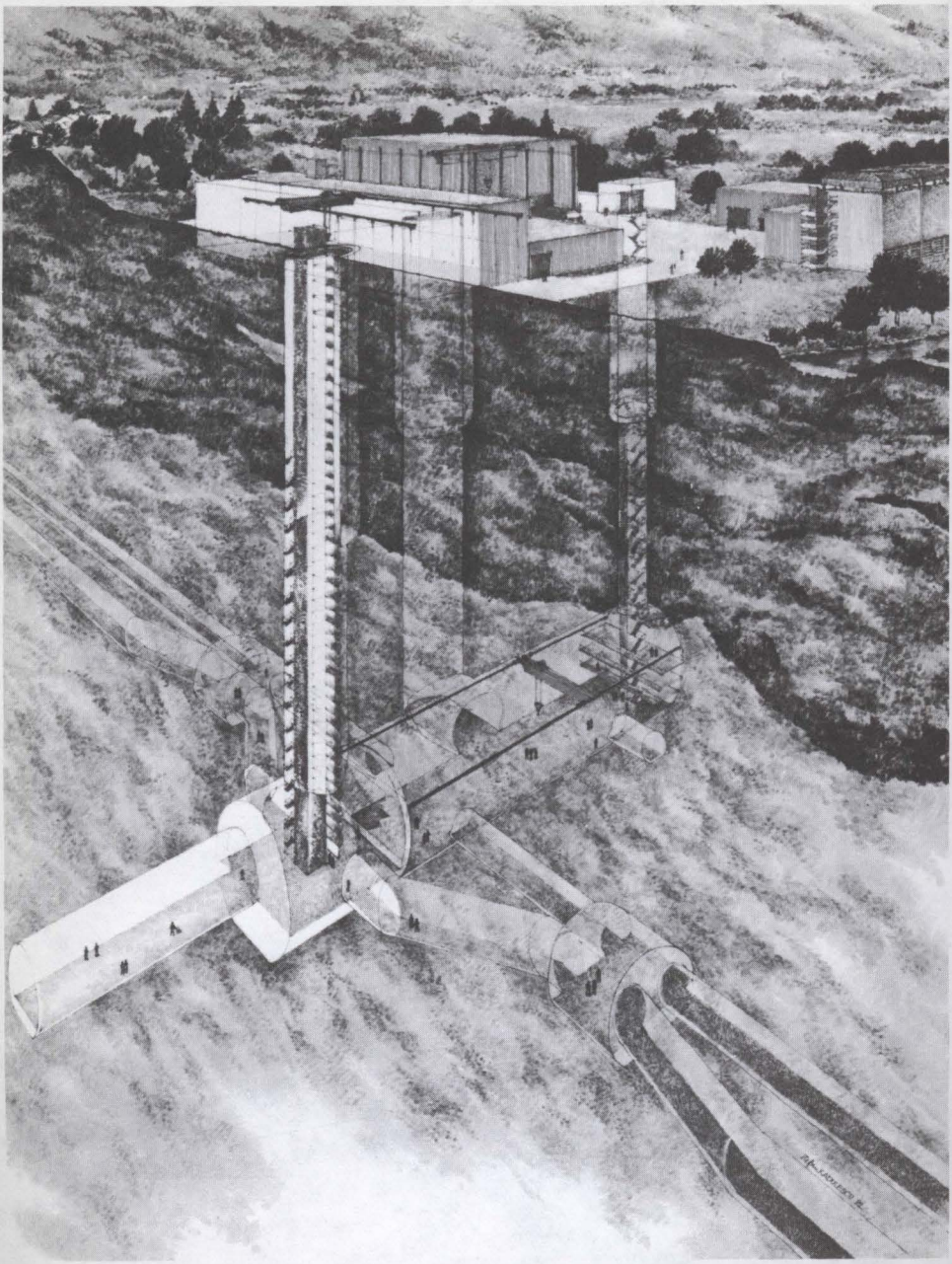


Fig. 4 Layout of an interaction region

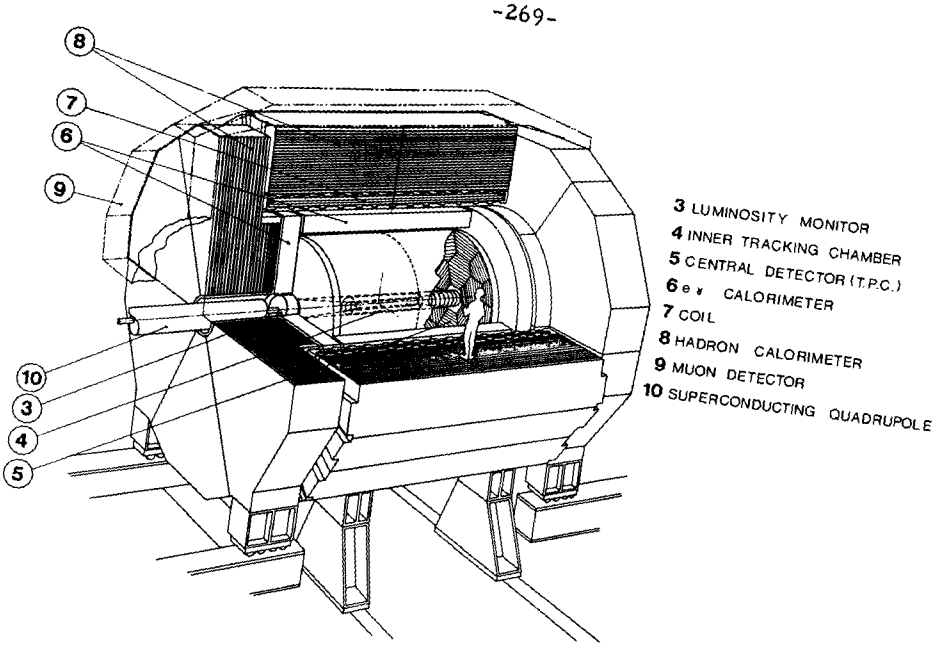


Fig. 5 The ALEPH detector

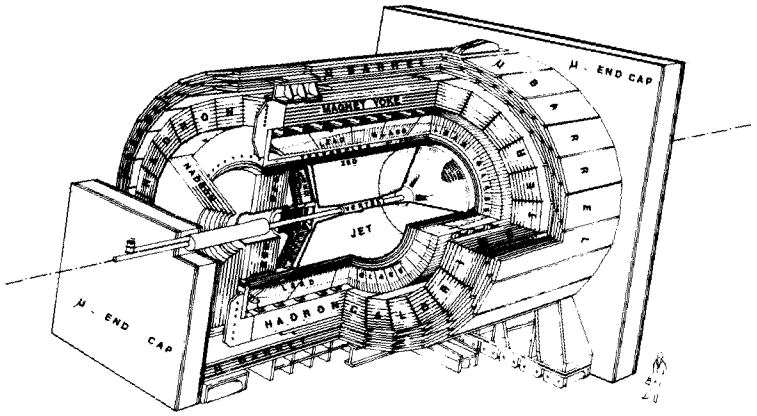


Fig. 6 The OPAL detector

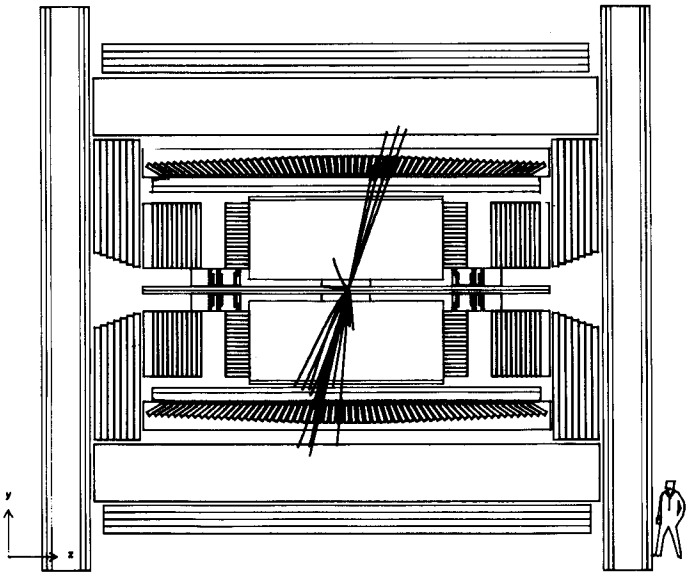


Fig. 7 A two-jet event in the OPAL detector (side view)

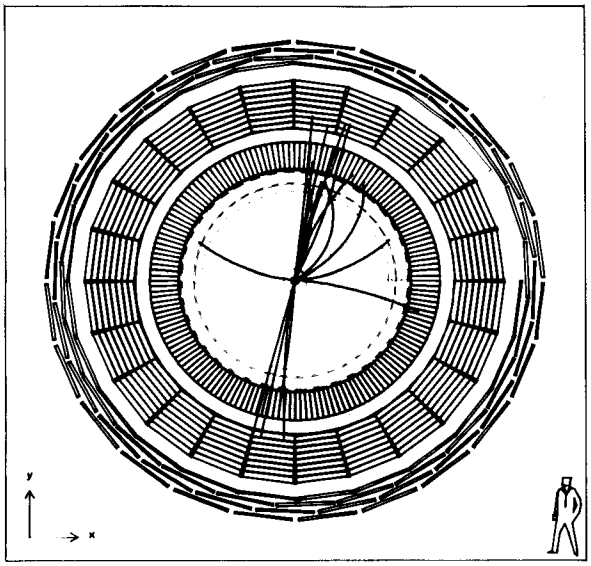


Fig. 8 Same event as in Fig. 7 (front view)

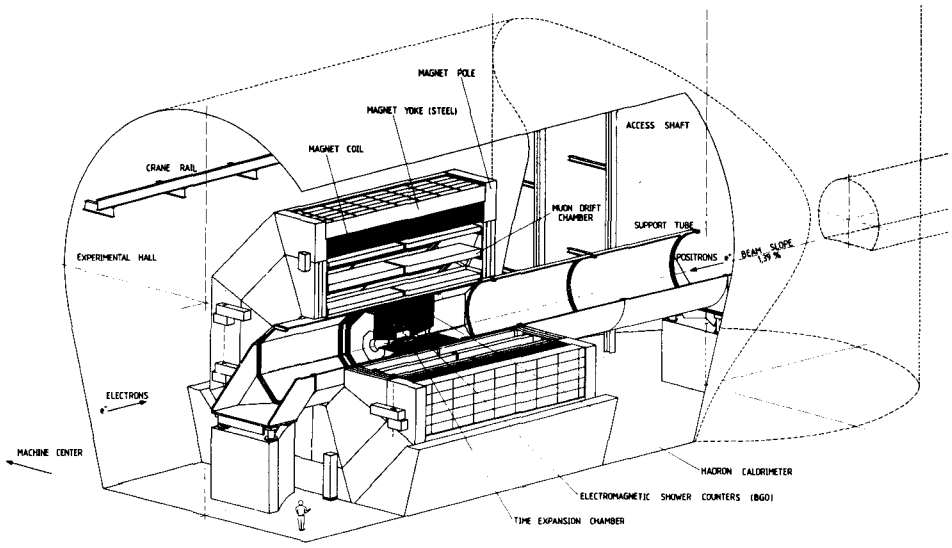


Fig. 9 The L3 detector

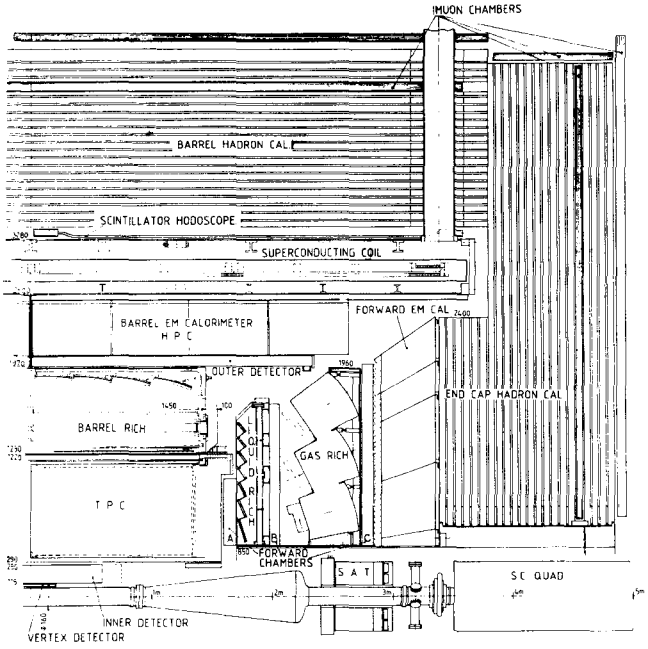


Fig. 10 Longitudinal view of the DELPHI detector

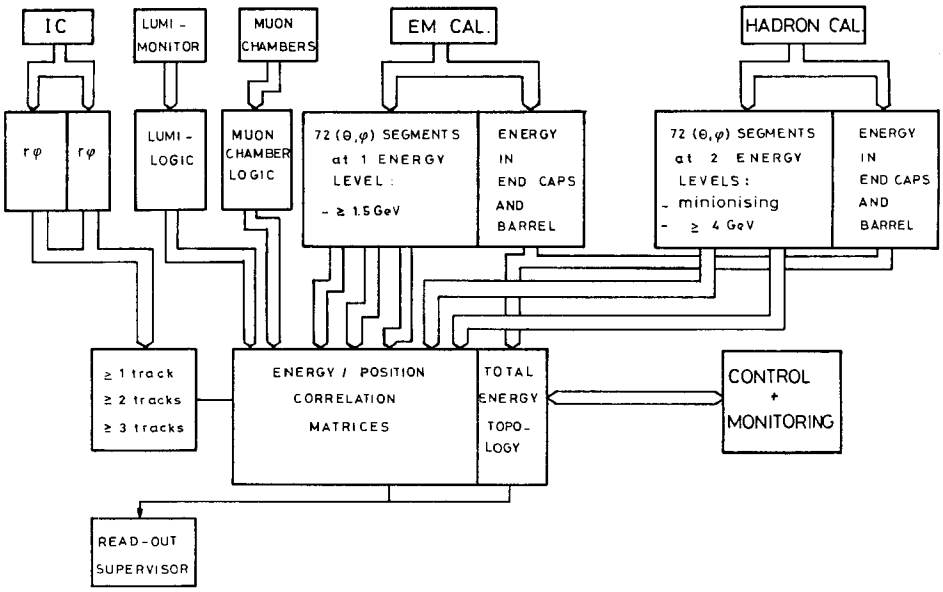


Fig. 11 Level 1 trigger logic for ALEPH

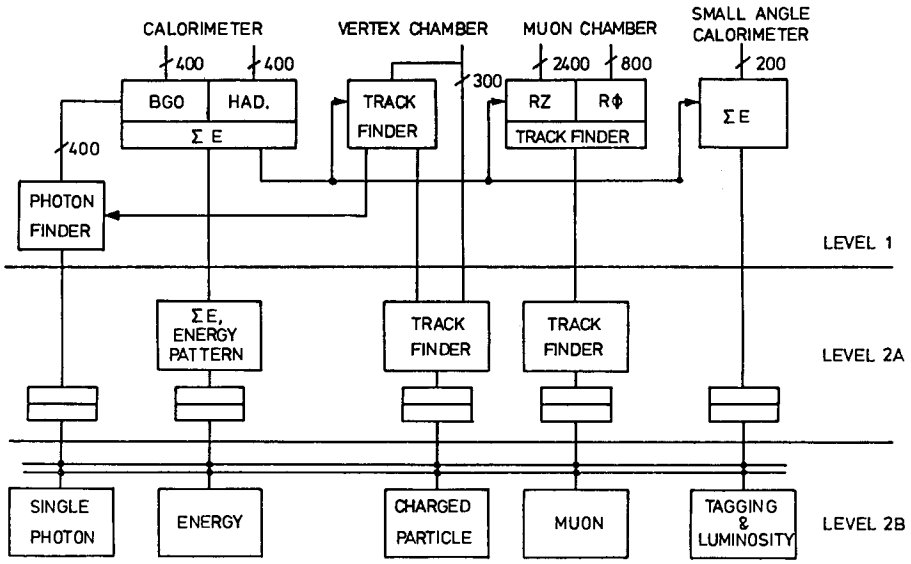


Fig. 12 Details of level 1 and 2 triggers for L3

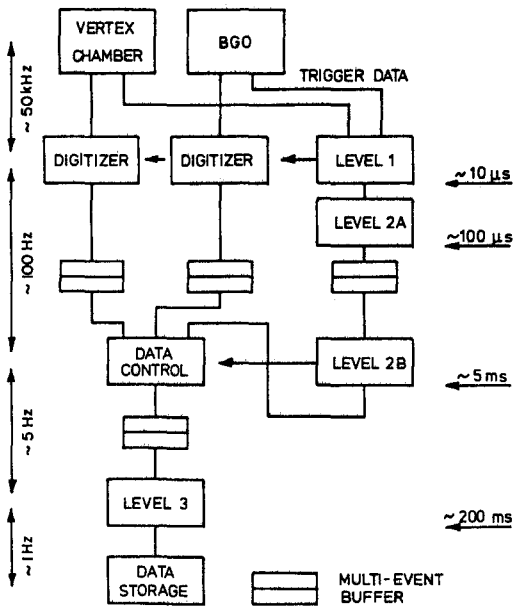


Fig. 13 L3 trigger and data acquisition system block diagram

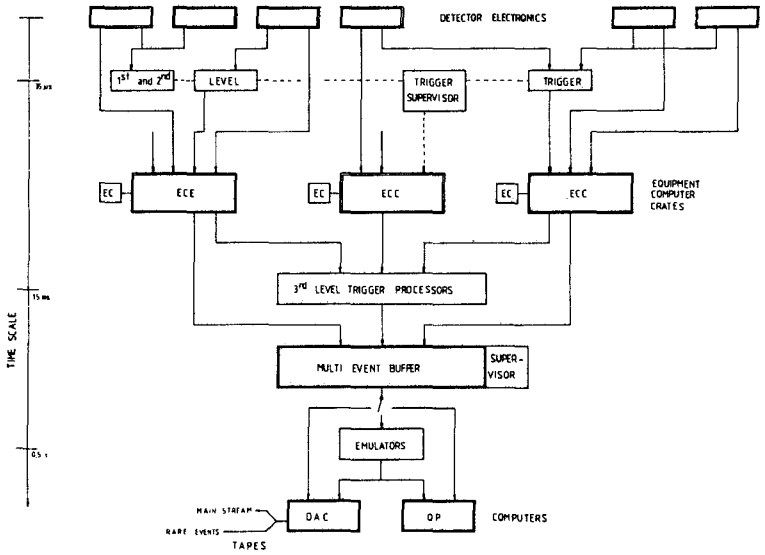


Fig. 14 Schematic diagram of the architecture of the data acquisition and filter system for DELPHI

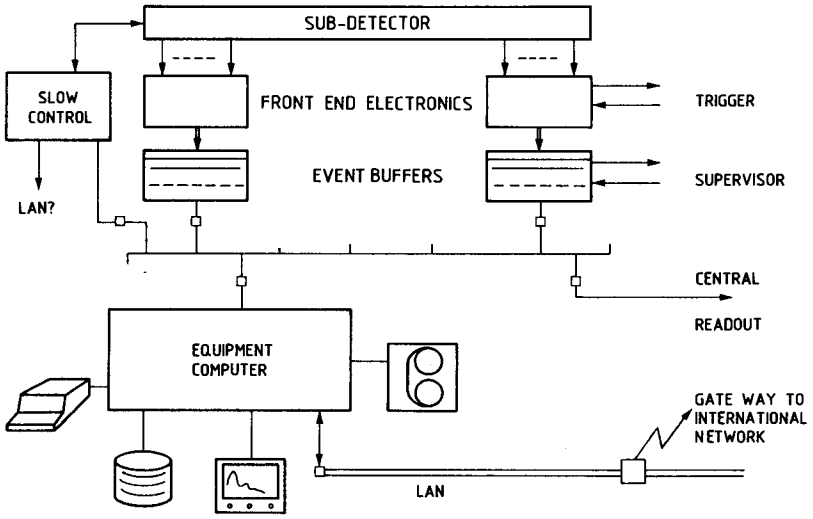


Fig. 15 A typical ALEPH sub-detector

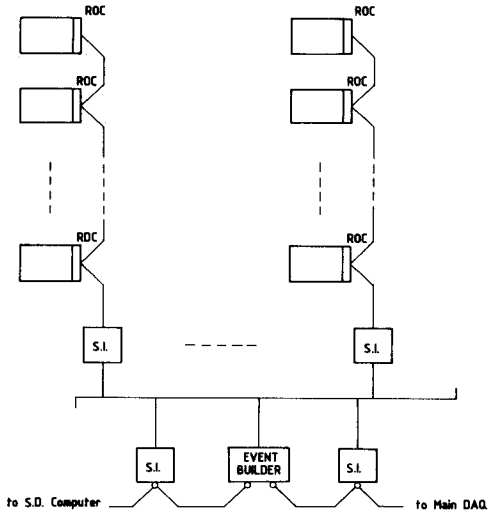


Fig. 16 ALEPH sub-detector readout

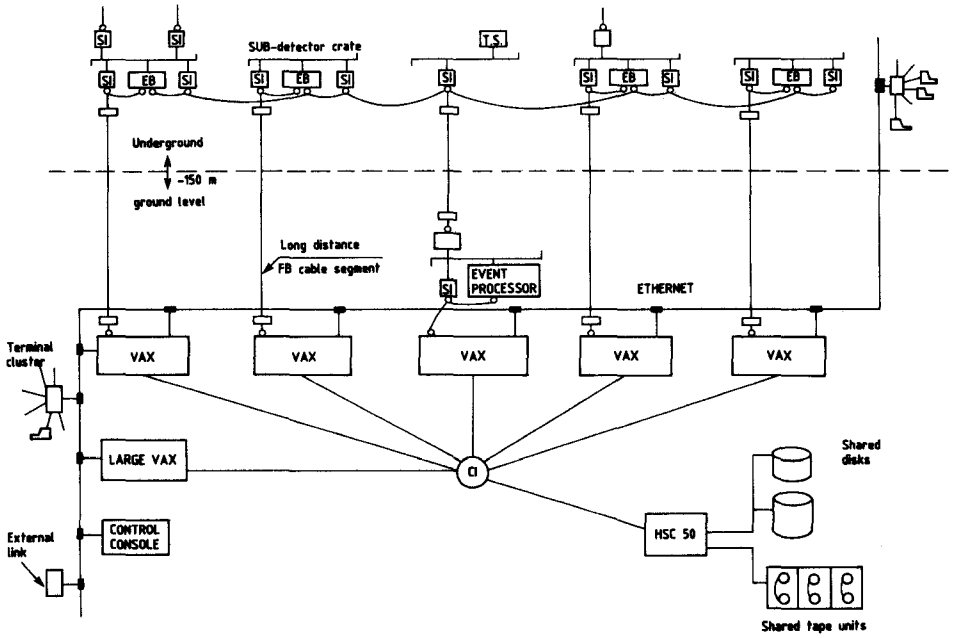


Fig. 17 ALEPH central readout (preliminary)

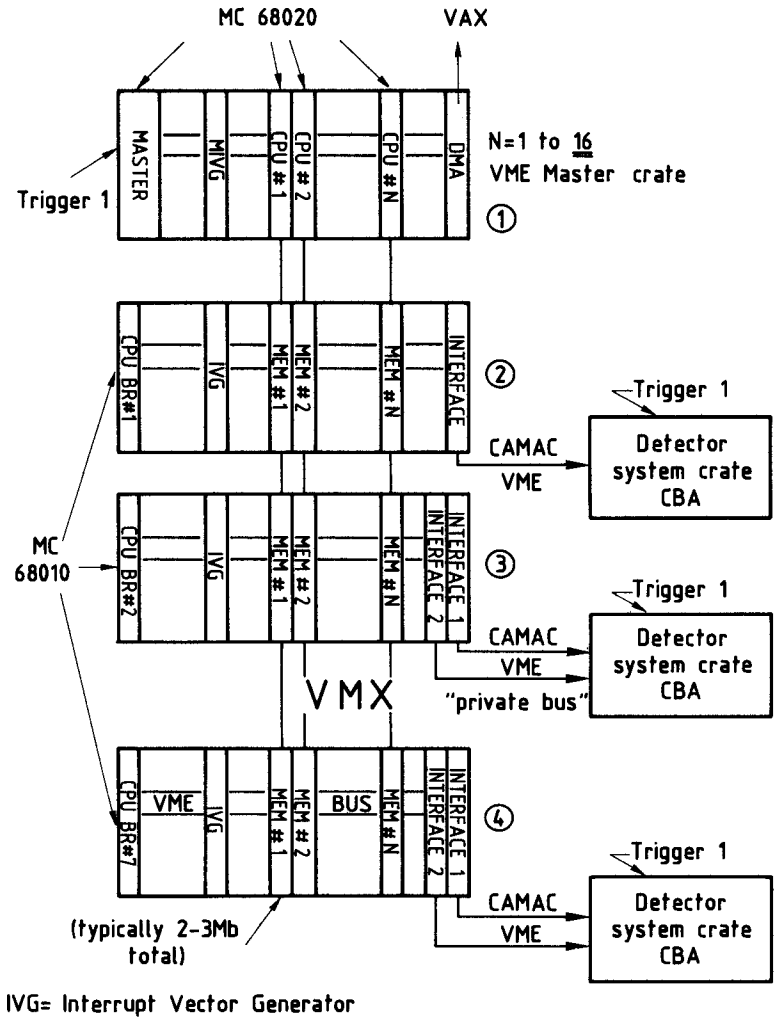


Fig. 18 OPAL VME interface and second level trigger

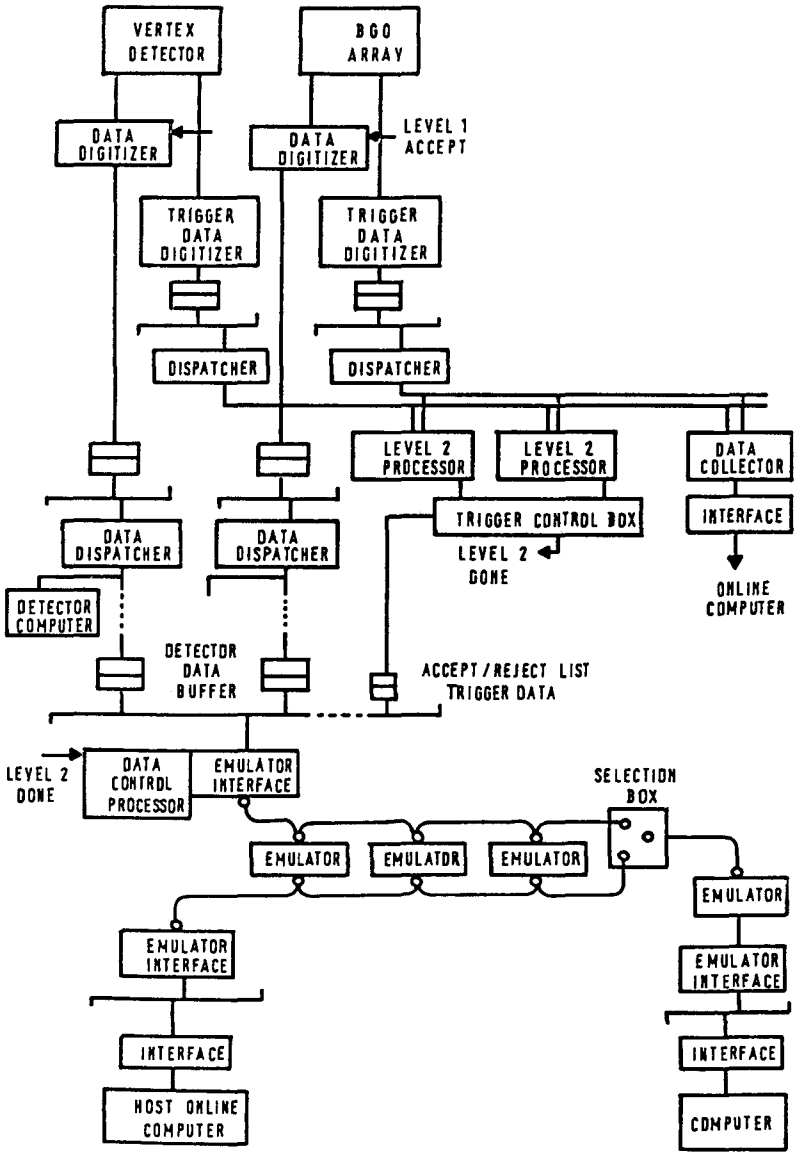
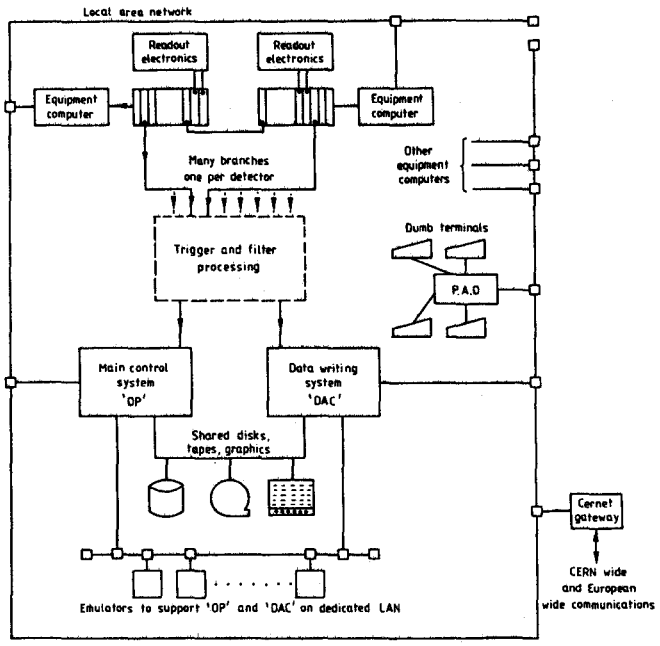


Fig. 19 L3 data flow



DELPHI on-line system

Fig. 20 Overall system architecture for DELPHI computer control/data acquisition system

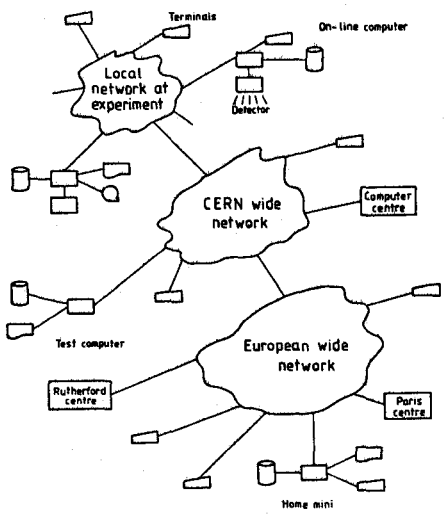


Fig. 21 Levels of networking required by DELPHI

QUESTIONS AND ANSWERS

Q: You are building very sophisticated detectors and processors, but you still talk about writing magnetic tapes. There are new mass storage media available, like optical discs; are you looking into these?

M. Delfino

A: Yes, we are definitely looking into optical discs. If we can obtain commercial units, we will use them instead of tape, assuming the prices are reasonable.

Q: How much computer time will it take to process the 20-30,000 tapes/year?

M. Fischler

A: Enormous

Q: Can one use Ethernet for the Aleph computer interconnect?

D. Notz

A: Ethernet might be too slow. The main Ethernet is used for terminals and networks.

Q: Can you comment further on the software problems of linking different systems?

T. Brody

A: The fast front-end processor will be single-tasked, mostly in assembly language; the problem will appear chiefly in the larger machines. We hope to avoid some of the problems through standardizing as far as reasonable on VAX/VMS.

State of the Art in University Computer Science and Industry

[Gallery of Cardinal Valenti, 1749, Giovanni Paolo Pannini, 1692-1765]

THE FASTBUS MICRO-VAX

Eric J. Siskind
NYCB Real-Time Computing, Inc.
Port Washington, New York 11050

The advent of the cheap, high performance, 32 bit microprocessors has made the sophisticated manipulation of data within a Fastbus system appear attractive, whether for online preprocessing and/or filtering, offline reconstruction, or a variety of unique monitoring and control functions. The imminent availability of VLSI microprocessors implementing the DEC Micro-VAX architecture at performance levels comparable to that achieved by the VAX-11/780 super-minicomputer has led the U. S. Department of Energy to support the development of Fastbus modules containing VLSI Micro-VAX processors via funding provided by the DoE Small Business Innovation Research (SBIR) program in the Office of Energy Research.

The hardware architecture that has evolved features two main sections: (1) a microcomputer unit containing microprocessor, main memory, ROM bootstrap, and console terminal interface; and (2) an intelligent I/O subsystem containing a second microprocessor, ROM firmware, RAM buffers, an Ethernet port, and a simple unintelligent high performance buffered Fastbus interface. Several physical packaging options ("form factors") are being considered for this hardware, and will be described below.

The microcomputer section of the Fastbus Micro-VAX is a standard DEC single board computer packaged as a quad height Q-Bus board, and is in fact the processor board of the future MicroVAX-II computing system. Although this project was initiated with the assumption that one would design with the Micro-VAX microprocessor chip directly, and in fact an architecture for the microcomputer section was developed before details of the MicroVAX-II board organization were forthcoming from DEC, subsequent discussions established the fact that the two architectures were essentially identical, especially in regard to the interrelationships between the microprocessor chip, the main RAM, and the I/O bus (here the Q-Bus). In addition, the design of the DEC board evidenced a great deal of attention paid to the details of RAM timing in order to minimize memory latency as perceived by the microprocessor chip, and thus maximize the computational performance of the system. Finally, the advantages of using a standard DEC board level OEM product, both in regard to certainty of availability, and in terms of additional support and maintainability, cannot be overestimated.

The ROM bootstrap provided by DEC on this board contains programs to initiate the loading of a disk based operating system or to trigger downline loading of an application into RAM via Ethernet. The on-card RAM is sized to accomodate most reasonable single application programs, and can be extended via the inclusion of a number of additional Q-Bus boards.

The I/O subsystem is centered around an Intel 80186 16 bit microprocessor, which is equipped with a pair of block DMA channels. Hardware in the I/O subsystem provides three independent 64 kilobyte page mapped windows into the 22 bit wide Q-Bus address space. These are used by the two DMA channels and the 80186 processor itself. The DMA channels are utilized to move data blocks between the Q-Bus and buffers in either the 80186 RAM or the Fastbus interface. An Ethernet port is provided by the inclusion of an 82586/82501 pair.

The prototype 80186 Fastbus interface is implemented with 13 ECL 10K drivers and receivers for the 80186 connection, 22 ECL 10K drivers and receivers for the Fastbus, 2 ECL 10K one-shots, 8 ECL 10K 16 by 4 bit RAMs, 16 ECL 10K 1024 by 4 bit RAMs, 156 ECL 100K control logic chips, and 4 MCA 2500 ECL data paths macrocell arrays. It is anticipated that the 156 ECL 100K control logic chips will be replaced in a production version with 3 additional MCA 2500 ECL macrocell arrays.

As a Fastbus master, the interface is capable of moving data blocks of up to 4 kilobytes between an internal RAM buffer and the Fastbus in handshake block transfer mode. As a Fastbus slave, the interface is capable of being accessed in data space as a second 4 kilobyte buffer. In addition, the slave hardware implements CSRs 0, 3, 8, A, B, and 100-10F (the first interrupt receiver block).

All control logic is of synchronous design with the exception of the Fastbus arbitration circuit. Clock speed for the prototype circuit in discrete ECL 100K is 100 MHz, while the design specification for the production version utilizing macrocell arrays is 150 MHz. The slave circuitry features DS to DK latency of $3\frac{1}{2}$ clock periods (average) and minimum cycle time of 4 clock periods, while the master circuitry features DK to DS latency of $3\frac{1}{2}$ clock periods (average) and minimum cycle time of 4 clock periods. When the master section of one interface is attached to the slave section of another interface, the average cycle time is thus 7 clock periods plus a round trip propagation delay (including bus drivers and receivers) on the backplane. Using a conservative 10 ns figure for unidirectional propagation delay and 150 MHz clocking, this corresponds to a design goal for transfer bandwidth of 15 MHz, or 60 megabyte/second operation within a single Fastbus crate.

The 80186 Fastbus interface is packaged as a single Fastbus board containing only ECL circuitry, and is accessed with 51 differential ECL signal pair on two 60 conductor ribbon cables, which connect to the board via the upper Fastbus backplane connector. The remaining portion of the I/O subsystem, consisting of the 80186, its RAM and ROM, the Q-Bus interface, the Ethernet port, and level shifters for the connection to the Fastbus interface, consists entirely of TTL logic, and is expected to be available in two form factors. In the first of these, it is packaged a single quad height Q-Bus board to be inserted into the backplane of an existing MicroVAX-II computer. This packaging is expected to appeal to those users who already own MicroVAX-II computing systems, or who wish to equip their Micro-VAX with the largest amount of expansion memory. The other form factor for the 80186 is as the rear half of a Fastbus board, the front half of which contains a single Q-Bus slot into which the MicroVAX-II processor board is plugged "piggy-back" style in a fashion similar to the SLAC Fastbus Controller.[1] The complete Fastbus Micro-VAX module then consists of this hybrid Fastbus board plus the 80186 Fastbus interface board. It is expected that one could still expand the memory capacity of such a module by piggy-backing additional Q-Bus memory cards onto passive Fastbus boards, and by providing the necessary additional cabling via the upper Fastbus backplane connector and/or over-the-top cabling.

The ROM firmware in the 80186 emulates the operation of 5 Q-Bus peripherals when viewed from the Micro-VAX: (1) list processing Fastbus segment driver, utilizing only the master portion of the Fastbus interface, (2) Fastbus interrupt receiver, (3) a flow controlled minimal network using both the master and slave hardware of the Fastbus interface, (4) an Ethernet interface (the DEQNA), and

(5) a standard small Q-Bus based disk. The disk emulation requires the presence on the Fastbus of a host VAX/VMS system with large disk capacity. A reasonable sized (e.g. 10 megabyte) file on the host system provides the actual storage medium, and is accessed with the aid of a Fastbus based protocol known to the 80186 and a disk server process running in the host. Alternatively, a standard commercial virtual disk driver on the host may be used to access this storage. The host disk server process utilizes VAX/VMS system services to access the disk file and standard Fastbus access subroutines to manipulate the Fastbus.

Exact emulation of the DEQNA and a standard DEC Q-Bus disk, although requiring a significant amount of 80186 programming effort, has the advantage of providing access to a vast amount of software which uses these peripherals as system devices. In particular, the disk emulation is sufficient to ensure that any disk based operating system for the MicroVAX-II will run on the Fastbus Micro-VAX with absolutely no modifications. This has the desirable effect of providing the convenient Micro/VMS operating system environment on the Fastbus Micro-VAX, albeit with a vastly reduced ability to support continual paging. Emulation of the DEQNA immediately equips that Micro/VMS node with a DECnet/Ethernet link to the host system, with all of the functionality provided by that additional network. DEQNA emulation also provides a means of downloading VAX/ELN applications to the Fastbus Micro-VAX with no changes to that software.

Ultimately, the goal of the software effort is to make a number of Fastbus Micro-VAX systems appear to a user of the host VAX system as unused processors on a DECnet. A user of the host can then utilize the DECnet to provide remote virtual terminals on the micros, to submit batch reconstruction jobs to the micros, and to otherwise spawn remote processes with I/O interconnections ("transparent inter-task communication") in order to minimize the programming effort required to utilize this additional processing power. The Fastbus is then used to provide a high bandwidth interconnection with which to furnish this processing system with the necessary data.

This research is supported by the U. S. Department of Energy under SBIR contract DE-AC01-83ER80078. Additional support has been provided by North Shore University Hospital/Cornell University Medical College, and by Digital Equipment Corporation, Laboratory Data Products Group. The author wishes to particularly acknowledge the efforts and encouragement of J. Morrison, M.D., and of T. Rabe, M. Peterson, and J. Giudice.

1. S. R. Deiss, "A Fastbus Controller Module Using A Multibus MPU," I.E.E.E. Trans. Nucl. Sci., NS-30, 216, (1983).

QUESTIONS AND ANSWERS

Q: How do you get around the fact that VAX software assumes true virtual memory (that is, swapping not only of load modules)?

M. Fischler

A: You lock task into memory, using SYSLOCK. Therefore, memory must be large enough. It is expandable.

LATTICE GAUGE THEORY CALCULATIONS ON THE CDC CYBER 205

D. Barkai
Control Data Corporation
at the
Institute for Computational Studies at CSU
P.O. Box 1852
Fort Collins, Colorado 80522, USA

K.J.M. Moriarty *
Department of Mathematics, Statistics
and Computing Science
Dalhousie University
Halifax, Nova Scotia B3H 4H8, Canada
and
Department of Mathematics
Royal Holloway College
Englefield Green, Surrey TW20 OEX, U.K.

and

C. Rebbi
Department of Physics
Brookhaven National Laboratory
Upton, New York 11973, U.S.A.

1. Introduction

The CDC Cyber 205, because of its rich instruction set and certain other features, is the ideal supercomputer for carrying out lattice gauge theory Quantum Chromodynamics (QCD) calculations. Below, we present a brief discussion of some of these features.

2. Make or Buy

We wish to study the gauge theory of strong interactions known as QCD by formulating the theory on a space-time lattice. This is a very computationally intensive problem as has been indicated in a number of recent publications [1-3]. The emphasis of the present meeting is on making your own "supercomputer". We have decided not to build our own machine but to use a commercially available supercomputer because:

*Presenter

1) in obtaining government funds it is probably as easy to obtain $20,000 as it is to obtain $20,000,000. Because the effort is the same, why not go for the larger amount?

2) physicists should spend their time doing physics rather than becoming quasi-computer scientists. As a result of building their own computers, physicists are duplicating effort, there is no coordination of effort and many mistakes, already known in the industry, are repeated.

3) in arriving at the cost of do-it-yourself computers faculty members salaries and equipment readily available in departments is not factored in. Thus, costs are undervalued.

4) if we buy off-the-shelf technology from the main vendors and insist on certain enhancements, we can have an effect on future developments in the supercomputer industry.

5) fields of physics change rapidly in the techniques used and we do not want these hardwired. Interest may be gone by the time special purpose processors are ready for use.

6) a generic supercomputer has multiple uses and is easy to program in portable code.

7) with the implementation of the 8X standard of FORTRAN, codes will be transportable across all scalar and vector machines.

8) certain sectors of the scientific community, e.g. the bomb makers, have lots of money and could therefore make their own machines but they do not. Why not?

3. Supercomputer Features

The major supercomputers currently available from commercial vendors are the CRAY-1, CRAY-XMP, CDC CYBER 205 and the Fujitsu VP200. Of these machines, the CDC CYBER 205 has the following features which make it attractive for Monte Carlo simulations:

1) Many "data motion" instructions, e.g. GATHER/SCATTER, COMPRESS/DECOMPRESS, MASK, MERGE etc., have been hardwired into the machine and thus proceed at machine rates.

2) A virtual memory facility exists which allows one to easily move out of real memory and consider problems too big to fit into real memory. Software features exist to help to do this efficiently.

3) Memory locations are bit-, byte-, half-word (32-bits) and full-word (64-bits) addressible. Thus, when working in 32-bit arithmetic, the amount of real memory is doubled and the arithmetic operation rate is doubled. This is useful in view of the fact that much arithmetic in scientific calculations only requires 32-bit accuracy.

4. Performance

We have recently implemented pure SU(3) gauge field calculations [4,5] of the static quark force on a $16^3 \times 32$ lattice on a 2-million word 2 vector pipeline CDC CYBER 205. In upgrading link variables [5] much use has to be made of the GATHER/SCATTER instructions, e.g. in the 32-bit arithmetic mode these amount to 30% of our execution time. In 32-bit arithmetic, the sustained performance rate achieved is 130 MFLOPS with a burst performance rate of 182 MFLOPS. With four vector pipelines (one result ever 5 nanoseconds), these rates can be increased by about 35%. In running a problem with about 9 million degrees of freedom utilizing only a little over 1 million words and using the virtual memory facility, 98% utilization of the machine results, a degrading of the machine of only 2%. These are impressive performance figures.

5. Acknowledgement

We would like to thank Control Data Corporation for awarding time on the CDC CYBER 205 at the Institute for Computational Studies at Colorado State University where the code described in the text was developed.

6. References

[1] D. Barkai and K.J.M. Moriarty, Comput. Phys. Commun. 25, 57 (1982); 26, 477 (1982); 27, 105 (1982).

[2] D. Barkai, M. Creutz and K.J.M. Moriarty, Comput. Phys. Commun. 30, 13 (1983).

[3] D. Barkai, K.J.M. Moriarty and C. Rebbi, Comput. Phys. Commun. 32, 1 (1983).

[4] D. Barkai, K.J.M. Moriarty and C. Rebbi, Phys. Rev. D30, (1984).

[5] D. Barkai, K.J.M. Moriarty and C. Rebbi, Submitted to Phys. Rev. D.

QUESTIONS AND ANSWERS

Q: Have you in fact been funded at the $20M level?
M. Kreisler

A: Yes

THE VLSI REVOLUTION IN COMMERCIAL NUMBER-CRUNCHING OPPORTUNITIES FOR IMPROVEMENT VIA SPECIALIZATION

Alan E. Charlesworth
Floating Point Systems

INTRODUCTION

Floating Point Systems manufactures what have been misleadingly named "Array Processors". They are often called APs for short. So far APs have not been "arrays of processors", but instead "processors of arrays." In fact, APs are nothing other than low-end super-computers [Thei81].

Most APs, along with most commercial super-computers, have been organized as pipelined Von Neuman machines [Norr84]. In particular, they are optimized to perform sums of products on rows and columns of matrices [Dong84]. Multiply-adds are fundamental to most signal processing, in the time or frequency domain filtering of signals [Bowe82]. And, of course, they are basic to scientific and engineering simulation, in the evaluation of continuous field problems via the solution of simultaneous linear equations [Vemu81].

So far, most array processors have been intended for signal processing, as befits their 32-bit arithmetic, small memory, and primative software environment. The FPS-164, though, is a legimate low-end super-computer, with 11 Million FLoating point OPerations per second (MFLOP) arithmetic, several megawords of memory, pipelining compiler, and operating/file system.

CHART 1: RECENT REDUCTIONS IN THE COSTS OF NUMBER-CRUNCHING

1.1 THE VLSI LOG JAM

The integrated circuit technology for building number-crunchers remained almost unchanged from the mid 1970's up until very recently. There are two primary reasons for this:

1. High speed numeric computing is a very small part of the computer business. Semi-conductor industry effort has focused on the main-stream of computing, which is business record procesing, text editing, and personal computers.

2. 64-bit wide floating-point arithmetic and data path components require much more silicon and packaging resource than those for non-numeric applications. Only the most recent state of the art could possibly accomodate them on a single chip. Arithmetic, in particular, does not separate well into sub-chips. Thus it has been a case of all or nothing, waiting for technolcgy to catch up.

So, only bulk memories for number-crunchers got denser, since such memory is universal to all computing. Today's Very Large Scale Integration (VLSI), when applied to memory, allows up to 256,000 logic devices on a chip - enough to hold a quarter million bits of data. This has allowed computer memories to become quite small and economical of power - important for reduced cost.

Processor components, though, have been stuck at Medium Scale Integration (MSI) - a few dozen to a few hundred gates per chip. Thus, fast numeric processors have remained very large and power hungry, and hence expensive (typically one half million to twenty million dollars).

1.2 VLSI FOR NUMBER-CRUNCHING

Finally, the number-crunching market has grown large enough tc warrent applying VLSI to its needs, and VLSI can now accomodate 64-bit functional units inside a single chip. The improvement over 1979 technology is enormous. Size and power consumption (both good indicators of cost) have been reduced by 1.5 to over two orders of magnitude. The challenge is to realize this major leap forward at the system level: either to make small low-end crunchers, or else to implement reasonably priced super-computers.

CHART 1: RECENT REDUCTIONS IN THE COSTS OF NUMBER-CRUNCHING

64-BIT FUNCTIONAL UNIT SIZE:
1979: a few boards ⟶ 1985: a few chips ⟶ 1988: fractions of a chip

Functional Unit	Bipolar MSI 1979	MOS VLSI 1984/5	Improvement
64-BIT FLOATING-POINT ARITHMETIC adder & multiplier	11 MFLOPs 1750 sq. inches 540 watts	11 MFLOPs 15 sq. inches 10 watts	116x 64x
64-BIT INTERCONNECT 64 Registers & 8 Busses	700 sq. inches 220 watts	24 sq. inches 8 watts	29x 27x
CONTROL Memory Addressing & Next Program Addressing	700 sq. inches 240 watts	6 sq. inches 2 watts	116x 120x
MEMORY	1 million words 375 ns cycle 5600 sq. inches 430 watts (MOS)	1 million words 200 ns cycle: 225 sq. inches 65 wtts	1.9x 25x 6.6x

CHART 2: EXPLOITING VLSI OPPORTUNITIES

2.1 REPLICATION

VLSI is indeed quite a different ballgame. To a much greater extent than earlier digital technologies, VLSI is oriented to replication [Seit82]. Chips are manufactured in parallel, by projection onto silicon, unlike the sequential board-level manufacturing steps of wirewrapping and chip insertion.

In a monolithic super-computer, components must be chosen for maximum speed, reguardless of expense. Replication, on the other hand, allows performance and cost-effectivity to be optimized independently. The speed of the replicated "cell" can be chosen for best cost-effectivity, given the current state of the art. Then total speed can be made up on volume by using enough copies, perhaps hundreds or thousands, to reach the desired speed. The only limit is that the natural parallelism of the problem be greater than the number of cells required.

2.2 COMMUNICATION VS. SWITCHING

The cost relationship between switches (logic gates) and communication (wires) is reversed between VLSI and older technology [Seit82]. In the extreme case, when computers were built from tubes, the logic was extemely hot, un-reliable, and expensive, and the wires between the tubes cost virtually nothing. VLSI is opposite. In the limit, wires and the chip area needed to drive their capacitive load are the dominating cost, both in speed and power.

This communications limit is a fundamental characteristic of VLSI. High interaction bandwidths are possible only between parts of a system that are physically close. Thus, generalized "everywhere to everywhere" types of connections carry a heavy cost penalty. Instead, it is better to tailor a system's connections to match the communications pattern of the dominant computational algorithm. A spectrum of localized communication geometries include 2-D nearest-neighbor meshes, hypercubes, and trees [Loca80]. There is considerable experience with 2-D meshes, from the Illiac IV [Hord82], to the PACS [Hosh83].

2.3 PROCESSOR SPECIALIZATION

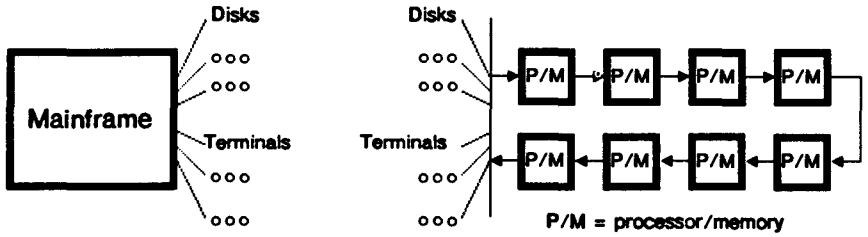
Only the minimum necessary degree of processor complexity should be replicated, so as to be able to get as many copies as possible localized onto a single chip. There is a spectrum of possibilities [Seit82]. Given that the purpose here is to do number-crunching, the absolute minimum to replicate is just arithmetic. A close approximation to this would be to duplicate four multipliers and six adders into a fixed pattern in an FFT butterfly array. An intermediate level of complexity is a systolic cell, which adds a little bit of adjustable control, a few registers, and perhaps a small amount of memory [Kung83]. At the upper end is the "homogeneous machine" cell [Seit82], of which a prototype example is the 64-processor Cosmic Cube [CalT83]. Each cell is a complete small computer, with enough generality to support an extensive run-time system.

CHART 2: EXPLOITING VLSI OPPORTUNITIES

VLSI IS ORIENTED TO REPLICATION

PRE VLSI ERA: → VLSI ERA:

Monolithic mainframe. → Replicated processors.



Choose a cost-effectiveness maxima, then replicate.

VLSI's DOMINANT COSTS ARE DIFFERENT

PRE VLSI ERA: → VLSI ERA:

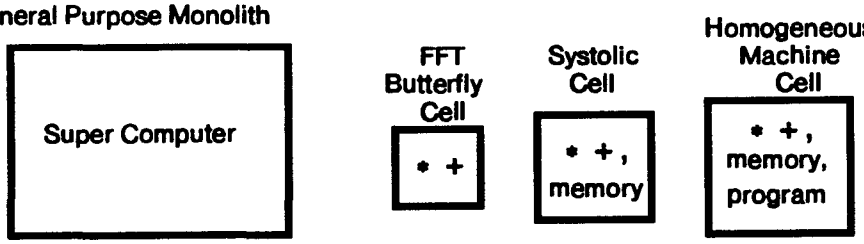
Most of the cost in switches. → Most of the cost in communication.

Implement a specific communication pattern.

OFF-CHIP COMMUNICATION IS EXPENSIVE IN VLSI

PRE VLSI ERA: → VLSI ERA:
Algorithm specific cells.

General Purpose Monolith



Specialize processors to get more on a chip.

CHART 3: APPLYING REPLICATION TO AN EASY PROBLEM

3.1 CHOOSE AN EASY PROBLEM

It is always best to start on easy problems first, which is the case with FPS's first replicated product. For a number-cruncher, the easiest problem is one which has a maximum amount of arithmetic to do, on a minimum amount of data, with a simple and orderly pattern of computation.

One such category are those matrix operations which do on the order of N cubed operations, on N squared elements - where N is the dimension of the matrix. The most important of these to scientific computing are matrix multiplication and matrix factoring, the dominant step in linear equation solving. Important to signal processing are convolution and discrete Fourier transforms, both of which do N squared operations on N data items. A typical size for N is 1000, which implies on the order of 1000 floating-point operations on each element. Such a high re-use of data implies that computation rates in the 100's of MFLOPs can be supported without need for enormous fast memories. The low data required can be supported from disk files.

3.2 MATRIX SUM OF PRODUCTS

The fundamental operation here is the matrix-vector product $c \leftarrow a \cdot B$, which pervades scientific computations [Dong84]. Vectors from the matrix 'B' will be loaded into multiple "vector registers", each associated with a multiply-adder. A vector 'a' (usually out of a complete matrix 'A') is then broadcast to all the multiply-adders. They multiply it against their partcular vectors from 'B', and keep their running sums going. In this way, parallelism is obtained up to the number of 'B' vectors that can be accomodated.

3.3 MINIMIZE WHAT MUST BE REPLICATED

To keep the replicated cell as small as possible, only the absolute minimum support structure is included. Ideally, each cell would occupy a single chip (or less), but currently, commercial VLSI is limited to implementing 64-bit arithmetic on several chips. This, then, sets the replicated cell size to be single-board scale. Future VLSI progress will make chip-scale replication possible.

Interconnect is minimized by supporting only four computational functions. Local control complexity is minimized by not allowing the cells to be programmed internally. Instead, they are sequenced by the program in the FPS-164. Local memory is minimized by having each cell hold only several vectors of data. The complete matrices are stored in FPS-164 memory, or on disk. The communication bandwidth required to feed the cells is minimized by filling them all simultaneously via inputs broadcast on the FPS-164 memory bus.

CHART 3: APPLYING REPLICATION TO AN EASY PROBLEM

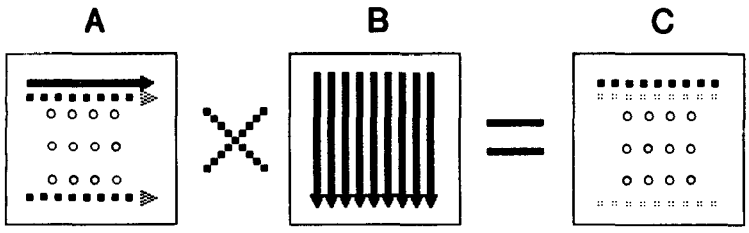
ORDER N OPERATIONS PER POINT

Matrix multiply	N^3 operations	N^2 data
Matrix factor	N^3 operations	N^2 data
Convolution	N^2 operations	N data
Discrete Fourier transform	N^2 operations	N data

Maximize arithmetic, minimize everything else.

MATRIX SUM OF PRODUCTS

(inner-product form shown)



Broadcast matrix

Multiply/add against many vectors simultaneously

Accumulate a vector of sums simultaneously

(4K max vector length)

(4 vectors/sums per replication, up to 124 total)

MINIMIZE THE SIZE OF WHAT IS BEING REPLICATED

Goal: Board scale replication.

Arithmetic	Use chip-set scale 64-bit floating-point.
Interconnect	Fixed for only three vector forms.
Control	Control from program in FPS-164.
Memory	Vector sized (4K words)
Communications	Broadcast on FPS-164 memory bus.
Mass Storage	Needs bandwidth of one to several SMD/ISI disks.

CHART 4: THE FPS-164 MATRIX ALGEBRA ACCELERATOR (MAX)

4.1 ARITHMETIC

Given the large size (16" x 22") of FPS-164 boards, it is possible to fit two multiply-add cells on one board. At a 5.5 Mhz system clock, this yields 22 MFLOPs maximum per board. The FPS-164 can accomodate up to 15 MAX boards, for a total of 31 multiply-add cells (including the FPS-164 itself), for a maximum rate of 341 MFLOPs.

4.2 COMPUTE FUNCTION

The only compute function the cells perform is a multiply followed by an add. One input operand is always the broadcast data from the FPS-164. Otherwise, the vector and scalar registers can be either inputs or outputs, so that both inner and outer product forms can be accomplished.

4.3 MEMORY

Each cell is supported by four 4K-word vector registers, to hold its matrix rows or columns, and four scalar registers in which to accumulate sums. Each of the sets of four vector and scalar registers is accessed in rotation, one per multiply-add cycle. Thus each cell accumulates four sums of products. The vector register address accessed by the arithmetic may be held fixed, incremented by one, or be supplied by the FPS-164. These combinations permit operations on real, complex, and sparse vectors.

4.4 COMMUNICATIONS

The MAX vector and scalar registers are mapped into the FPS-164 memory space. Thus they can be read from or written into by an FPS-164 program just like any other portion of memory. In addition, all the cells respond to writes to a special "broadcast" memory address, which supplies data simultaneously to the multipliers of all the cells, and initiates a multiply-add operation.

4.5 CONTROL

A multiply-add operation is initiated simultaneously in all the cells by the FPS-164 program writing to the broadcast address. Upon completion of a dot product, a "collapse" address is written to by the FPS-164 program, which causes the eight partial sums within the adder pipeline to be added together and stored into the scalar registers..

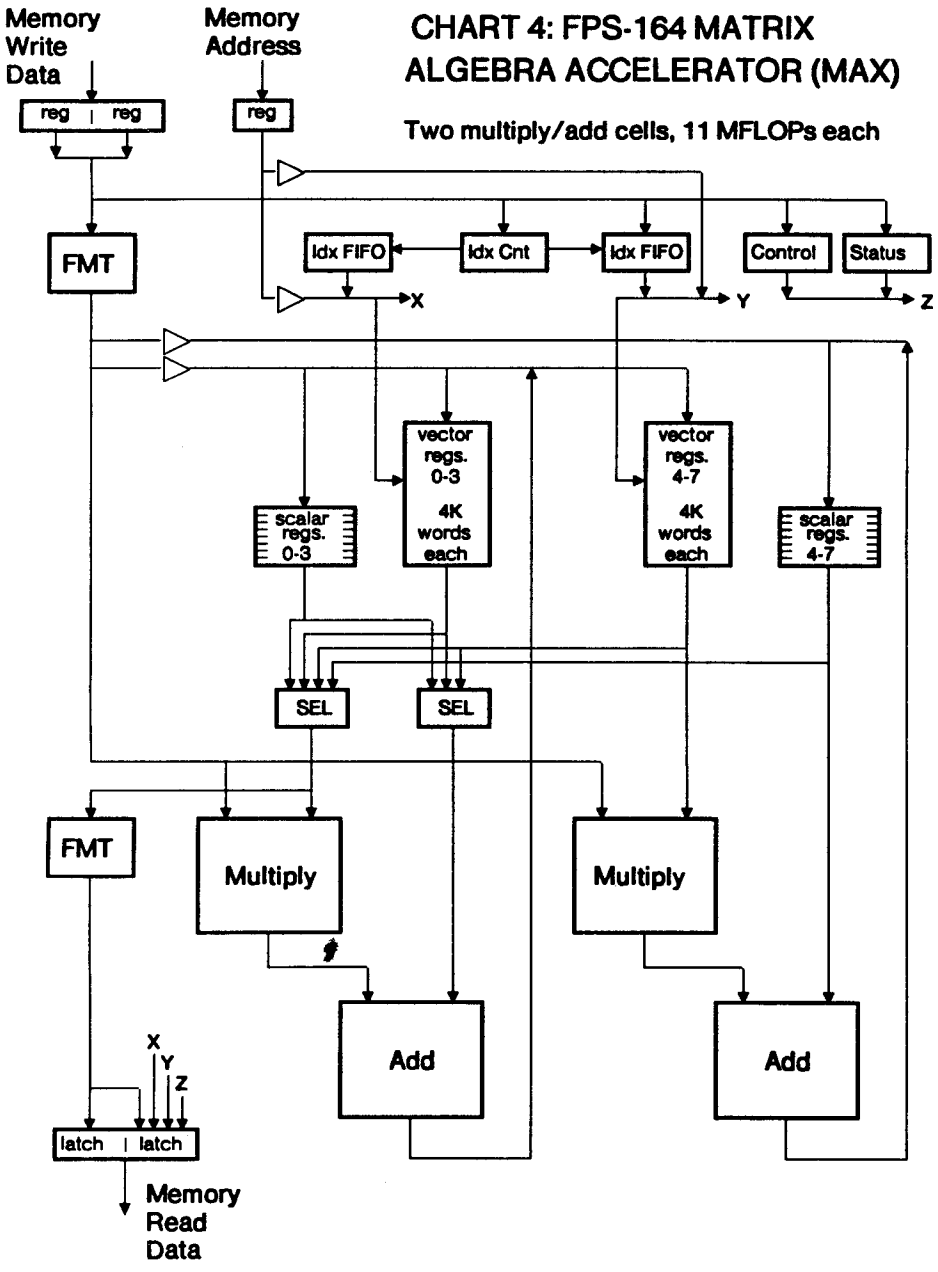

Memory
Write
Data
Memory
Address
CHART 4: FPS-164 MATRIX
ALGEBRA ACCELERATOR (MAX)
Two multiply/add cells, 11 MFLOPs each
reg
reg
reg
FMT
Idx FIFO
Idx Cnt
Idx FIFO
Control
Status
X
Y
Z
vector
regs.
0-3
4K
words
each
vector
regs.
4-7
4K
words
each
scalar
regs.
0-3
scalar
regs.
4-7
SEL
SEL
FMT
Multiply
Multiply
Add
Add
X
Y
Z
latch
latch
Memory
Read
Data

CHART 5: THE FPS-164/MAX FUNCTIONS

5.1 VECTOR FUNCTIONS

The MAX multiply-add cells perform a vector dot product function at the maximum rate of 11 MFLOPs each. There is, however, a five microsecond delay at the end of the operation to collapse the partial sums in the adder pipeline. The two vector/scalar functions operate at only half speed, since only one cell per MAX board is used. This is because two vector operands are required. Thus, except for very short matrix dimensions (where the collapse would be a noticable overhead), the inner product form will be twice as fast as the outer product form.

5.2 OVERALL USE

Matrices are processed four vectors per multiply-add cell. If all 31 cells are present, then 124 vectors can be processed in one broadcast pass. High efficiency will be obtained given two circumstances:

1. That the dimension of the matrix stored in the MAX cells is near a multiple of the available number of cells times four, so that all the available cells will be fully utilized.

2. That the matrix being broadcast is sufficiently large to amortize the time spent loading vector operands into the cells, and unloading the scalar results.

CHART 5: FPS-164/MAX FUNCTIONS

Vector Form	Executed for each value broadcast
VDOT Vector Dot Product	M * V0..7 + S0..7 --> S0..7
VSMA0 Vector scalar multiply/add	M * S0..3 + V0..3 --> V4..7
VSMA1	M * S0..3 + V4..7 --> V0..3
VMSA0 Vector Multiply/Scalar add	M * V0..3 + S0..3 --> V4..7
VMSA1	M * V4..7 + S0..3 --> V0..3

S0..7	Eight scalar registers
V0..7	Eight vector registers (4K words each)
M	Broadcast value
+	Floating-point Add, Subtract, or Sub. Reverse
*	Floating-point Multiply

(The vector registers can also be indexed to handle sparcity.)

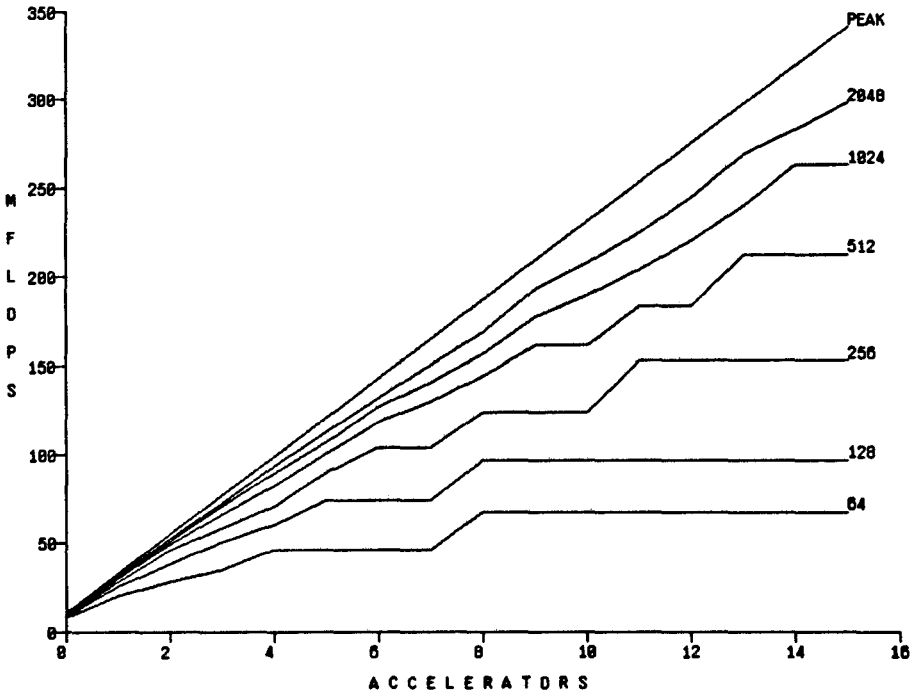
Complex Dot-product:

- Real part in V0, V2, V4, V6; imaginary part in V1, V3, V5, V7.
- Broadcast real part of M without index increment;
  then broadcast imaginary part of M with index increment;
  in an 8 cycle loop for each complex element.

CHART 6: FULL MATRIX MULTIPLY PERFORMANCE

The step-wise jumps result from the matrix dimensions not being exactly a multiple of four times the number of multiply-add cells. This causes under-utilization of some of the cells. The efficiencies are high when the dimension is 32x greater than the number of cells, which is enough to dilute the effects of loading and unloading the accelerators.

CHART 6: FULL MATRIX MULTIPLY PERFORMANCE



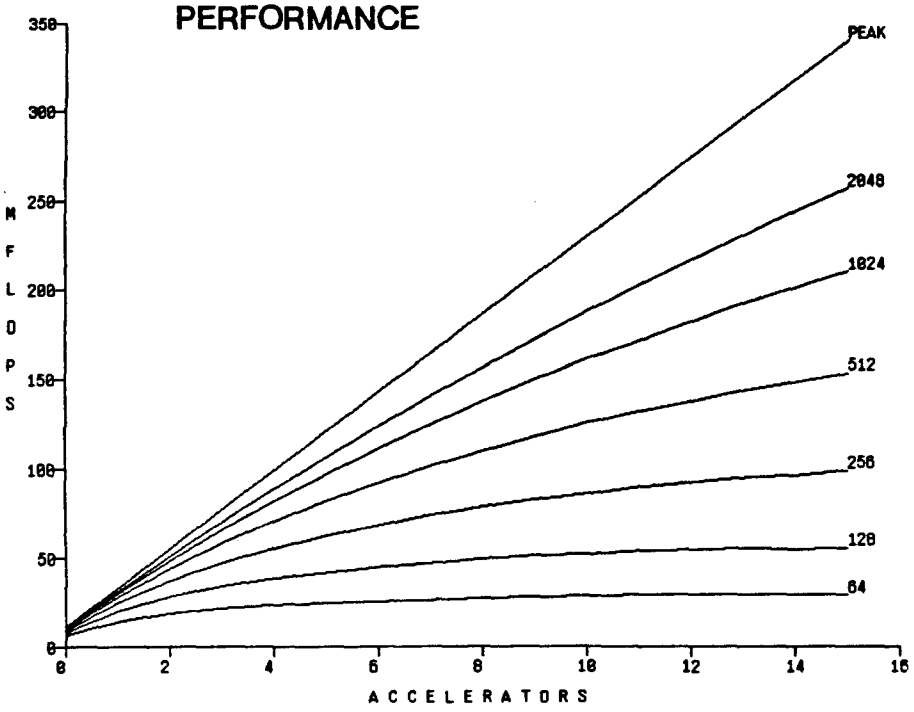
1 to 31 multiply/add cells

Matrix dimensions from 64 to 2048

CHART 7: SYMMETRIC, BANDED MATRIX FACTOR PERFORMANCE

These curves are smooth because the matrix dimensions are very large, 10x the bandwidth, and so minor under-utilization of the cells is almost not noticable. The performances are somewhat lower because a matrix factor has a bit more data-dependency than a matrix multiply.

CHART 7: SYMMETRIC BANDED MATRIX FACTOR PERFORMANCE



1 to 31 multiply/add cells

Half bandwidths from 64 to 2048

Matrix dimensions are 10x the half bandwidth

REFERENCES

[Bowe82] B. A. Bowen, W. H. Brown, VLSI System Design for Digital Signal Processing, Prentice Hall, 1982.

[Broo81] E. Brooks, G. Fox, R. Gupta, O. Martin, S. Otto, "Nearest Neighbor Concurrent Processor", Cal Tech Report CALT-68-867, Sept 1981.

[Dong84] J. Dongarra, F. Gustavson, A. Karp, "Implementing Linear Algebra Algorithms for Dense Matrices on a Pipelined Machine", SIAM Review, Jan 1984.

[Hord82] R. Hord, The Illiac IV, Computer Science Press, 1982.

[Hosh83] T. Hoshino, T. Kawai, T. Shirakawa, J. Higashino, A. Yamaoka, H. Ito, T. Sato, K. Sawada, "PACS: A Parallel Microprocessor Array for Scientific Calculations", ACM Transactions on Computer Science, Aug. 1983.

[Kung83] H. Kung, A. Fisher, L. Monier, "Architecture of the PSC: A Programmable Systolic Chip", Proceedings 10th IEEE Conference on Computer Architecture, June 1983.

[Loca80] B. Locanthi, "The Homogeneous Machine", Cal Tech Report 3759:TR:80, 1980.

[Norr84] C. Norrie, "Supercomputers for Superproblems: An Architectural Introduction", IEEE Computer, March 1984.

[Seit82] C. Seitz, "Ensemble Architectures for VLSI - A Survey and Taxonomy", 1982 Conference on Advanced Research in VLSI, MIT, Jan. 1982.

[Thei81] D. Theis, "Array Processor Architecture", IEEE Computer, Sept 1981.

[Vemu81] V. Vemuri, W. Karplus, Digital Computer Treatment of Partial Differential Equations, Prentice Hall, 1981.

QUESTIONS AND ANSWERS

Q: Will you be able to offer an AP equivalent to the present FP164 at 1/10 of the present price?

C. Bunge

A: By the end of 1985 we expect to have it at the price people reasonably expect.

Q: Do you see any application for tiny array processors to hook onto microprocessors?

I. Gaines

No answer.

Q: Is there a FORTRAN compiler? And, if not, is it true that any inner loop to complex for machine coding is a bad application of MAX?

M. Fischler

A: No and yes.

ETA DIRECTION IN LARGE SCALE COMPUTING

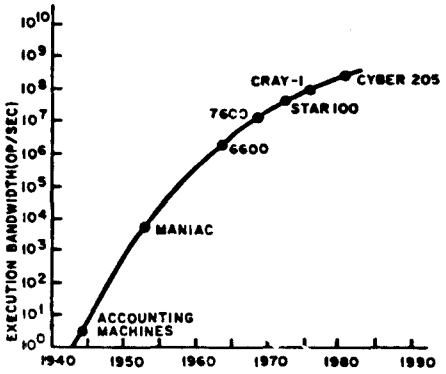
Kent Steiner
ETA Systems, Inc., 1450 Energy Park Drive, St. Paul, Minnesota 55108 USA

ETA SYSTEMS

— DESIGN —
— DEVELOP —
— MANUFACTURE —
— MARKET —
— SUPPORT —

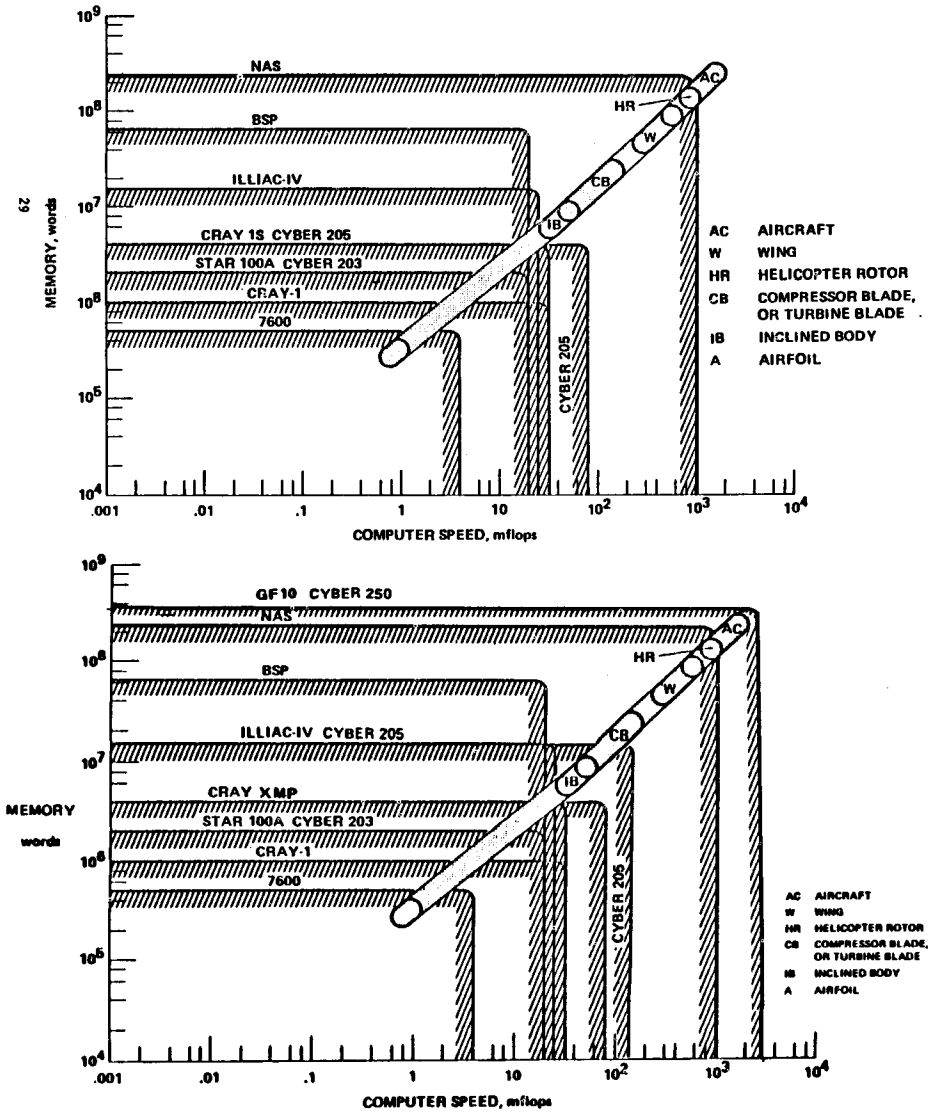
THE WORLD'S FASTEST SUPERCOMPUTER SYSTEM

- EQUITY CARVE OUT FROM CDC
- EMPLOYMENT
  - — CURRENT 180
  - — 1988 PROJECTION 500
- OUR PLANS ARE TO BE A PUBLICLY HELD COMPANY
  - — WHERE CDC HOLDS A MINORITY INTEREST



COMPUTER SPEED AND MEMORY REQUIREMENTS COMPARED WITH COMPUTER CAPABILITIES

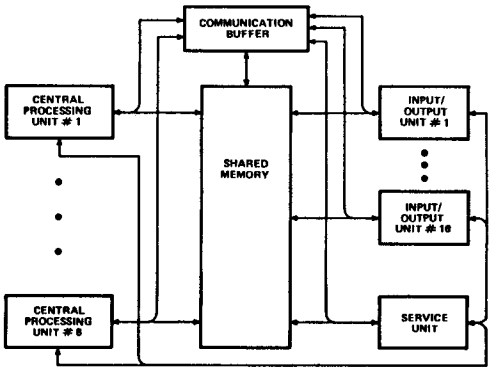
SPEED REQUIREMENT BASED ON 15-min RUN WITH 1985 ALGORITHMS REYNOLDS-AVERAGED NAVIER-STOKES EQUATIONS



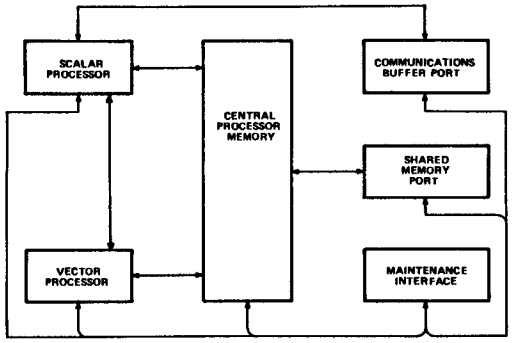
SUPERCOMPUTER CHARACTERISTICS

- MASSIVE COMPUTATIONAL ENGINE
- HIERARCHICAL MEMORY SYSTEM
- LARGE I/O BANDWIDTHS
- FAST ARITHMETIC PROCESSORS
- MULTIPROCESSOR CONFIGURATIONS

GF-10
FUNCTIONAL DIAGRAM



GF-10
CENTRAL PROCESSING UNIT



SOFTWARE SYSTEM OBJECTIVES

o PROVIDE SOFTWARE COMPATIBILITY FROM CYBER 205 THROUGH GF-30

o PROVIDE AN ADAPTABLE SYSTEM CAPABLE OF OPERATING IN MULTIPLE ENVIRONMENT
  - BATCH
  - INTERACTIVE
  - MONOTASKING/MULTITASKING
  - USER INTERFACE

o HIGH SPEED SCIENTIFIC COMPUTATION AND I/O

o VERY HIGH RELIABILITY

o INTERFACE TO A VARIETY OF FRONT-ENDS AND WORKSTATIONS

o EASE OF USE

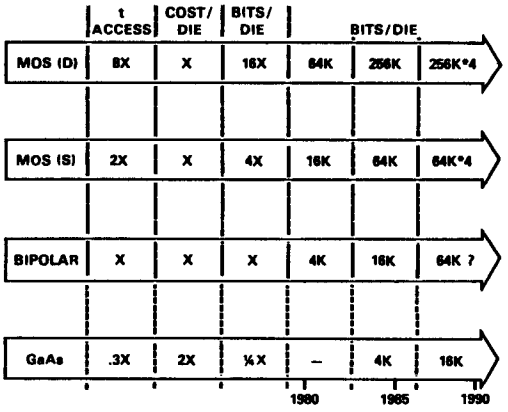
SUPERCOMPUTER TECHNOLOGY REQUIREMENTS

- **ARCHITECTURE**
- **LOGIC CIRCUITS**
- **MEMORY**
- **PACKAGING**
- **DESIGN TOOLS (ECAD)**
- **INPUT/OUTPUT BANDWIDTH**
- **PERIPHERALS**
- **SOFTWARE**

TRANSMISSION TIME VS INTERCONNECT MATERIAL

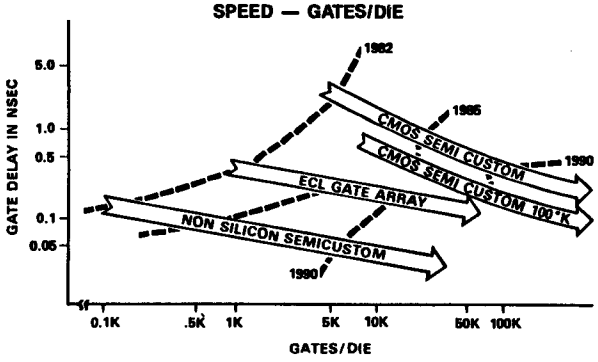
MATERIAL	FACTOR (PS/MICRON)	DISTANCE (MICRON)	DELAY PS	RATIO
SILICON	7.5×10^{-2}	2K	150	1
PC BOARD	10^{-2}	150K	1500	10
COAX	5×10^{-3}	900K	4500	30
LIGHT (REF)	3×10^{-3}			

STORAGE DEVICE ROADMAP



TECHNOLOGY ROADMAP

SPEED — GATES/DIE



ETA-30/1990 APPROACH

SINGLE "MASTER" DESIGN

TECHNOLOGY LEVERAGE

SOFTWARE LEVERAGE

MAXIMUM TOTAL POWER WITH
A MINIMUM NUMBER
OF PROCESSORS

"BRUTE FORCE" PROGRAMMING
OF MULTI-PROCESSOR
APPLICATIONS

ETA-30

- **1 BILLION WORDS**
- **30-40 GIGAFLOPS**
- **3-4 GFLOPS SCALAR**

QUESTIONS AND ANSWERS

Q: What existing operating systems will your new one look like?

M. Fischler

A: We are doing primitives from ground floor. Shells available will likely include VOS for compatability with Cyber 205, and Unix.

Q: Several months ago I asked Neil Lincoln of ETA how the classic problem of processor-memory contention would be handled on your proposed 8 processor system. His answer was that there would be no contention, period. Would you care to amplify or clarify this statement?

C. Maples

A: Obviously there will be contention between processors for memory access as you indicated in your talk yesterday. We believe, however, that by providing a sufficiently high speed bus to shared memory, this contention can be minimized. Conflicts that occur will be arbitrated by the communication processor.

Q: Do you believe that FORTRAN users will be the main group of users for this kind of machine?

H. von der Schmidt

A: Yes

Q: How expensive is the ETH-10 per processor?

A. Charlesworth

A: Less than or equal to a Cyber 205.

Q: Why do you limit yourselves from the start to 8 processors when there are other architectures which have been demonstrated to scale upwards indefinitely?

D. Kaplan

A: Memory contention will become severe if there are more processors attached to the central memory.

Follow-up: But that's a limitation imposed by the architecture you've chosen--these other architectures get around it by not having a central memory.

A: The other limitation is keeping the processors close enough to the memory to maintain the memory bandwidth. The 1990 version will have enough bandwidth to support 16 processors.

Q: What potential customers are you anticipating?

T. Nash

A: Structural Designers - auto, aerospace, nuclear plants
Simulations - aerospace, electronic parts, nuclear physics, oil reservoir
Other - oil (seismic), movie productions, education, weather prediction, national security applications, chemical modeling

Q: What is the mean-time-to-repair (MTTR), given the fact that you have to warm-up the machine to change hardware?

W. von Rueden

A: The MTTR for the CPU's, which are the only elements cooled with nitrogen, is expected to be in the order of 4 to 6 hours.

DIGITAL EQUIPMENT CORPORATION'S PRESENT AND FUTURE COMPUTING ENGINES FOR SCIENTIFIC PROBLEMS

Tom Peterson
Digital Equipment Corporation
1 Iron Way, Marlboro, Massachusetts 01752 USA

[Editors' Note: No paper was received from Digital Equipment, but because of the interest this talk generated, we reprint the questions and answers.]

QUESTIONS AND ANSWERS

Q: What new have you learned, as a marketer, about the needs of high-energy physics from this conference that you did not know before? In particular, your comment that people are unwillling to redevelop algorithms for parallel opportunities seems at variance with what I am hearing.

T. Nash

A: My comment was directed toward the computer-market in general. In fact, high-energy physics is enormously willing to redesign algorithms. This was one of the reasons for seeking out physicists/physics applications in the first place. Also, DEC needs to pay more attention to specialized computer users a la 3081/E, 370/E, etc.

Q: What can you tell us about currently defined members of the Micro-VAX family?

D. Shambroom

A: The only currently defined Micro-VAX is the Micro-VAX I. Any other systems are unannounced other than a direction to continue to develop and enhance the Micro-VAX line.

Q: Are you funding research into more fundamental theory? I suspect that present-day code cannot be partitioned without basic reworking that requires much new theory.

T. Brody

A: Yes, we do, in several university departments.

Q: Are there any **cost effective** DEC products now or before 1990 to upgrade the present VAX 11/780?

C. Bunge

A: Yes, VAX 11/78S upgrade.

Q: What else can you tell us?

A: I can say that we are trying to develop algorithms for automatic decomposition of existing codes for execution on parallel processor systems. Also, I can say about the hardware that the PPA project involves nμ-VAX's, four per circuit board.

Q: This is both a statement and a question. Although you indicated that DEC would not have a commercial multiprocessor system available until about 1990, the experimental prototypes will be fielded considerably before this and members of the scientific community will be able to gain some experience on the machines, learn the directions DEC is proceeding, and be able to feed back suggestions. The question is related to your statement to the effect that the parallel machine would not use VMS. Could you clarify?

C. Maples

A: I said that the parallel machine would not use VMS as it currently exists.

Q: When will people be able to buy the high-end VAX, what about the low-end VAX's?

W. von Rueden

A: High-end VAX's are as yet unannounced and would require specific non-disclosure to release such information. Micro VAX I is available today.

Comment: Physicists build hardware **and** write software today, because they do not find the solution in industry. They would be happy to buy hardware even without software, if it can be found for reasonable prices!

W. von Rueden

FACOM VECTOR PROCESSOR SYSTEM:

VP-100/VP-200

---- CURRENT STATUS AND PERFORMANCE MEASUREMENTS -----

KENICHI MIURA

FUJITSU LIMITED,KAWASAKI, JAPAN

This short paper is supplementary to the one presented at Padova conference in 1983 (1). In this paper, we will describe the recent results of performance measurements,algorithm development and the installations of the FACOM Vector Processor System,VP-100 and VP-200.

1. Performance Measurements

Some results of the benchmarking of FACOM Vector Processor System are summarized in Table 1 and Table 2. Table 1 shows the scalar and vector performances (in MFLOPS) of VP-100 and VP-200 for the 14 Livermore kernels. It should be noted that none of the original loops have been modified for this measurement. Table 2 compares the scalar and vector execution times (in Seconds) of some application programs which have been developed by Fujitsu. The first three are either taken from subroutine packages or part of programs, and two others are complete programs. They are all written in FORTRAN. Some other results of benchmarking of FACOM Vector Processor System have also been reported in Ref.(2).

2. Algorithm Development

We are continuing our efforts to establish the methodology for selecting vector algorithms which are suitable for FACOM Vector Processor architecture. We have obtained very high performance algorithms for FFT,matrix triangularization,random number generation, etc. Detailed analysis of these algorithms will be reported in a forthcoming paper(3). We are also investigating the vectorization algorithms for Monte Carlo calculation in the Lattice Gauge theory.

3. Currently operational VP Systems

Table 3 summarizes the sites, models and other information of the VP Systems which are operational. We expect several more installations in 1984.

References (1) Miura,K.:FACOM VECTOR PROCESSOR VP-100/VP-200,in Proceedings of Three day In-depth Review on the Impact of Specialized Processors in Elementary Particle Physics, Padova, Italy, March 23-25,1983

(2) Mendez,R: SIAM News, Vol.17, No. 2 (March 1984)

(3) Matsuura,T. and Miura, K.:Supervector Performance without Toil, FORTRAN Implemented Vector Algorithms on the VP-100/VP200, to be presented at VAPP II Conference, August 1984.

Table 1. PERFORMANCE MEASUREMENTS OF VP-100/VP-200
(14 Livermore Kernels)

Loop No.	Scalar	Vector VP-100	Vector VP-200	Loop No.	Scalar	Vector VP-100	Vector VP-200
1	10.1	187.1	331.4	8	13.3	86.1	90.4
2	11.1	104.7	180.4	9	12.7	160.8	260.8
3	7.8	174.7	338.2	10	7.8	50.0	85.9
4	5.7	73.9	88.1	11	4.8	4.8	4.8
5	10.0	10.0	10.0	12	4.8	59.1	115.3
6	9.5	9.5	9.5	13	2.5	6.0	6.2
7	14.0	189.2	331.0	14	5.8	12.9	13.8
Arithmetic Average					8.6	80.6	133.3

(Units: MFLOPS)

Table 2. PERFORMANCE MEASUREMENTS OF SOME APPLICATION PROGRAMS ON VP-200

Program No.	Description of Programs	Computation Time Scalar (Sec.)	Computation Time Vector (Sec.)	Performance Ratio
1	Matrix Multiplication (Order: 100)	307.66	4.08	75.4
2	Linear Equation Solver			
	(Order: 100)	.141	.01	14.1
	(Order: 256)	2.27	.056	40.6
3	Self-sorting Type FFT (4096 points, radix 2)	---	.000973	(227 MFLOPS)
4	Molecular Dynamics (High Density Liquid)	144.22	9.97	14.5
5	Simplified Marker and Cell (Poiseuille Flow)	137.0	15.8	8.7

Table 3. Currently Operational VP Systems

Site	Model	Main Storage Capacity	Front-end Processor	Installation
1. Institute of Plasma Physics	VP-100	32 MB	M-200	Dec. '83
2. Computing Center Kyoto Univ.	VP-100	32 MB	M-380	April '84
3. Computing Center Fujitsu Limited Numazu Works	VP-200	64 MB	M-160 M-200 M-382	March,'83

QUESTIONS AND ANSWERS

Q: Is Fujitsu optimistic about the users' willingness to change algorithms to gain the increase in cost effective available in vector processors?

M.J. Levine

A: 1) There is a trade-off in effort vs. performance. If, with no or very little tuning, user can obtain 3x improvement, for example, he will be generally happy.

2) For tuning, Fujitsu supplies, besides powerful compiler, interactive vectorizer, debugging aids. Therefore algorithm change would not be so difficult.

THE CYBERPLUS MULTIPARALLEL PROCESSOR SYSTEM

Vito Bongiorno
Control Data Corporation, Minneapolis, Minnesota 55440

INTRODUCTION

The CYBERPLUS multiparallel processor is the first in a series of multiparallel processors from Control Data Corporation. The CYBERPLUS processor provides a high-speed scalar capability for scientific and business applications. Expanded system performance may be achieved by adding up to 63 additional CYBERPLUS processors.

At a time when applications problems are growing at an alarming rate, the solutions required to keep pace with technology in a number of industries are beyond the reach of current systems hardware designs. The CYBERPLUS processing system provides a bridge into the next generation of applications required to address these growing needs. Utilizing a ring architecture, the CYBERPLUS system provides a multi-parallel capability designed to provide a solution for

those applications that we have not yet dared to develop; a total solution to the applications problems of this century and the next.

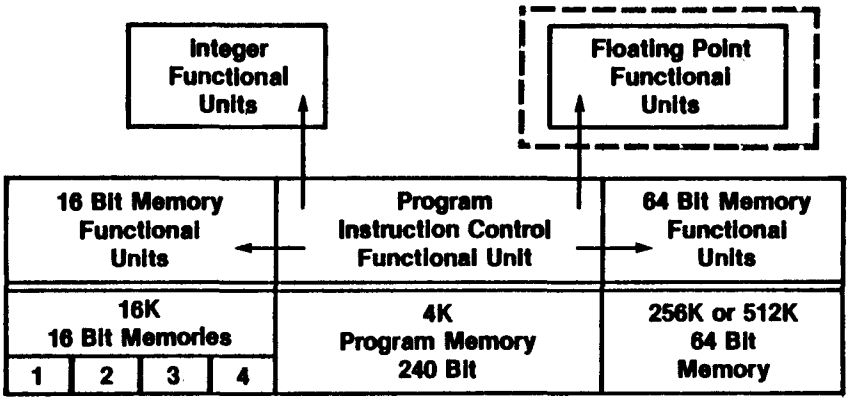
The material presented here covers the basic concepts of the CYBERPLUS processor and the capabilities inherent in the features of this multiparallel processing system.

CYBERPLUS PROCESSOR

The CYBERPLUS processor has 15 independent functional units that execute in parallel in a 20 nanosecond cycle time. Each functional unit is cross-bar connected to the other functional units so output from a functional unit can be input to several other functional units at the same time. A major ingredient of the CYBERPLUS systems is that each and every functional unit can be initiated by the functional control unit every cycle.

The heart of the CYBERPLUS system is the program instruction control functional unit. As in conventional machines, this unit reads instructions from program memory and decodes the instructions into executable statements within the processor. To improve the performance of the processor, each instruction word can initiate 1 to 17 functional units, all in a parallel mode of operation. See Exhibit #1.

CYBERPLUS PROCESSOR



- **Program Instruction Control Function Unit**
  - **— Executes every cycle**
  - **— Initiates functional units in parallel**
- **Functional Units Initiate Every Cycle**

Exhibit #1

CYBERPLUS RING ARCHITECTURE

Interconnection by a dual ring structure provides a high performance interchange of data and control information between CYBERPLUS processors. The CYBERPLUS ring interconnect architecture contains two independent rings, Each provides fo transfer of a ring packet around the circular ring every machine cycle. A ring packet contains 16 bits of data and 13 bits of control information. For expandability, the dual CYBERPLUS rings support up to 16 CYBERPLUS processors.

The CYBERPLUS ring provides the application three interconnect functions. The direct address provides for the direct transfer of data and control into any other CYBERPLUS PROCESSOR ON THE RING. This technique eleminates several of the normal handshaking conventions required in a typical multiparallel processing system. An indirect address provides a queue-driven system where a CYBERPLUS processor puts information into a queue for another CYBERPLUS processor on the ring. The broadcast capability is probably the most intriguing aspect of CYBERPLUS. A CYBERPLUS processor can communicate the same information to any number of CYBERPLUS processors on the ring using the broadcast structure. If the application needs to send the same information to all 15 CYBERPLUS processors on the ring, the application merely adds to the ring packet the address or the processor number for all the processors that are to receive the data.

Using the ring connection, each CYBERPLUS processor can read and write ring packets every machine cycle. Data moves around the ring in a circular fashion so it takes one machine cycle to transfer a ring packet to an adjacent CYBERPLUS processor. Interprocessor delays can be reduced by making one ring clockwise and one counterclockwise. Since each processor is connected to the dual rings, the CYBERPLUS application task can put two separate ring packets onto the dual rings every machine cycle. Each ring can accept a different packet of information from each of the 16 CYBERPLUS processors every machine cycle. The ring transfer rate is 800-Mbit of 16-bit data, and with 16 processors and two rings, the dual ring provides a total transfer of 25,600-Mbits. (See Exhibit #II)

CYBERPLUS
CYBER CHANNEL CONNECTION

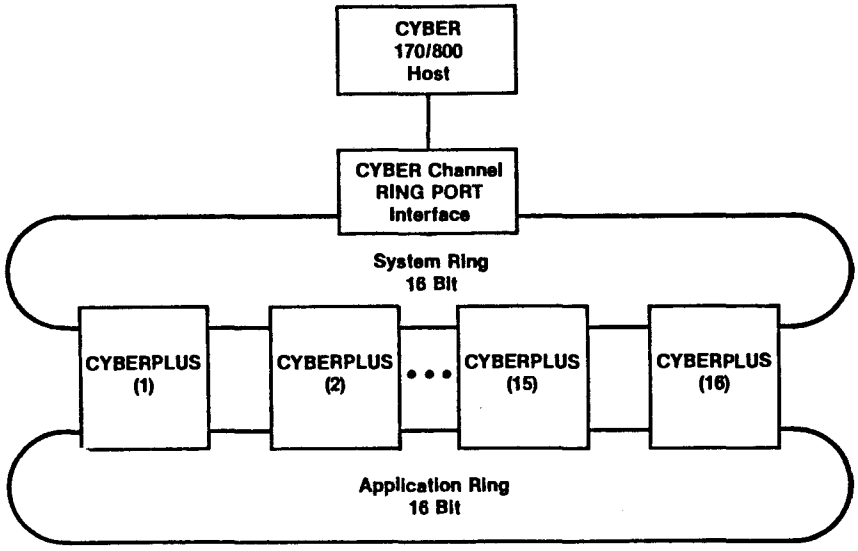


Exhibit II

CYBERPLUS MEMORY RINGS

The first memory ring is the processor memory ring. A CYBERPLUS processor using the 64-bit, 20 nanosecond memory ring reads and writes data between CYBERPLUS processors. Thus a CYBERPLUS processor can transfer 64 bits of data every machine cycle to another CYBERPLUS processor.

The second memory ring is the CYBERPLUS central memory ring. This 64-bit, 80 nanosecond ring transfers 64 bits of data between a CYBERPLUS processor and a CYBER 170/800 host every four CYBERPLUS machine cycles. The CMI (CYBER Memory Interface) supports up to four CYBERPLUS memory rings. A CYBER host can be configured to support up to 64 CYBERPLUS processors. Each of the 64 CYBERPLUS processors can transfer data from a CYBERPLUS to CYBERPLUS memory and from CYBERPLUS to CYBER 170/800 host. (See Exhibit #III & IV)

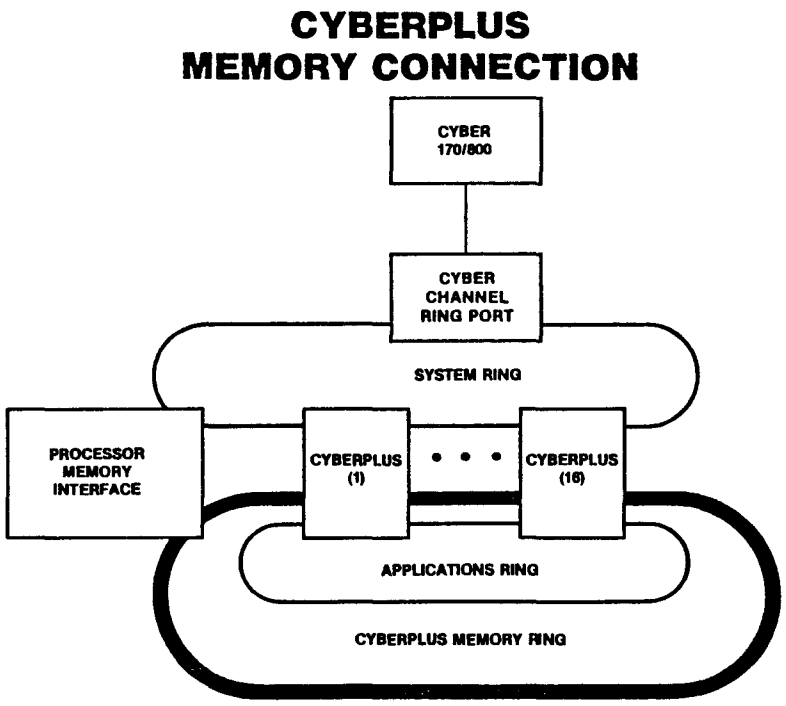

CYBERPLUS
MEMORY CONNECTION
CYBER
170/800
CYBER
CHANNEL
RING PORT
SYSTEM RING
PROCESSOR
MEMORY
INTERFACE
CYBERPLUS
(1)
CYBERPLUS
(16)
APPLICATIONS RING
CYBERPLUS MEMORY RING

Exhibit III

CYBERPLUS
CYBER MEMORY CONNECTION

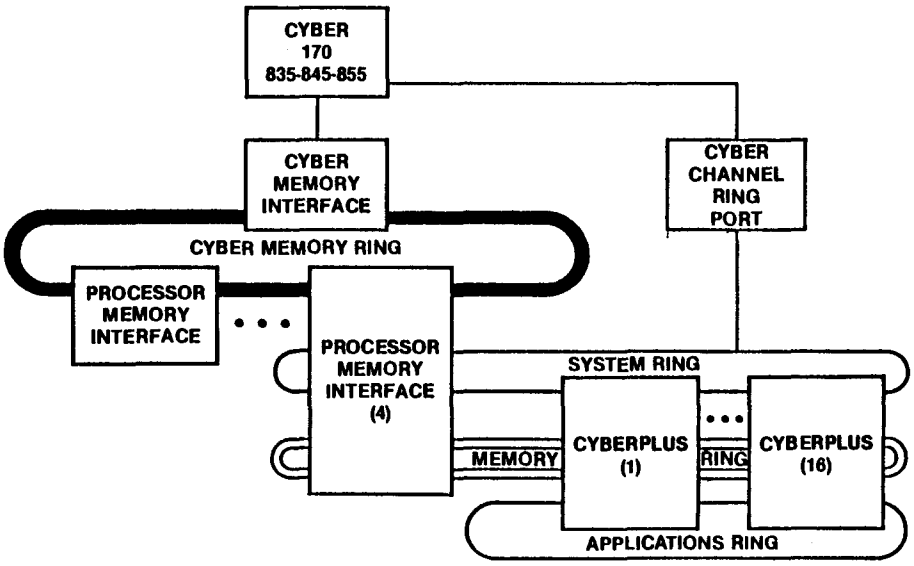


Exhibit IV

CYBERPLUS COMPUTATIONAL EXPANDABILITY

Computational power can be increased by adding CYBERPLUS processors within the ring architecture. A single CYBERPLUS processor can execute at a rate of 650 mips and, by adding the floating point option, 62.5 megaflops in 64-bit mode of operation or 103 megaflops using the 32-bit options. Additional CYBERPLUS processors can increase the overall performance capability. A 64 CYBERPLUS processor system would provide over 44,000 mips and four gigaflops in 64-bit mode of operation. The 32-bit option provides over 6.4 gigaflops.

CYBERPLUS SOFTWARE

The CYBERPLUS software supports the protocol that allows CYBERPLUS processors to communicate with the CYBER 170/800 host via the channel and memory interface connections. There are five software products: 1) System Software, 2) CYBERPLUS Cross Assembler, 3) CYBERPLUS Simulator, 4) CYBERPLUS Cross ANSI 77 FORTRAN, and 5) Debug Facility and other utilities. (See Exhibit #V)

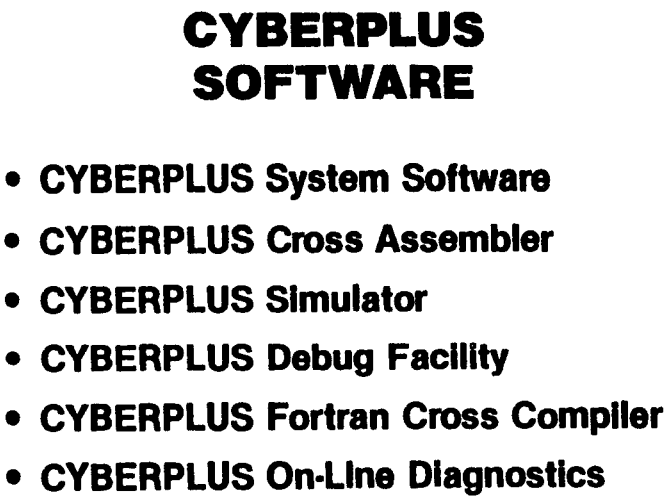


Exhibit V

SUMMARY

The CYBERPLUS Systems provide the computation and expandability to now address the total problem and not require the application designer to scale down the task to fit the computer system.

QUESTIONS AND ANSWERS

Q: Do you plan to use dynamic repartitioning of tasks between processors when one of them fails?

T. Brody

A: Yes.

Q: Was your FFT example written in FORTRAN or hand-coded, and how good will the FORTRAN compiler be at keeping all the functional units busy?

I. Gaines

A: Hand-coded. Would hope to do as well as 30-40% utilization of the functional units.

Q: The Cyber plus system seems more powerful than your subsidiary's machine and also available sooner. How will they be able to compete?

D. Kaplan

A: You should ask the previous speaker!

Follow-up: But will this be more expensive than the ETA? How much does it cost?

A: Of course the Cyberplus is built using old technology, since we just took the AFP design and fixed the important parameters, for example, fast memory is 4K ECL RAM's, and then there's 16K 55ns MOS RAM, just as in the 205. A base processor will cost $750K, and there is a volume discount!

Q: Will it be possible to use a 170/815 host?

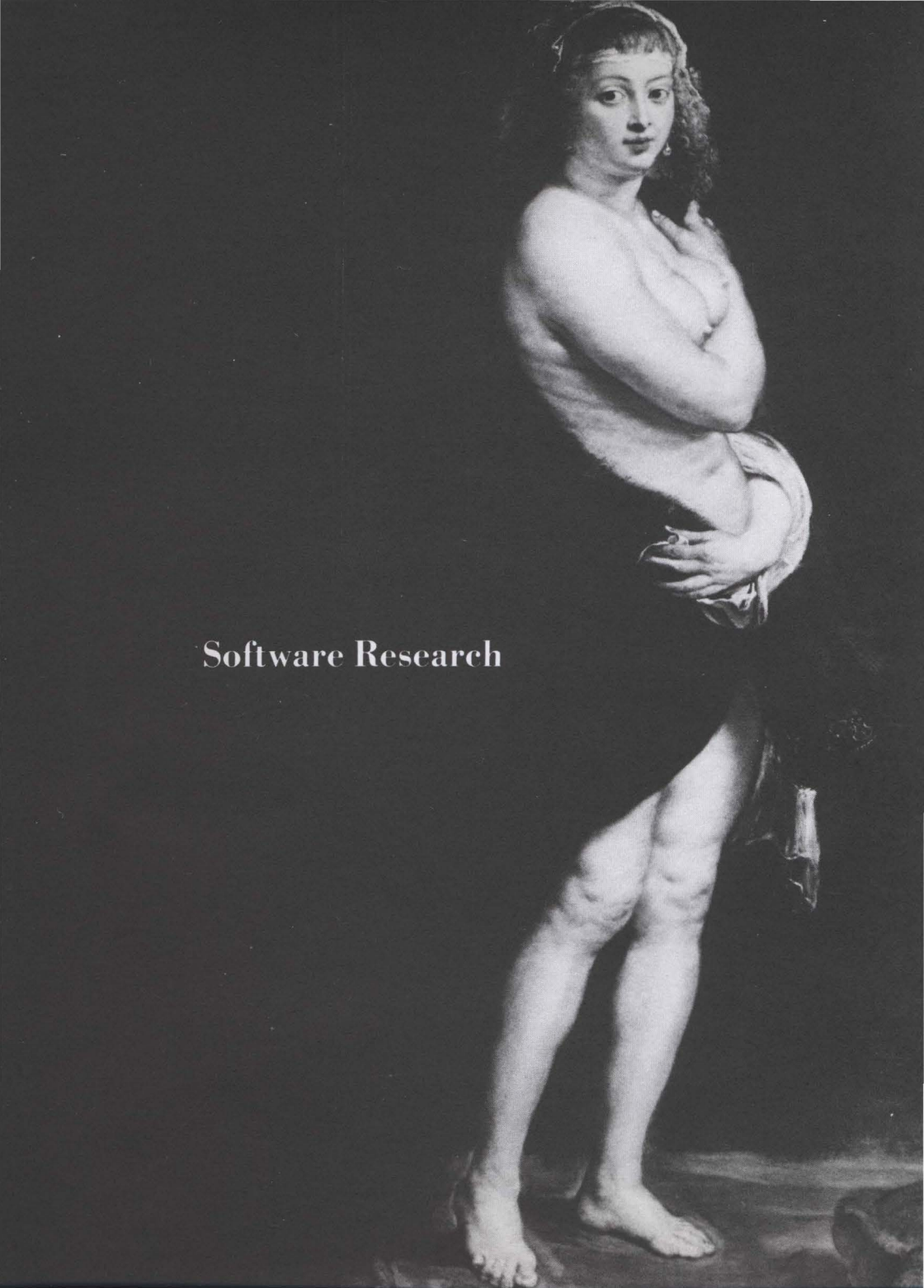
C. Bunge

A: Yes, channel connected.

Q: Statements I have heard from NSA users of the AFP indicated that they had achieved the equivalent of 10 times the CRAY-1 performance with 8 AFP's. They also indicated, however, that "the system could not be considered user friendly" and they were tired of doing all coding in assembly language.

C. Maples

A: We are developing a FORTRAN compiler for the system.



Software Research

[The Pelt Skin, 1638, Peter Paul Rubens, 1577-1640]

A MOLECULAR MECHANICS WORK STATION FOR PROTEIN CONFORMATIONAL STUDIES

*R. Fine, C. Levinthal, Columbia University;
B. Schoenborn, G. Dimmler, C. Rankowitz,
Brookhaven National Laboratories

Interest in computational problems in Biology has intensified over the last few years, partly due to the development of techniques for the rapid cloning, sequencing, and mutagenesis of genes from organisms ranging from E.Coli to Man. The central dogma of molecular biology, that DNA codes for mRNA which codes for protein, has been understood in a linear programming sense since the genetic code was cracked. But what is not understood at present is how a protein, once assembled as a long sequence of amino acids, folds back on itself to produce a three dimensional structure which is unique to that protein and which dictates its chemical and biological activity. This folding process is purely physics, and involves the time evolution of a system of several thousand atoms which interact with each other and with atoms from the surrounding solvent. Molecular dynamics simulations on smaller molecules suggest that approaches which treat the protein as a classical ensemble of atoms interacting with each other via an empirical Hamiltonian can yield the kind of predictive results one would like when applied to proteins.

Such an undertaking is extremely computationally intensive, even for a small protein. Time steps in molecular dynamics calculations have to be small compared with typical vibrational periods in proteins (~.1 ps). Proteins require milliseconds to fold. Even using cutoffs, each step requires the evaluation of ~1E5 pairwise interactions; on our VAX 780, each interaction takes 150 us to evaluate. Simulating the folding of an entire protein would thus require several thousand VAX years. More tantalizingly feasible is the possibility of calcualting the final configuration of a portion of the protein, given constraints from knowing the configuration of the remainder of the protein. Such problems arise in the study of, for example, antibody molecules, and could be used to predict the results of changing small groups of base pairs in the gene for a given antibody molecule using recombinant DNA to produce something entirely new. Such studies are still uncomfortable on a VAX; since these types of studies are best done interactively, we have undertaken the design of a special purpose processor to gain several orders of magnitude in computing power which can be used as the centerpiece of a a work station for protein conformational studies.

The tools which are applied to the system described above are energy minimizations, to find local configurational minima around an initial starting configuration; molecular monte carlos, to build up a description of the entropy of a minimal configuration; and molecular dynamics, to describe the time evolution of portions of a molecule or its docking to a substrate. The structure of all of these calculations is similar, and consists of three parts. The most time intensive of these is the calculation of the total conformational energy and the vector forces on each atom. Once the 3N component force vector is formed, a second part of the calculation consists of updating the 3N coordinates of the atoms given a knowledge of the forces and total energy. Occasionally, a new list of atoms which are close enough to each other to be included in the calculation is generated. The hardware approach we have taken mirrors this software architecture (Figure 1). There are two home-built modules, one of which prepares the list of neighboring atoms and one of which calculates the total energy and vector force on each atom. The positional updating is done in a commercial array processor, which recieves the energy and vector forces via a fast DMA link. Only that portion of the inner loop which is the most time intensive and stable is frozen in hardware; the implementation of the coordinate update in a programmable array processor allows flexibility in tailoring the integration or minimization technique to a specific problem.

*Presenter

The overall schematic of the energy and force calculator is shown in Figure 2. The architecture adopted is a parallel, synchronous pipeline. Each element in the machine is a multiplier, an adder, or a table lookup. All arithmetic is done in 32 bit floating point, and is accomplished by utilizing the Weitek chip set. The clock cycle is presently 125 ns, to be updated to 100ns by chip replacement before completion of the project. Each clock cycle an entry corresponding to an atom is fetched from a Pair List Memory. The structure of the pair list is such that every i atom entry (slow moving index of an ij pair) is followed by all j>i atom entries (fast moving index) in the list. Each entry (32 bits total) consists of a coordinate address, used to look up the coordinates of that atom; a pair type (j atoms only), which identifies the type of interaction which that j atom forms with the most recent i atom; and an ij bit, which identifies the atom as an i or j atom. The coordinate adresses are fed simultaneously to three coordinate memories (x, y, and z). These coordinates are passed to one of the inputs of three subtractors. If the atom is an i atom, nothing happens (an empty cycle is generated). If the atom is a j atom, the subtraction is done to form delx, dely, and delz. These are summed to form delr**2, which is used as input to a table lookup to obtain the value of an interpolation function (which may be a square root to yield r). This, along with the pairtype which describes what kind of pair of atoms this ij pair is, is used after integerization to look up the value of the energy and the value of F/r. The cartesian components of the force vector are obtained by multiplying F/r by delx, dely, and delz, and the total energy and the force vector memories are updated. A note added since the conference: Weitek has released its 1066 register file chip, which should permit the accumulation of forces and energies to be done to 64 bits of precision.

The energy and the value of F/r are both calculated as a sum of two contributions, ie. as E = a(pt)*Ea(pt,r) + b(pt)*Eb(pt,r), to limit the total size of memory. Overall, there is ~4 Mbytes of memory distributed throughout the system. Each of the table lookups is done by quadratic interpolation, as shown in Figure 3. Per bin, the value of the function, the first derivative, and half of the second derivative are stored. The incoming value of the independent variable is scaled, integerized, and split into high and low order bits. The high order bits are added to an offset which is looked up as a function of the pair type and which gives the starting address of the table associated with that pair type. This sum is used to fetch the value, slope, and (half of the) second derivative from the tables, and these are used to compute y = y0 + dx*y' + .5*dx*dx*y''.

All tables are connected to a slow bus (Q bus) which can be downloaded either from the host VAX or a resident 68000. The system is designed so that single step mode is available for debugging, with selected pipeline registers readable from the slow bus. The hardware is being implemented in 6 layer DEC boards (13" x 18"), with interboard connections being implemented through the back plane and via ribbon cable.

The energy and force calculator described above has been designed to evaluate pairwise, central forces between objects in three dimensional space only. In the empirical Hamiltonian which is usually manipulated in molecular mechanics calculations, there are additional terms associated with bond angle bending, torsional excursions, and out-of-plane excursions which are not pairwise and central. The angle bending terms have been centralized by placing a non-Hooks-law spring between atoms 1 and 3 in a 1-2-3 atom triplet which maps out the angular dependence normally used. This introduces a maximum error of 10% at 10 kt. The torsional terms can be handled in one of two ways: firstly, by positioning a point in space away from the 1-2-3-4 quadruplet using atoms 1,2, and 3 and rigid springs and subsequently dropping a non-Hooks-law spring to mimic the torsional potential for excursions of the 4th atom; and secondly, by calculating these torsional contributions in the array processor which would otherwise be idle during the time the force and energy accumulator is working. These calculational times match well for a Numerics MARS 432 and several other processors of its class.

The status of the project is, that construction has begun. The prototyping CAD/CAM station in the Instrumentation Division at Brookhaven is being used presently to lay out the first several boards, which are the quadratic table lookup boards. There exists presently at Brookhaven the inhouse capability for quick turnaround of prototypes in multi-layer boards. These first boards should be plugged in and actively undergoing testing by August. The completion target for the remainder of the project is December, 1985.

This project could not have been concieved or carried out in vacuo in a university biology department, and we are fortunate to be surrounded by individuals at Columbia who are involved in designing special purpose processors. We would like to specifically thank Paulo Franzini and Norman Christ of the Physics department for discussions and encouragement. We would also like to thank Bill Sippach of Nevis for discussions, and Gary Benneson for his initial interaction in discussions of feasibility, layout, and generally providing us with an education. No publication could be complete without mentioning the inexhaustible efforts of David Yarmush, who manages resident wizardry with software in the Biology department at Columbia.

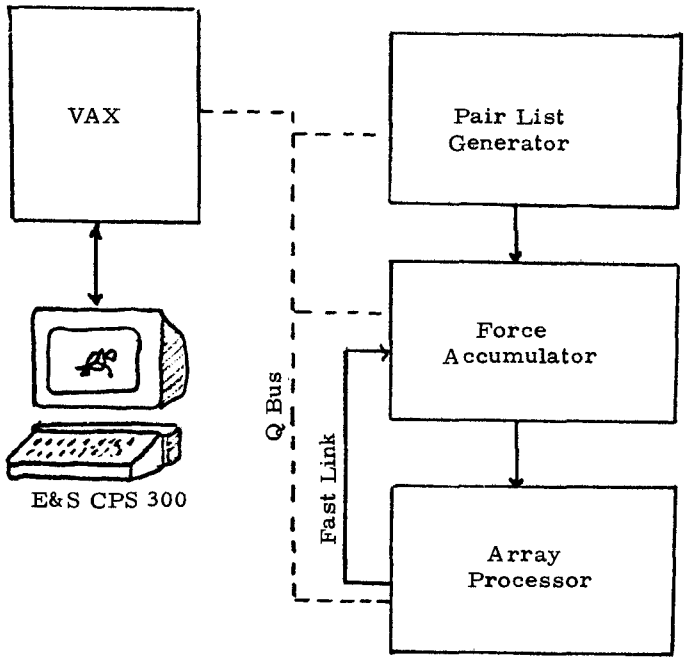


Figure 1.

FIGURE 2.

General schematic of the force and energy calculator.

The processor can functionally be broken into six units:

Unit I: The Neighbor List. This contains the addresses necessary to look up the coordinates of the individual atoms; an ij bit which identifies an atom as an i or a j atom; and a pair type, which indicates the type of pair a j atom forms with the most recently accessed i atom. Each clock cycle, these three atom oriented pieces of information are delivered to the other units.

Unit II: The R Squared Calculation. This unit contains the coordinate memories, and the associated hardware to calculate

$$(xi-xj)^{**}2 + (yi-yj)^{**}2 + (zi-zj)^{**}2.$$

Unit III: The R Calculation. This unit performs a quadratic table lookup to obtain the value of R, given the value of R squared.

Unit IV: The Energy and Force/r Tables. This unit contains the quadratically interpolated tables used to look up the energy and the magnitude of the force/r between a j atom and the most recently considered i atom. The Energy and the Force/r sections are identical, and consist of weighted sums of two contributions: for the Energy,

ACoeff*(Table A) + BCoeff*(Table B).

Unit V: The Energy and Force Accumulators. This unit contains sum- accumulation memories which accumulate the total energies associated with a few selected classes of atoms and the total force components associated with each atom. The components are calculated by multiplying F/r by (xi-xj) to obtain Fx, etc.

Unit VI: This unit updates the variable portion of the Neighbor List occasionally during a run. It uses Unit II to calculate the distance between pairs of atoms, to determine whether they are within a preset cutoff distance for their pair-type. If they are, they are entered into the variable portion of the Pair List.

Throughout the diagram, various delays, either labelled "Delay" or "D", have been indicated. These are either fifo register delays or shift register delays which guarantee the arrival of information pertaining to a given pair at the appropriate input to a unit in synch.

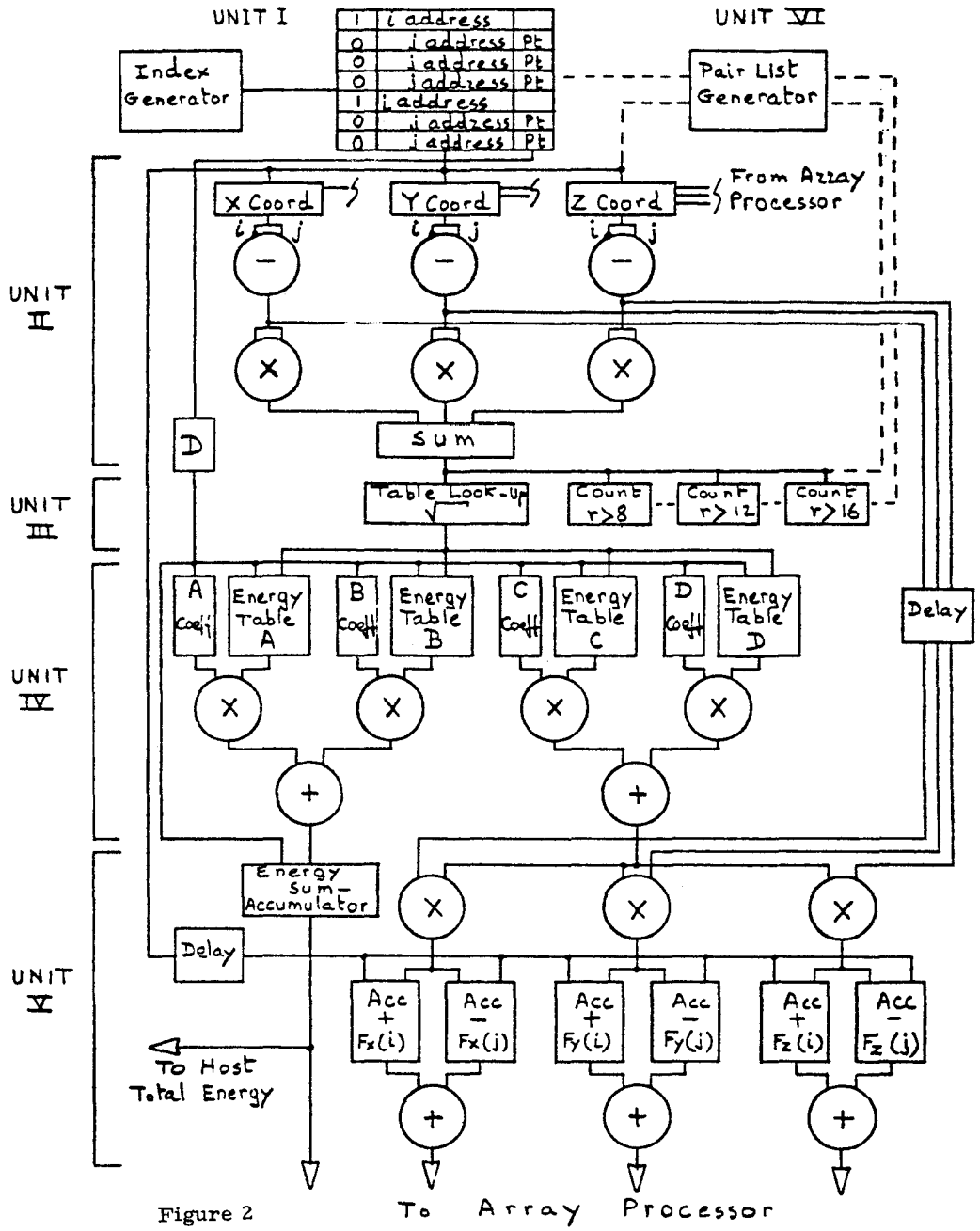

UNIT I
UNIT VI
Index Generator
Pair List Generator
1 i address
0 j address Pt
0 j address Pt
0 j address Pt
1 i address
0 j address Pt
0 j address Pt
From Array Processor
X Coord
Y Coord
Z Coord
UNIT II
UNIT III
UNIT IV
UNIT V
D
Sum
Table Look-Up
Count r>8
Count r>12
Count r>16
A Coeff
Energy Table A
B Coeff
Energy Table B
C Coeff
Energy Table C
D Coeff
Energy Table D
Delay
Energy sum-Accumulator
Delay
Acc + Fx(i)
Acc - Fx(j)
Acc + Fy(i)
Acc - Fy(j)
Acc + Fz(i)
Acc - Fz(j)
To Host Total Energy
To Array Processor

Figure 2

FIGURE 3.

Schematic of the quadratic interpolators used to look up the values of the energy, the force/r, and the square root (note that only a portion of the hardware is used for the square root). The central memories, y0, y', and y'', contain tables of the functional values, slopes, and 1/2 times the second derivatives of the desired function. The beginning of the appropriate table for a given pair type is stored in an offset memory. This offset is added to the high order bits of the incoming (integer) indepedent parameter, and the result is used to address the central memories. This value is represented by the letter x in the figure. The adders and multipliers form the combination

$$y0 + dx*y' + dx*dx*(1/2*y'')$$

where dx consists of the low order bits of the independent parameter. There are also two memories, Max and Min, which store the maximum and minimum addresses of the tables associated with any given pair type. If the generated address falls outside of this range, an "invalid bit" is set.

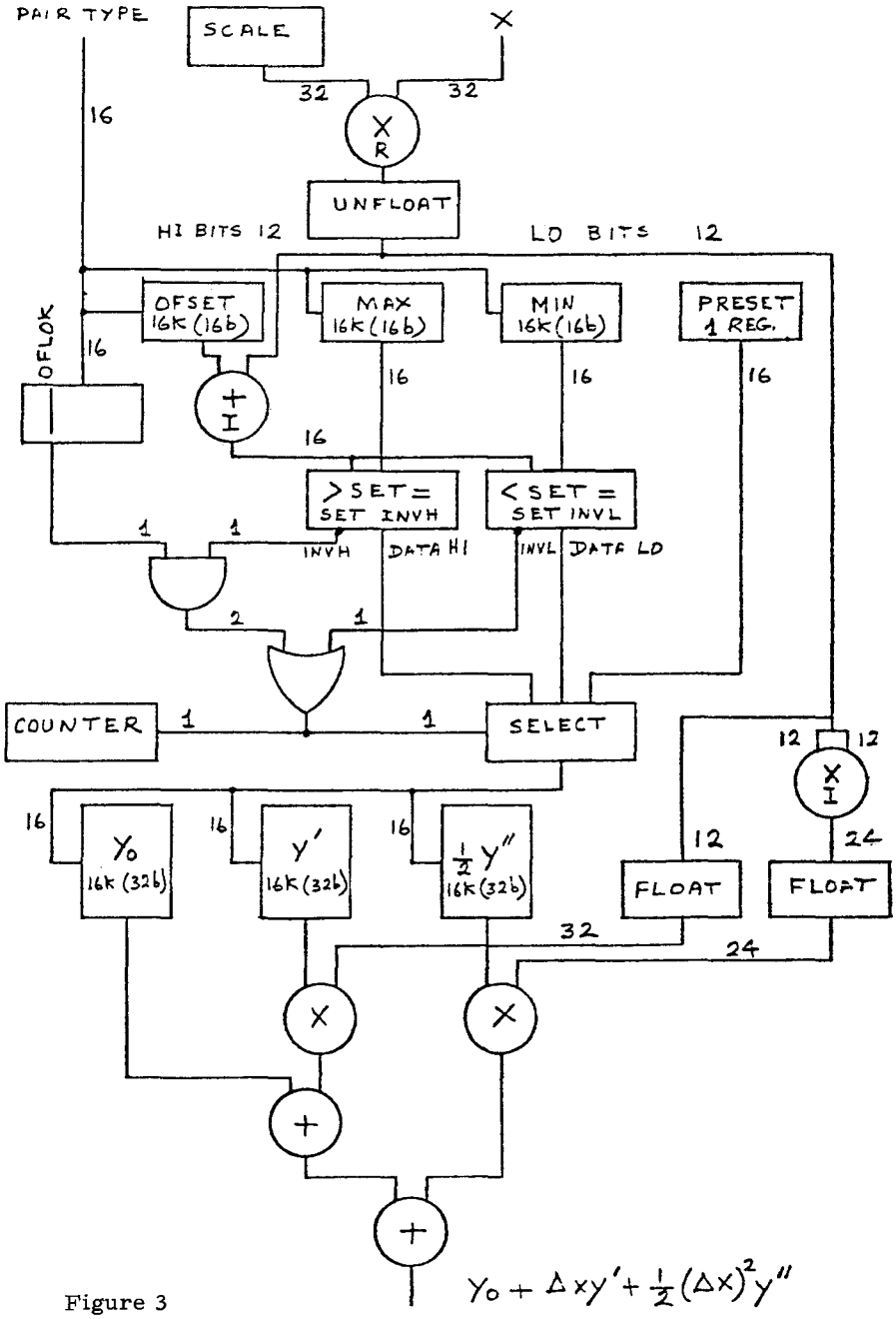

PAIR TYPE
SCALE
X
32
32
16
X R
UNFLOAT
HI BITS 12
LO BITS 12
OFLOK
OFSET 16K(16b)
MAX 16K(16b)
MIN 16K(16b)
PRESET 1 REG.
16
16
16
16
16
+ I
>SET= SET INVH
<SET= SET INVL
1
1
INVH
DATA HI
INVL
DATA LO
2
1
COUNTER
1
1
SELECT
12
12
X I
16
16
16
Y0 16K(32b)
Y' 16K(32b)
½ Y'' 16K(32b)
12
24
FLOAT
FLOAT
32
24
X
X
+
+
Y0 + Δx y' + ½ (Δx)² y''

Figure 3

QUESTIONS AND ANSWERS

Q: How much to replicate processor?

J. Amann

A: $100K

Comment: That fraction of the biological community which needs supercomputing, or fast vector processing, is in fact more than willing to rewrite code or to pursue hardware development to get what they need. What reluctance there is to changing code comes typically from users who need low level, quick results from scientific packaged subroutines to evaluate results from low statistics experiments. They don't need supercomputing, and hence should not be considered in evaluating how eager potential supercomputer users in general science are to adapt to new architectures. As is true in HEP, when a biologist needs to adapt, he will.

R. Fine

COMPUTATIONAL BOTTLENECKS IN MOLECULAR ELECTRONIC STRUCTURE CALCULATIONS

Carlos F. Bunge

Instituto de Física, Universidad Nacional Autónoma de México,
Apartado 20-364, Delegación Alvaro Obregón, 01000 México D.F., México

Abstract

After briefly reviewing the role of quantum chemistry in contemporary science, various software and hardware requirements in ab-initio calculations of molecular electronic structure are discussed. Current efforts to develop an alphabet and a vocabulary of high-level programs to allow people versed in computational molecular physics to write and debug their own codes in a matter of days are reported. The repeated computation of millions of two-electron molecular integrals involving the interelectronic potential and complicated orbitals centered in up to four different locations requires a small data set, gigaflop capabilities, and it is amenable to extensive parallel processing. Four-index transformations and the calculation of eigenvectors of huge sparse and symmetric matrices are best implemented with matrix multiply operations (and concomitant large data bases), without which the required sustaining I/O rates would exceed present technical possibilities, except for slow single-processor scalar machines. Key-sort of large lists of small items requires scatter and gather operations on 4-64 Mbytes memories at about N/20 Mmoves per second or better, for N Mflops of processing speed. The matrix multiply operation emerges as the unifying concept to impel the development of parallel and vector processing. Other desirable advances are discussed.

1. Introduction

Recently, the Computer Systems Division of Gould Inc. sent the following memorandum to many quantum chemists around the world:
"If you could receive the undivided attention of a major computer manufacturer for a one-hour conversation about needed developments in Computational Quantum Theory of Matter, what five items would you emphasize and why?"

In answering this welcome initiative, after a brief review on the computational background of quantum chemistry, I will focus on present-day bottlenecks in molecular electronic structure calculations. Finally, I will give my opinion on what I consider to be the most needed developments in computer-related areas affecting quantum chemistry.

2. Evolution of quantum chemistry

Quantum chemistry is the corpus of theoretical knowledge generated by considering Schrödinger's equation:

$$H\Psi = i\hbar(\partial/\partial t)\Psi \qquad (1)$$

as the starting point for any discussion on the Quantum Theory of

Matter. Formidable computational problems surrounding efforts to solve (1) have shaped the evolution of quantum chemistry. Any attempt to find approximations to (1), whether a variational or a perturbative method is employed, must face from the outset the problem of calculating many two-electron molecular integrals (pq/rs) involving the interelectronic potential r_{12}^{-1} and atomic orbitals ϕ_{pA}, ϕ_{qB}, ϕ_{rC}, ϕ_{sD}, centered in usually different locations A, B, C, D, respectively:

$$(pq/rs) = \int\int \phi_{pA}(\vec{r}_1)\phi_{qB}(\vec{r}_1) r_{12}^{-1} \phi_{rC}(\vec{r}_2)\phi_{sD}(\vec{r}_2) d\vec{r}_1 d\vec{r}_2. \qquad (2)$$

In the *semiempirical* approach,[1] approximations to (pq/rs) are dictated by computational expediency, such as the arbitrary zeroing of most of them, or the parametrization of some of them to fit experimental data with respect to a *well-defined* model wave function. Such calculations are of central importance in pharmacology, the dye industry, plastics' manufacture, and the synthesis of many new organic molecules of relatively small size.

In the *ab-initio* approach,[2] approximations to (2) are based on rigorous mathematics, so that approximate solutions to (1) can be systematically improved upon, by conceiving a reasonable sequence of increasingly accurate models for the wave function expansion, and by regulating various thresholds to neglect sufficiently small intermediate quantities, in accord with the desired precision. Accurate bond lengths and bond angles, barriers to rotation and inversion, and inner-shell electron excitation energies are in the realm of current ab-initio methods. Increasingly complex and accurate molecular energy surfaces and valence-shell electron spectroscopy are two of the areas where substantial progress is under way. Just to mention some of our own work on atoms, the calculation of energy levels to spectroscopic accuracy has led to the discovery and characterization of new states relevant to UV and soft X-ray laser design.[3]

3. Software bottleneck

The community of quantum chemists is estimated at 8000-10000 members, however, about half of them are marginal computer users or non users, especially in localities with poor access to computer resources. On the other hand, the huge Fortran programs currently in use (ranging from 5000 to 100000 statements with an average of 15000-25000) have been written by hardly more than 300 people all along the last 20 years. Thus most quantum chemists *select options* in programs they have not written, and run them with little or no modifications. As *bright* students find little motivation to be passive computer users, they mostly end up doing clever semiempirical work, or they plunge into heavy and specialized programming, viz., they hardly do any decent science at all. In order to get more and better people to actively participate in the formulation of quantum chemistry problems it appears necessary to develop an alphabet and a vocabulary of robust and efficient quantum chemistry programs, so that any worker versed in computational quantum chemistry could write and debug his (her) own codes in a matter of days. An outline of my ideas is given next.

A *letter* is a subprogram that cannot be meaningfully subdivided in natural sciences or social sciences contexts. It operates on well-defined data structures by means of logical and/or mathematical operations of general character, according to one or several options. For example, a subroutine to calculate eigenvalues and eigenvectors of a real and symmetric matrix, or a subroutine to order a list of items according with a list of key values, are letters in scientific contexts. Of course, each of these subroutines are usually subdivided into many other subroutines which themselves may be *letters* at another level, say, in a general computer sciences context.

A *phonem* is a subprogram which selects a letter within a set of different letters which resemble each other in the following sense:

(i) they operate on the same data structures,
(ii) they generate the same logical results, and
(iii) they produce numerical results which, in principle, can become as close from each other as it may be desirable, just by controlling certain parameters. For example, a subprogram which selects a given subroutine to diagonalize real and symmetric matrices. In its simplest form a phonem is a single letter. Generally, phonems have applications in several fields of science.

A *word* is a well-defined sequence or a string of phonems, each one of them appearing with *all* their options explicitly set. Moreover, a word has a well-defined meaning in quantum chemistry; for example, a subprogram that starting from a list of one- and two-electron integrals generates an independent-particle model self-consistent wave function.

Different words with the same (or almost the same) meaning are called *synonyms*. A *synonymizer* is a subprogram that selects a given word within a group of synonyms. For example, a subprogram that chooses one of several methods to carry out the four-index transformation of quantum chemistry. Often it is convenient to have access both to a synonymizer and to an equivalent word made up of a larger number of phonems.

Finally, a *phrase* is a well-defined sequence of synonymizers which when operating over the correct data structures produces results that may be pertinent to quantum chemistry. We are in the process of developing a basic set of efficient, portable and user friendly synonymizers for molecular electronic structure calculations,[4] which, we hope, will facilitate access of computational quantum chemistry to a wider audience, while also contributing to a more universal and cooperative working style, already manifest through the outstanding services of the Quantum Chemistry Program Exchange library set up at the Department of Chemistry of Indiana University in Bloomington, Indiana, U.S.A.

The general use of phonems and synonymizers requires a vast amount of validation checks. Input data and parameters generating intermediate data in different phonems are always checked for compatibility. The few quantities which define all array dimensions in a given subroutine are declared in parameter statements and also appear as arguments to allow immediate validation. Phonems and synonymizers may have arguments to take advantage of possible simplifications depending on the nature of their input data. For example, in the SORT phonem, the argument NOEQKY is zero in the general case. If no keys are equal,

however, a scatter process or an efficient bin sort method will be employed if the range of key values does not exceed the number of items to be sorted by a certain factor. In such instances the user may get substantial savings in execution times by writing NOEQKY in the appropriate argument place. However, if he (she) does so when there are equal keys, impredictable results will be obtained. Although in more critical applications, like dam control or C^3I (command, control, communications and intelligence) work, such risks are inadmissible, computer bound scientific applications such as quantum chemistry must necessarily operate close to the frontier where efficiency and reliabilty may conflict with each other.

4. Hardware bottlenecks

Low I/O requirements. This is the case in subroutines where the number of bytes of both input data and output results is negligible when compared with the number of bytes that must be handled to produce the results. Let us consider the evaluation of the molecular integrals (pq/rs) defined by (2). In our own approach to molecular electronic structure calculations, the atomic orbitals ϕ_{pA} are written as:

$$\phi_{pA} = (\vec{u}_{pA}.\vec{r}_A)^{\ell_p} r_A^{-\ell_p} \phi_{pA}^{HF}(\eta_p r_A)\exp[\chi_p(\vec{v}_{pA}.\vec{r}_A)], \qquad (3)$$

where $\vec{u}_{pA}$, ℓ_p, η_p, χ_p, and $\vec{v}_{pA}$ are parameters specifying orientation, the azimuthal quantum number, scaling, polarization and orientation of the polarization, respectively. Usually, each Hartree-Fock atomic orbital ϕ_{pA}^{HF} is expanded in terms of 3 to 6 gaussian functions:

$$\phi_{pA}^{HF} = \sum_{i=1}^{3-6} \exp(-\alpha_{ip} r_A^2) C_{ip}, \qquad (4)$$

where the orbital exponents α_{ip} and contraction coefficients C_{ip} are fixed parameters. Thus we have up to 20 parameters per atomic orbital after including the three cartesian coordinates defining the location of A. Assuming 400 orbitals (quite a gallant proposition) we have a data base of 8000 double precision numbers to characterize all atomic orbitals, to which we must add 3N atomic coordinates and N nuclear charges. For M atomic orbitals, the number I_M of (pq/rs)'s is $I_M \cong M^4/8$, considering that only integrals with $p \geq q$, $r \geq s$, $pq \geq rs$, have distinct values. For M=40 and 400 we have $I_M = 6.4 \times 10^5$ and 6.4×10^9, respectively. If we assume that the expansion (4) runs up to four gaussian functions G_{ip},

$$\phi_{pA}^{HF} = \sum_{i=1}^{4} G_{ip} C_{ip}, \qquad (5)$$

then (pq/rs) may be written as:

$$(pq/rs) = \sum_{ijkl} (ij/kl) C_{ip} C_{jq} C_{kr} C_{ls} \qquad (6)$$

involving 256 integrals (ij/kl). Since each (ij/kl) requires in average over 200 floating point operations (flop), a single (pq/rs) requires 0.05 Megaflop (Mflop), and the entire list of two electron integrals for M=40 will require 32000 Mflop. On a VAX11/780 computer, with a maximum processing speed of 0.2 double precision Mflop per second, this step of the calculation alone will take about 40 hs of CPU time. Optimization of the parameters in (3) considerably increases Mflop demands, even if not all integrals must be recalculated at each step, and assuming that the orbital parameters have already been optimized in a similar molecular fragment environment. Moreover, the calculation of molecular energy surfaces requires the unavoidable recalculation of *all* molecular integrals.

Preliminary estimates of the magnitudes of each (ij/kl) can be used to neglect them altogether or to choose smaller gaussian expansions in (6). Yet, the problem remains a formidable one. Since all (ij/kl)'s can be evaluated independently from one another, *parallel processing* is applicable. The same comment applies to the (pq/rs)'s. The increased visibility of quantum chemistry in contemporary science and its almost unlimited demands on Mflop seem to justify the development of a gaussian molecular integral chip.

Medium I/O requirements. This is the case in subroutines where the ratio R between the number of both input and output bytes and the number of bytes that must be handled to produce the results is much smaller than one but still large enough to cause a significant I/O overhead. (Although the value of R is entirely determined by the program itself, increasingly large available central memories can make possible better implementations of algorithms at considerably reduced R values.) For example, on a VAX11/780 with slow RK07 disks, the maximum I/O transfer rate in Fortran is about 0.15 Mbytes per second, while the CPU can effectively process up to 3.5 Mbytes per second, viz., any time R is 1/23 or larger, execution times must *necessarily* become I/O bound.

In quantum chemistry, however, R is much larger or much smaller than 1/23, except when one is constrained to very small memories. Nevertheless, if an array processor, such as the FPS164, is attached to a VAX11/780, the situation may change dramatically. Since the FPS164 has a CPU burst throughput of about 192 Mbytes per second, R will become 1/1280 (other things being equal), which is well within the domain of many quantum chemical applications.

Let us consider the evaluation of eigenvalues and eigenvectors of very large sparse and symmetric matrices, needed when going beyond the independent-particle model description. The wave function Ψ is expanded in terms of configurations Φ_K as:

$$\Psi = \sum_{K=1}^{K_{max}} \Phi_K a_K, \qquad (7)$$

where the a_K's are linear coefficients to be determined from

$$\underline{\underline{H}}\,\underline{\underline{a}}_j = E_j \underline{\underline{a}}_j . \qquad (8)$$

Let us consider an expansion (7) including a reference closed-shell configuration of N_o occupied orbitals and all single and double substitutions of them into N_{no} nonoccupied orbitals. Recent developments,[5] allow to solve for the lowest eigenvalue of (8) by means of roughly 2H flop, H being the number of $H_{ij} \neq 0$ in (8); this also includes the calculation of the H_{ij}'s themselves! In Table 1 we give various data pertinent to the present discussion.

Table 1. Mflop needed to solve for the lowest eigenvalue of (8) using a singles and doubles approximation.

N_o	N_{no}	K_{max}	$2H/10^6$ or # Mflop
10	10	5 151	7.3
15	15	25 651	88.9
20	20	80 601	517
40	40	1 282 401	34 899.3

The entry corresponding to an $\underline{\underline{H}}$ matrix of dimension 80601 runs on our VAX11/780 under a *virtual memory system* in 50 minutes of elapsed time, of which 90% were accounted for by the CPU, thus exhibiting the maximum sustaining rate of 0.2 double precision Mflop per second, even on a small working set of 100 Kbytes and with almost no extra central memory for quick virtual memory work.

The last entry of Table 1, which corresponds to an $\underline{\underline{H}}$ matrix of dimension 1282401, should run on the VAX11/780 in about 45 hs, *if* enough central memory is available to sustain the maximum 0.2 double precision Mflop per second rate.

To proceed with this discussion, let us consider the sustaining I/O requirements for a matrix times vector operation, assuming a 12 Mflop per second processor. From the results in Table 2, we see that the I/O

Table 2. Sustaining I/O rate requirements for matrix times vector, as a function of dimension N, assuming a 12 Mflop per sec processor

N	Central memory (Mbytes)	Mflop	time (seconds)	I/O rate
10	8×10^{-4}	2×10^{-4}	17×10^{-6}	50 Mb/sec
100	8×10^{-2}	2×10^{-2}	17×10^{-4}	50
1000	8	2	17×10^{-2}	50
10000	$8\times10^{+2}$	$2\times10^{+2}$	17	50

rate requirements for a matrix times vector operation are quite beyond present technical capabilities for a processor like the FPS164. Even for a VAX11/780, its 0.2 double precision Mflop per second rate demands 1 Mb/sec I/O rates, which are above the possibilities of the fastest disks presently commercialized by Digital Equipment Co., which run at 0.68 Mb/sec though they are advertised to run at almost twice that speed.

We can see that any attempt to formulate (8) as a matrix times a vector is condemned to founder on account of unrealistic I/O rate demands. Our own implementation is based on matrix multiply operations which allow for more favorable sustaining I/O rates, as shown in Table 3.

Table 3. Sustaining I/O rate requirements for matrix times matrix, as a function of dimension N, assuming a 12 Mflop per sec processor

N	Central memory (Mbytes)	Mflop	time (seconds)	I/O rate
10	2.4×10^{-3}	2×10^{-3}	1.7×10^{-4}	14 Mb/sec
100	2.4×10^{-1}	2	1.7×10^{-1}	1.4
1000	$2.4\times10^{+1}$	$2\times10^{+3}$	$1.7\times10^{+2}$	0.14
10000	$2.4\times10^{+3}$	$2\times10^{+6}$	$1.7\times10^{+5}$	0.014

The central role of the matrix multiply operation (MMO) in quantum chemistry has been amply demonstrated by Saunders and coworkers.[6] At about N=1000, as can be seen from Table 3, the MMO appears as a good candidate for implementation with parallel processors supporting vector pipelined capabilities. Even for N=100, the use of 10 parallel processors would demand a not unreasonable I/O rate of 14 Mb/sec.

High I/O requirements. In order to set up (8) it is necessary to transform the (pq/rs) integrals into (ab/cd) integrals over orthogonal orbitals a, b, c, d. If ϕ_a is given by:

$$\phi_a = \sum_p^M \phi_p \alpha_{pa}, \tag{9}$$

then (ab/cd) is calculated as:

$$(ab/cd) = \sum_p \sum_q \sum_r \sum_s [(pq/rs)\alpha_{sd}]\alpha_{rc}]\alpha_{qb}]\alpha_{pa}. \tag{10}$$

The calculation of the entire list [(ab/cd)] requires about M^5 flop for a basis of M molecular orbitals. The most efficient implementation for vector machines[6] uses at least two *sort* processes over large lists. Unless a rather slow CPU processor is run in competence with the fastest I/O equipment, *external* sort will always be I/O bound.[7] Because the required sorts involve lists where the range of (*distinct*) key values does not greatly exceed the number of items to be sorted, the external sort can be replaced by a *scatter* process if enough fast memory of some kind is available. For M=70 orbitals, there are up to 3 million (ab/cd)

integrals requiring a memory of 72 Mbytes to carry out the scatter process. For a total of $M^4/8$ input integrals, $3M^4/4$ Mmoves are needed. Therefore, the four-index transformation of quantum chemistry, Eq.(10), will become I/O bound if the ratio R between the number of Mmoves/sec and the Mflop/sec processor speed is $R \leq 1/M$. A value $R \approx 1/20$ appears as a reasonable compromise when taking into consideration the complete budget of molecular electronic structure calculations. It should be stressed that the requirements on R are inversely proportional to M, so that the *full use* of very large *intermediate speed* memories, say, of 10 Gbytes, would be entirely adequate for the present purposes with R as low as 1/100.

5. Final considerations

Most quantum chemists, not being electrical engineers nor computer scientists, are *captives* of computer manufacturers whose commercial policies are addressed to a still more defenseless crowd of unsophisticated users. Ab-initio calculations are made with highly regular, predictable codes emphasizing processor and I/O rates. We need good programming development facilities, efficient matrix multiply operations and concomitant sustaining I/O rates which when put together demand 4-24 Mbytes of fast central memory. Our quest to get more Mflop/peso(dollar) finds its motivation in lessening the ever growing frustrating ordeal of quantum chemistry props behind gloomy TV screens. As old habits die hard, we also promise semiempirical investigations on *larger* systems and *better* ab-initio calculations for a fixed cost.

What are the best options to upgrade our equipment? Today's worst option for VAX11/780 *owners* is certainly to buy another VAX, or the 782 or 784 models offered by DEC. The best option for *poor* owners seems to be the MAP-6420 64-bit array processor selling at about US $100,000 and which might upgrade a VAX11/780 by a factor of about ten in *Fortran language*. From *open floor discussions* at this Conference, it seems that we will not hear good news from Floating Point Systems on 64-bit array processors until late 1985 or early 1986, while DEC is not promising any cost-effective upgrading equipment until early 90's!

Burroughs' announcement in 1980 of the cancellation of its scientific array processor project spelt widespread discouragement among university computer centers in Latin America and overseas, as their B6000 and B7000-series mainframes became overnight white elephant dead ends. Nevertheless, at least two National Universities (Costa Rica and Uruguay) have recently acquired B6900 mainframes, adding support to my opening sentence above.

Other institutions (such as Instituto Politécnico Nacional and Universidad Autónoma Metropolitana in México City) made *heavy* investments in CYBER170/815 computers. Until recently, the 170/815 could not compete with the virtual memory capabilities of the half-priced VAX11/780, and could only attain a modest parity with the same VAX in programs conditioned by floating point operations. After CDC's announcement of virtual memory last April, the expensive CDC170/815 became potentially ahead of the VAX11/780 because of superior I/O capabilities. Faithful to expectations, CDC announced last May the *Cyber Plus* processor, which together with minor additional equipment upgrades the CYBER170/815 by

an average factor of 100, with a peak burst upgrading factor of about 3000 (for a single processor; up to 64 Cyber Plus parallel processors may be ordered). Thus, CYBER170/815 *owners* appear as prime candidates for entering the supercomputer age in Latin America, after an investment of US $800,000.

I consider virtual memory an essential asset, not as a pretext for sloppy programming but rather as an indispensable tool in connection with straightforward maintenance of 100000 statement codes.

Quantum chemistry needs to develop a *basic vocabulary* of computer programs to enable every student to write and debug his (her) own codes in a few days. In this way, from about one hundred *active* program writers we might grow to about 5000, which in time should pull in another 30000 chemists deeply interested in chemical *problems*.

In large urban centers such as México City, where quantum chemistry projects are pursued in many groups located at 9 Departments scattered in 4 Institutions, it should be feasible and highly benefical to create, stimulate and support a computer network, especially for updating and enlarging the program libraries.

As economical times go, it is clear that meaningful access to supercomputers for scientists from underdeveloped countries will only materialize through personal arrangements with supercomputer centers. Perhaps one of our bold administrators decides to patronize the noble cause of upgrading a CYBER170/815 with a Cyber Plus processor (not with a /825 model, please!). Meanwhile, in view of inexorable prospects anticipating a widening gap of computer resources with respect to the advanced countries, we find a stimulating challenge in seeking new ways of inquiry so that we can persist to accompany and to contribute to the development of computer-bound areas of science.

Acknowledgments

I have received all sorts of enlightenment, personally and otherwise, from colleagues at home and abroad. I am particularly indebted to Tomás Brody, Annik Vivier Bunge, Gerardo Cisneros, Alejandro Jiménez, Enrique Poulain, Tomás Seligman and Manuel Urraca for their unfailing example and generous advice.

References

1. G.A.Segal (Ed.) Modern Theoretical Chemistry, Vols. 7 & 8: *Semiempirical Methods of Electronic Structure Calculation*, Plenum, New York (1977).
2. H.F.Schaefer (Ed.) Modern Theoretical Chemistry, Vols. 3 & 4: *Methods of Electronic Structure Theory*, Plenum, New York (1977).
3. C.F.Bunge, Phys.Rev.Lett. 44,1450(1980); C.F.Bunge, M.Galán, R.Jáuregui and A.V.Bunge, Nucl.Instr.Meth. 202,299(1982); J.Helman, C.Rau and C.F.Bunge, Phys.Rev. A27,562(1983).
4. A.V.Bunge, C.F.Bunge, G.Cisneros and E.Poulain (in preparation).
5. G.Cisneros, E.Poulain and C.F.Bunge, preprint.
6. V.R.Saunders and M.F.Guest, Comput.Phys.Commun. 26,389(1982); V.R.Saunders and J.H. van Lenthe, Molec.Phys. 48,923(1983).
7. C.F.Bunge and G.Cisneros, Comput. & Chem. (submitted); C.F.Bunge, E.Poulain and G.Cisneros, ibid.(submitted).

QUESTIONS AND ANSWERS

Q: Is "ab initio" the only way to analyze molecular electronic structure for large number of electrons? Do solutions to handling giant sparse matrix exist? Is it efficient on pipeline architecture?

K. Miura

A: There is a loosely defined accuracy (cost!!) threshold above which "ab initio" calculations make sense from the physical or chemical point of view. Below this threshold they may be as unsatisfactory as semi-empirical approaches. For N (N = number of electrons) greater than about 100 at present, feasible "ab initio" calculations can hardly be considered more significant than well-parameterized semi-empirical calculations. Matrix multiplication of huge sparse matrices is feasible on pipeline architecture if only <u>one</u> of the matrices is sparse or if one ignores sparseness in one of the matrices. V. Saunders at Daresbury Lab in England obtains 147 M flop/sec with Cray 1-S for MMD's with dimension approaching 64, and more than 120 M flop/sec for matrices in the 30-64 dimension range, in Assembly language. [In FORTRAN, MMD reaches (linearly with matrix dimension) 37 M flop/sec for dimension 64.] The handling of a large sparse matrix is quite a different thing, as the major problem here may be its calculation and storage, which if possible must be <u>avoided</u>. If one is interested in sparse matrix + vector, one can often write the vector as a matrix and use MMD's appropriately on portions which are chosen according to the particular characteristics of the sparse matrix. If the huge sparse matrix is given, the whole MMD's must be subdivided appropriately to fit into the central memory and care must be taken so that chunks of the result matrix require a single I/O operation.

TRACK RECONSTRUCTION IN PLANAR CHAMBER SYSTEMS

David Bintinger
University of California, San Diego

A long sought after goal is to analyze the problem of track reconstruction in general terms such that the derived lessons can be applied to any experiment requiring track reconstruction. From experience on two different High Energy Physics experiments a method of track reconstruction in planar chambers which has general applicability has been developed. Further, this method leads to a way in which track reconstruction can be performed on dedicated processors with event reconstruction times that would be theoretically independent of multiplicity.

The track reconstruction procedure that will be sketched below was developed for the Tagged Photon Spectrometer at Fermilab. Subsequently this procedure was used on another experiment, PEP-9 at SLAC, with a very different geometry. The nature of the method and the fact that it has been used on two different experiments leads one to believe it has general applicability.

The procedure has two phases. The first is the rapid finding and storing of all possible track candidates. The second is the selection from among the track candidates of the best tracks. These best tracks are then reported out as final tracks, ready for momentum fitting. The first phase seems to be saying: to find all the tracks first find all the possible tracks. To some degree this is exactly what it is saying. Finding possible tracks, or track candidates, is a straight forward procedure. Any planar chamber system is usually over-constrained. This allows one to propose a trajectory based on a few hits and then to follow the trajectory finding which planes have hits associated with it. The trajectory following procedure consist of proposing a coordinate in a given plane and then determining whether the plane contains a real hit close enough to the proposed coordinate to be accepted. This process is followed until one has passed enough planes without finding the required number hits, at which point one abandons the proposed trajectory, or until all planes have been examined and the required number of hits have been found. The trick is to not ask questions while one is carrying out this procedure. If the proposed trajectory has more than a minimum number of hits it is simply stored. No time is wasted asking whether it is a good or bad, real or spurious, track. The successful proposed trajectories become track candidates, not necessarily final tracks. It is easier to follow trajectories than to predict which trajectories will produce final tracks.

Now perhaps it can be seen why this first phase might be fast and, more importantly, suitable for use in dedicated tracking processors. Simple "find a hit that matches this proposed coordinate" searches dominate the trajectory following. That such searches dominate track reconstruction has been recognized before. The Fermilab Advance Computer Project has proposed co-processors that carry out just such searches. When one completes this first phase a list of track candidates has been assembled. It should be noted that hits are not marked out as they are used. Thus there is no bias against finding tracks that share hits. Photon conversions are a prime example of such sharing.

The second phase of this method selects the final tracks from among the track candidates. Often the number of track candidates will be a factor of four or five

times larger than the number of final tracks. The selection phase begins by sorting the candidates into groups. Each group contains several representations of the same physical track. These representations of the same track may differ by including or excluding different hits, having different drift chamber left-right assignments, or having differing projections in some tracking regions. In general one has in each group all the variations of a real track one might find due to differing search patterns. By selecting the best representation from among the group as the final track one is not at the mercy of accepting the first representation that one stumbles across in a given search pattern. For each representation one has all the information that is possible to obtain for it. Thus selection of the best tracks is as precise as hit information allows. If in phase one of this method one tried to conjecture which path was going to lead to the final best representation one would have to make uncertain time consuming estimates during the search based on less than complete information. This results in poorer tracking and longer search times. One can now see the reason for the "no-decision" search of phase one of this method. The grouping in this second phase can be done in any number of ways that uses the parameters of the stored candidates. Candidates with approximately the same parameters are grouped. Also ghost tracks are discovered and discarded in this phase.

As has been mentioned before this reconstruction method has been used in two experiments with very different geometries. In both, this method was faster and more efficient than the existing track reconstruction programs. Another benefit was the ease of constructing the necessary programs. In both cases four physicist-months were required to write and test the necessary programs. This is to be compared with the several physicist-years that were expended on the prior existing programs. This consideration is as important as the considerations of reconstruction speed and efficiency.

The final goal of this paper is to propose a general plan for reconstructing tracks on dedicated processors. The plan utilizes the reconstruction method that has been outlined above in conjunction with one additional crucial observation. This observation is that proposed track trajectories may be investigated independently for the necessary confirming hits. Essentially this means that many track searches within the same event can be occurring simultaneously. The limiting factor is the availability of sub-processors to handle the trajectory extrapolations. These searches would be initiated by a central coordinating processor which would order proposed trajectories and assign them to lower echelon extrapolating processors. Figure 1 is a crude block diagram of how a dedicated track reconstruction processor based on these ideas might be organized. The Central Ordering Processor (COP) proposes a trajectory based on as few plane hits as possible and then assigns the extrapolation of this trajectory to a free Extrapolating Processor (EP). The COP must insure that trajectories are ordered such that undue duplicate trajectories are not assigned. (Some duplication at this level is not a serious shortcoming as the final results will not be affected.) Attending each EP are the "find a hit that matches this coordinate" co-processors that have been mentioned previously. EP's may be subdivided by natural break points in the extrapolation. Thus an EP that after searching n planes finds insufficient hits to warrant continuing the extrapolation will signal to the COP that it is ready to accept another trajectory. If on the other hand it finds a sufficient number of hits, a less likely probability than insufficient hits, it will pass the trajectory with its added information on to the next phase of extrapolation. This means that fewer stage two EP's would be needed then stage

one EP's. At the end of this processing a collection of track candidates will have been assembled.

Theoretically if the COP has been efficient and there were sufficient EP's, the time to reconstruct all candidates should be equal to the time it takes to reconstruct one track candidate. This means event reconstruction time would be independent of track multiplicity. At this point phase one of the reconstruction method has been completed. Phase two now proceeds to select the best tracks as final tracks. Phase two requires all the candidates to be available. Thus it cannot be done in a parallel processor fashion, however this phase is not time consuming. One dedicated processor must handle this task for each event. How the experimenter would program for the processor must also be considered. The scheme presently envisaged is for the user to write a FORTRAN reconstruction program, necessarily utilizing the reconstruction method outlined in this paper, that would run on a large mainframe computer. A form of super-compiler would then take this FORTRAN program and parcel it out amoung the various processors and sub-processors. Thus the entire dedicated processor would look like a large mainframe computer to the experimenter.

Finally this paper has addressed itself only to planar chamber systems. No experience has yet been gathered on cylindrical tracking systems. However it seems the principles of reconstruction given above are general enough such that there is hope in applying them to any geometry.

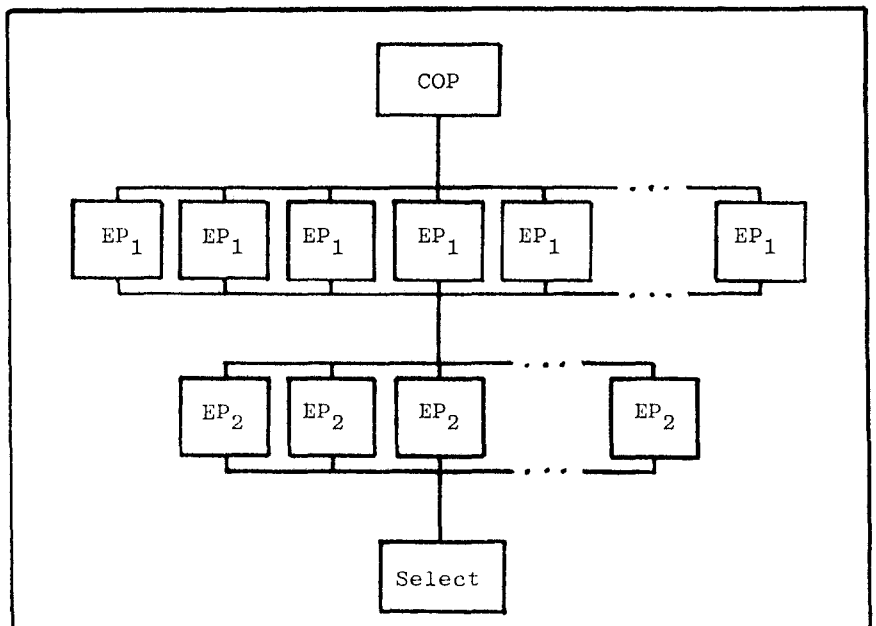


Figure 1

QUESTIONS AND ANSWERS

Q: Do you feel that you can learn anything (concerning pattern recognition, simultaneous testing of multiple hypotheses, parallel computation, hypothesis rejection) from work in the areas of speech recognition - and chess programs?

M.J. Levine

A: From chess programs, yes. I am not familiar at all with speech recognition problems. In both chess and track reconstruction programs it is very true that what you do at the start affects the outcome.

Comment: Since the speaker mentioned that track reconstruction program is similar to chess program, I would like to comment that the strongest chess program is written in FORTRAN and runs on CRAY-XMP (CRAY=BLITZ), i.e., chess program is vectorizable. Does this fact indicate that track reconstruction is vectorizable?

K. Miura

SOFTWARE AND THE DANGERS TO PHYSICS FROM COMPUTING

T. A. Brody
Instituto de Física, UNAM
Apdo. Postal 20-364,
01000 México, D.F.
MEXICO

A B S T R A C T :

Some presently neglected dangers that arise for physics from the unconsidered use of computers are discussed. One basic problem involved in all four is the black-box nature of present-day programs. Several computing techniques, for not all of which implementations have been developed, are proposed that together might yield a more satisfactory programming system than those now available. But it is stressed that such methods can provide at most part of the solution: we must also evolve more appropriate methods of incorporating extensive computer usage into our research procedures and organisation.

I

That the exponential increase in computing power achieved over the last few decades has brought complex problems in the wake of its undoubted benefits is by now a commonplace as regards the administrative applications; but that within the field of physics analogous difficulties are developing is less widely recognised in the physics community. Yet I believe there are serious dangers that need urgent attention, precisely because of our general unawareness: physicists are supposed to know about computing, unlike the business community, say; indeed, we do; and yet we all too often use computers as simple tools without any adequate understanding of the further implications.

The uses of computers in physics are multifarious. They range from the tiny program written, debugged and run in half an hour - and deleted almost at once - to quarter-million-line systems, worked on for years and widely distributed among various user groups. I shall here ignore the first type, as well as the uses of computers in physics education, not because I think them innocent of possible problems (they clearly have many), but because these problems fall into too many different categories, many of which have received quite adequate attention already. It is the problems created by the medium-to-large program systems that I wish to discuss here.

The first and most obvious one is that of ensuring the program's correctness. Here the solution is known, of course, at least in principle. It is simply that of structuring the program into reasonably independent parts. If there are, say, 100 binary decisions in the program, then $2^{100} \approx 10^{30}$ different paths through it must be explored before correctness is reasonably certain; if we break this up into 10 blocks with 10 decisions each, the number of paths is reduced to $10 * 2^{10} \approx 10^4$, a large but not unmanageable number. But this fact is not very widely appreciated, so that too many physicists continue to use programming languages like FORTRAN which not only add further decisions to be checked by making us write three nested DO-loops for a matrix multiplication, for instance, but also foster the production of huge unstructured and sometimes unstructurable program segments. A properly structured program offers the further non-trivial advantage of logical clarity. Yet by the side of numberless honorable exceptions there are still too many programs which according to the author may have been well debugged, but which nobody else can really check. And for very large systems, written sometimes by a sizable group, the problem still is far from satisfactorily solved.

The situation is made worse when, as all too often happens, the comments are missing or inadequate or even (a case we have all come across) misleading.

A second and much more difficult problem is that of the precision and the scope of the program. The algorithms it incorporates are approximate in one way or another; the numerical procedure, which represents continuous variables by a finite discrete set, is necessarily so; and many of the parameter values in the program are likewise only approximations. Hence the results will only be good to within certain error limits. These will be adequate for the original purposes for which the program was written; but further users, whether the author or not, will apply it to other cases, where either the original precision will not suffice, or we have moved out into a region of argument values where the precision is very much worse. A careful writeup will usually have something to say about such matters. But the scope of values for which the precision is not seriously degraded is commonly very difficult to establish, so that we tend to be ultracautious in estimating the scope; this policy tends to provoke the reaction "Oh well, let's try it and see". Unfortunately, the program will all too often produce some sort of result which is then accepted without further ado. Moreover, a careful writeup is just what is not always available, for any of several reasons.

But all these are only the superficial difficulties; the real one is that in order to estimate the precision we need some sort of comparison result, if possible either a better theoretical calculation or a well confirmed experiment. Yet generally we write and use the program precisely because it is the best or even only way to determine the values whose precision is in question. And again, provided the user is fully aware of these matters and therefore exercises due caution, all will be well. But ... how many physicists exercise due caution ?

Such problems, however serious at the moment, should yield to a determined effort towards better education in the use of computers. We must make efforts not only to make the present generation of physicists aware of such problems, but above all to ensure that the future generation is taught about such matters and can gain adequate experience in handling the problems of physics on a machine. This is a far from trivial problem: students these days arrive at their university with a very misleading experience of computing gained through BASIC on the so-called personal computers, and the usual courses offered them do little to correct this.

The next problem to be discussed has graver implications. When results obtained from a large (and hence largely impenetrable) program are submitted for publication, how is the referee to judge whether what has been done satisfies the criteria for a good and original paper ? If you are among the authors, you will not worry overmuch; if you have ever refereed, you will recognise that the problem goes deep: refereeing is part of that process of mutual criticism and discussion which alone keeps physics alive as a scientific discipline. If we let difficulties grow in its path, physics is in danger of degenerating into dogmatism.

The last two problems I want to mention are equally worrisome. One arises in the use of computers for experiments and might be called the serendipity problem: We almost always use the computer not merely to treat and interpret the data but to select the relevant ones. This may mean that we remove a few extreme cases among a few hundred valid data, or it may mean that we select a few hundred relevant events from among 10^{10} or more. In either case the selection criteria are derived from previous theory, and the experimenter is not usually offered a chance to review the rejected data (if only because of their huge volume). Yet how often in the past has it not been an aberrant observation that gave rise to a real jump forward ! No guarantee, of course; and even if a suggestive observation is found, to recognise it as such and profit by it requires a physicist of no mean calibre. But effectively to cut us off from this possibility altogether, as present-day automatic selection procedures do, is to eliminate a significant creative element from physical research and to reinforce the tendency to mere mechanical extensions of present-day knowledge.

A related danger faces us in the large-scale use of simulation programs in theoretical work. Such programs inevitably embody preconceived notions held (sometimes quite unconsciously) by their authors. These notions may be perfectly justified, or they may not. But when the program is used simultaneously by the experimenter as his theoretical structure and by the theoretician to provide his 'experimental' data - then we are in trouble: any bias in the program's conception pulls both experimenter and theoretical physicist in the same direction, so that the agreement between theory and experiment seems to confirm the validity of these preconceived notions. Moreover, any real analysis of the resulting problems is made more difficult by the fact that computer physics, by its intercalation between experimental and theoretical physics, increases their already uncomfortable separation. Once more, the scientific character of our work is in danger, this time because the confrontation of experiment and theory becomes harder and is shifted out of focus.

Most people working in computer physics are to some extent aware of these difficulties, of course. But surprisingly little work seems to be done to solve them, perhaps because of the widening gulf between physicists and professional computer scientists, each specialising away in his own little corner; and there is even less recognition of the fact that all these problems create essentially the same dangers. Indeed, in a sense they also have the same origin: the acritical use of computers. And as this comment strongly suggests, the solution will not be found entirely within the realm of computing techniques.

This is obviously true of the referee's problem, of course. Merely handing him a program listing is no solution, not even if he is given access to the one computer system on which the program is known to run correctly (portability

claims notwithstanding). For he would probably need more time than he can afford before he understands the program sufficiently well; and supposing he then finds something to criticise - what then ? Is two or three years' work to be repeated ? And if so, who will finance it ? One possible solution would be to introduce the referee much earlier, in the actual research rather than at the very end, when all the mistakes have been made. Such a proposal seems not impossible, though it would require a much profounder reorganisation of the refereeing and publishing process than for instance the mere abandonment of anonymity so much discussed at present.

The serendipity problem might be alleviated by allowing several experimental groups, with different ideas, to set up simultaneous and possibly conflicting selection procedures, to be carried out while the experiment is running. Again, though not obviously impossible, such a solution would require very profound changes in the traditional ways of organising and financing research; many people might even resent such suggestions, and who would blame them ?

I am not here advocating that we make such drastic changes in the immediate future. Yet it appears to me that we cannot escape them for very long: we must face the fact that in the long run, the impact of computing in physics will go far beyond what we have already experienced. For instance, we have still to consider what a development presently coming to maturity may imply: the use of computers for handling algebra. Nor have we taken into account the promise - or threat - of future developments in computer architecture, of which parallel processors and distributed processing are but the foretaste.

For the moment, the only feasible thing is that these problems should receive very wide discussion in the physics community. This would not only increase our awareness of the problems; it would also increase the likelihood of better proposals appearing than my tentative and incomplete comments.

There are, however, a number of aspects of these problems than can be tackled by computing techniques. Perhaps the most significant such aspect is the black-box character that most of our programs derive from the use of languages like FORTRAN, developed when only batch processing was available. In what follows, I wish to present some ideas towards developing a programming language which might help to open up the black boxes; I shall call this as yet unborn language MEXLAN, an acronym which may be taken variously to refer to its country of origin or to expand as Meaningfully EXtended LANguage.*

In designing such a language, I have in mind computers whose architecture is essentially of the presently most common type: one CPU, extensive secondary storage, parallel processing only in certain well specified areas such as I/O. There are two reasons for this: firstly, such machines will form the physicist's staple for a good many years to come - if

only because a Cray remains rather expensive; secondly, the special designs such as pipe-line machines are, at least for the moment, special-purpose designs, while the vector processors are extremely efficient only for certain limited problem types, largely because we still lack a basic programming theory for them. Hence the middle range that I am concerned with offers the greatest generality. I do not ignore in this the tremendous surge of the small and very small computers. But it seems a much better policy to develop a language or a system for a type of machine where size and speed restrictions are not too significant, and once experience has been gained with it to choose from among its possibilities those which should be implemented in a more limited environment.

II

MEXLAN would be a language of recursive type, not too unlike ALGOL and its various successors, with several additional features that would go far towards making it a programming system rather than a programming language; I present these features, roughly in order of feasibility, beginning with those whose implementation is either obvious or has already been achieved in other contexts.

Firstly, all variables used in the program should be declared. But beyond the usual declaration, giving the bare minimum information the compiler needs, MEXLAN will require that the dimensionality (in the physical sense, not the mathematical nor the computing one) and the units for each variable be given. Note that in a larger program such information would be included in the comments; MEXLAN merely makes this information available to the compiler. The purpose is twofold: it obliges the user to pay more detailed attention to the physical meaning of what he does (a process which tends to occur only after some error has been detected) and it enables the compiler to detect a sizable class of errors, either revealed by a dimensional check or in the case of inconsistent units avoided by the introduction of the appropriate conversion coefficient. This checking by the compiler can of course be nullified by making all quantities dimensionless; but it can also be made quite effective by choosing suitable dimensional schemes. Even commercial

* Many of those ideas arose during casual conversations with colleagues over the years, long before the MEXLAN concept arose, and I am now unable to attribute them with any certainty to their authors. To avoid injustices I have therefore suppressed all references, and I apologize to all those whose contribution I appear unwittingly to have appropriated.

applications could benefit by this: by assigning different dimensions to capital, salaries, various kinds of costs, and so on, but using the same units for all, one can have a built-in check whether amounts are posted to the right sort of account; alternatively, using different units but making the conversion factor adjustable daily, one obtains automatic currency conversion.

(To accommodate small programs, one might allow a definition facility for undeclared variables: Thus

c '= f(a, b)

might establish the quantity c with the dimensions and units defined by the expression f. But because such a mechanism is extremely dangerous, suitable limits (such as a site-defined upper bound to the number of such variables in any one program) should be set.)

An apparently small but in fact important requirement is that program listings should be properly set out, with spaces, indentations, and blank lines to separate and mark logical levels. Certain manufacturers' early versions of FORTRAN have accustomed us to featureless seas of massed symbols; such listings are unreadable and therefore lead to errors. A 'prettyprint' facility such as many LISP systems offer could improve listings (and source files !); it should be the default option.

A third facility of importance is that building structures of any necessary complexity, as for instance in PASCAL, should be possible in MEXLAN; but together with the data structures, operations on them should be definable, so that e.g. sums and products of matrices could be written in standard notation. Complex numbers should be handled both in real and imaginary parts, as usual, and in amplitude-argument form; this scheme has obvious advantages. If operations on such structures can be freely defined, the interconversion can be made automatic. To make such possibilities useful in practice, the MEXLAN system should be able to store complete structure-operations packages in libraries, to be called in automatically when objects of that type are declared. Each installation could then build its own libraries, so that MEXLAN could be tailored to local requirements.

A fourth feature is that the more complicated operations, e.g. matrix diagonalisations, should not be carried out by algorithms decided upon when building the corresponding library; the system should instead interrogate the user, either at compile time or at run time, about the kind of matrix to be treated, in order to select the appropriate subroutine - and the user should have the option of answering "don't know" to any of the questions, at a risk (which the compiler explains to him) of getting a program that works but is very slow or has large error bars.

Fifthly, the MEXLAN compiler should be conversational; when an error is detected, the user has the option of correcting it before continuing the compilation, and any corrections made will be incorporated in a new version of the source file. This possibility appears to have been implemented in a personal computer; that it would save much time and effort on a mainframe machine is something the computer manufacturers have not yet realised.

Sixthly, the comments should possess a proper organisation and be available on-line whenever the program runs. A suitable structure might be accessible through successive menu levels; if the MEXLAN compiler can interact with its text editor, a simple analysis of program levels would suffice to create compiler prompts for comments at appropriate points (subroutine entries, first appearances of variables, or after too long a stretch with no comment). Unhappily, at the present stage it does not seem possible to let the compiler verify that the comments are clear, complete, and correct; should something like this ever be possible, we could do away with all languages and just use the comments. A further help to the user would be to display the nearest preceding comments when run-time errors occur. The existence of such facilities might stimulate the user into writing more adequate comments.

None of these points are at all difficult to implement; but they do require that we give up the all-too-common concept of a series of independent software facilities on the machine which the user calls explicitly. Instead, they would have to be designed according to a common philosophy, and they would call each other as needed. The next points, however, would imply more extensive changes.

Point seven covers the need for a flexible way to analyse and correct errors during runs. The system should suspend rather than abort a program when an error occurs (even in batch: here a recursive, ALGOL-like structure to the language means that the compiler can determine at suitable breakpoints the minimal information required for restarting, instead of copying the complete image to disk). In the same or a following interactive session the user can then backtrack and discover how the program arrived at the result in error; he can change any relevant values; and he can, finally, restart the program at a suitable point. Of course, it should also be possible to set breakpoints, where the examination of intermediate results is possible even when no error occurs. (Note that such a facility would offer automatic protection against machine errors or power failures, since the operating system could now restart any program that had not concluded when the failure occurred.)

Eighth point: In order to facilitate on-line debugging, a simulated desk calculator should be available that has access to the symbols and values in the program. Any changes in these values made in using this desk calculator should by default only be temporary, but could be made permanent if

explicitly stipulated. Whether, as an extension of this idea, it would also be possible to let the desk calculator execute any function in the subroutine as if it were a defined operation has yet to be explored.

The ninth point is a more complicated extension in the same vein: to make an automatic graph facility available, so that either the final output or intermediate results (when an error has been detected, for instance) could by a means of simple interactive commands be displayed graphically at the terminal. Considerable further research might be needed before a suitable form and default action for these commands could be defined; the huge variety of present-day display formats (and graphics terminals' internal codes) makes this almost impossible at the moment.

Ten: It should be possible to set suitable limits on the acceptable values for any quantities in the program (including intermediate ones), and to do so interactively during execution. If the currently new value of a variable exceeds these limits, an error condition arises; but the user should have the option of allowing the exception.

Eleventh - and last - point: One of the most urgent problems in computer physics is our lack of automatic or even semi-automatic methods for determining reasonable error bounds on the results of lengthy calculations. If such methods existed, they would aid greatly in eliminating useless calculations and establishing the exact value of the better ones; above all, they would much reduce the problems I have mentioned of using programs outside their scope. One such procedure might be an extension of something implemented in the IBM Stretch: with the option of setting the fill bit for left shifts in addition and normalisation either to 0 or to 1, one could on that machine run a program twice; any digits in the final output that coincided were then taken to be 'good'. If insted of using merely the fill bit, one could specify for each number how many bits should be affected, perhaps as an extra byte in the word, then quite general algorithms using this device might be developed. Another possibility might be to compute simultaneously with every numerical operation on two or more operands a worst error estimate (the sum of either the relative or the absolute errors on the operands) and the uncorrelated error estimate; at the end of each block or subroutine, an explicitly programmed calculation of the error in that block is made, using a ponderation that characterizes the algorithm. Whether such a method is feasible and yields reasonable error estimates is at present under study; it may not be sufficiently general to cover all needed cases. In any case, much further work in this area is clearly required.

To conclude with a general point of the design philosophy for MEXLAN, great efforts should be made to make the syntax flexible. It should be designed so that different syntactic forms differ by as many elements as possible, so that the compiler could correct trivial mistakes because they would not in general introduce ambiguities. On the other hand, the

upper-case and lower-case letters should be taken to be distinct: physicists are accustomed to treating x and X as different symbols. Perhaps reserved words should be all upper case: this would make it easy to pick them out in listings.

III

Many other ideas could, of course, be added. I have outlined those I consider to be compatible, in the sense of contributing to a useful and integrated language for the physicist (and probably very many others), and not too ambitious at the present moment. But they are evidently insufficient to define a language; rather, they constitute minimum requirements. Moreover, they are by no means equally easy to implement, and at the least, a good deal of further work on them is needed. Some comments are in order.

In the first place, it has been stated above that what is proposed here under the name of MEXLAN is a complete programming system rather than simply a language definition; in fact, further elements such as a special-purpose editor might conceivably be added. I believe this is vital; to a large extent, present-day languages restrict and hamper the user because they do not function in a reasonably well-defined environment of programming and debugging services: 'user-friendly' is often no more than a sales point, instead of indicating that users' needs are considered. FORTRAN, for instance, is a language that has been implemented on almost every computer; but the different implementations vary greatly in usefulness, to a large extent because of the enormous differences in the remaining elements of the FORTRAN system as between different machines. If we do not do something to provide a reasonably complete and well structured environment to users, the defects in some of these implementations (and I need hardly give examples, for this audience) will hold even the best language back by reducing it to its lowest common expression. But there is a further point: we use computers in order to solve specific problems, and the machine is useful (or "user friendly") to the extent that it can make use of any information relevant to the problem but does not need anything else; the extended declaration of variables and the limit checking described above are a first step in that direction. To put it differently, what the computer must handle are not numbers but quantities: objects that possess not only a value but also a range, a dimension (and units), and certain specifications as to type of variable, limits of physically meaningful values and form of representation.

At the same time, and because the need is urgent, I have tried to limit my ambitions to something that could be achieved in a reasonably short time, even if it is later

superseded by something better. Provided MEXLAN has contributed to stimulating the production of this 'something better', it will have done its job.

For the first few points the software techniques are all available; but much more work will be needed before e.g. the backtracking mechanisms I have asked for can be built in a way that is not enormously wasteful in disk space. Presumably some sort of limitation will have to be set up, making the backtracking less and less detailed as one goes further back from the current position in program execution; a fully recursive structure for the language will be of great help here; but precisely how this is to be engineered has still to be worked out.

Lastly, points ten and eleven go beyond present computer architectures. They could only be implemented if we add some new parallel arithmetic processors: one that keeps track of how new values compare with the limits set on them, the other that evaluates error bounds by the Stretch method (or a better one, if ever it becomes available). Such parallel processing, fortunately, creates no problems from the programming point of view, since the operations are well defined, disjoint from those carried out in the main arithmetic unit, and no complex interlocking is required. The main difficulty - once the algorithms have been properly defined - in getting such processors into our computers would appear to be economic: can we persuade the computer manufacturers to take up such ideas ?

Whether the development of such systems as MEXLAN will make a decisive difference in how physicists use computers only time will show. But it must be remembered that, as I have insisted above, computing science is only half the problem: we ourselves are the other half, and to avoid future disaster we must improve both our computing services and ourselves.

QUESTIONS AND ANSWERS

Q: Please comment on the following:

There are no bad languages, only bad programmers.

F. Beck

A: I agree partially, but some languages help, whereas other hinder, good programming.

Comment: Isn't it true that most of the errors detected by interactive compiler or executor are single, "trivial", easily corrected.

P. Lebrun

A: As yet errors, like inverting, bad flow chart, bad optimization cannot be caught by the most intelligent compiler. It is unfortunately those errors which are time consuming.

Q: There is a lot of software being written for personal computers that incorporates some of these ideas. How do you view the migration of these packages into scientific computing?

M. Delfino

A: There seems to be migration in both directions from mainframes and micros.

Q: Is your group working on these ideas (to create such a system) or do you know someone who is?

D. Kaplan

A: No, but I'll be happy to work with anyone who's interested.

SOFTWARE FOR LARGE SCALE TRACKING STUDIES

J. Niederer
Brookhaven National Laboratory
Upton, New York 11973

Over the past few years, Brookhaven accelerator physicists have been adapting particle tracking programs in planning local storage rings, and lately for SSC reference designs. In addition, the Laboratory is actively considering upgrades to its AGS capabilities aimed at higher proton intensity, polarized proton beams, and heavy ion acceleration. Further activity concerns heavy ion transfer, a proposed booster, and most recently design studies for a heavy ion collider to join to this complex. Circumstances have thus encouraged a search for common features among design and modeling programs and their data, and the corresponding controls efforts among present and tentative machines. Using a version of PATRICIA[1] with nonlinear forces as a vehicle, we have experimented with formal ways to describe accelerator lattice problems to computers as well as to speed up the calculations for large storage ring models. Code treated by straightforward reorganization has served for SSC explorations. The representation work has led to a relational data base centered program, LILA[2], which has desirable properties for dealing with the many thousands of rapidly changing variables in tracking and other model programs.

The mathematical basis of tracking is being refined to find a manageable formulation that preserves the off axis position and momentum phase space of orbiting particles over many turns in the presence of numerous nonlinear forces. The PATRICIA and RACETRACK[3] programs with relatively simple transfer matrices do not have this symplectic behavior, and hence may introduce artifacts in simulated particle tracking. The MARYLIE[4] approach has shown how to express the tracking problem properly through low orders, but gets very complicated when higher order multipoles are included. Approaches which synthesize an appropriate Hamiltonian are being pursued[5,6], and will be included in a future version of the MAD[7] design programs at CERN.

We are collaborating with CERN colleagues in a broad effort to make the MAD programs more flexible and general for the accelerator community. The programs will respond to an agreeable input and control language, with familiar and dynamic vocabulary, applicable to all accelerator programs in principle. The language is to be that of MAD[8], with such modification as suggested by the TRANSPORT collaboration[9]. The generalized language and input handlers developed for LILA apply to MAD. The MAD physics internals will be modified for a message-object[10] style of coding, based upon LILA prototypes which in effect are quite flexible data base handlers and operators. The accelerator physics parts are to work from a centrally managed relational data base that permits an almost unlimited variety of relations among the components of the accelerator model. The new version will make use of interactive capabilities and displays, and be developed on a modern workstation to make use of program development techniques. Finally, the language and the data base services will be acceptable to both design and control programs.

The language and its interpreter make use of the notion of objects, relations, and operators. Details are available in BNL documentation[11]. An object is anything that can take a unique name; in the computer it is a stored collection of information about some real life "thing". Relations are simple lists of objects. Operators read relations, which in turn point to objects, and the objects tell the operators how they are to be processed. Attributes are properties of objects; parameters help to describe operators. Keyword (class) objects show how attributes and parameters are to be contained in the data structures of objects or operator descriptions.

The data base itself has to create, modify, or delete objects and relations. Other duties are to find objects by name, or to link the names of relations to their objects and data. Memory management and directory services are included. The user drops a number of statements describing the accelerator into a pool of memory, and the programs treat all of the storage and association details. Normally initialization routines will convert the input variables into forms, such as matrices, better suited to rapid computation.

The MAD programs being aimed at immediate LEP applications are of obvious interest to the larger SSC. As the newer algorithms are verified, they will be optimized and programs will move from the slower workstations to more suitable computing machines. Although largely a matter of programming technique, the formulations noted here replace almost half of the MAD code with more general language and data handlers, aiding expansion and further program development. The data structuring scheme resembles that of the Xerox Smalltalk Project[12], but there is no correspondence between these languages. The central idea of using a common data base mechanism over many classes of application program is often noted as "Data Driven Prototyping"[13] in the commercial world, with exuberant claims about speeding up the construction of application programs.

This work has benefited through discussion and encouragement from D. Lowenstein and R. Peierls at BNL, and F. C. Iselin and E. Keil at CERN. Support has been provided by DOE Contract DE-AC02-76CH00016.

References

1. S. Kheifets, H. Wiedemann, "Modification of PATRICIA to Include the Influence of Higher Field Multipoles in Different Elements", SLAC PTM-151, (May 1978).

2. J. Niederer and B. Morris, Proc. Workshop on Accelerator Orbit and Particle Tracking Programs, Brookhaven National Laboratory, New York, BNL 31761, p.26, (May 1982).

3. A. Wrulich, ibid, p.277.

4. A. J. Dragt, "Lectures on Nonlinear Orbit Dynamics", Physics of High Energy Accelerators, Fermilab Summer School, 1981, AIP Conference Proceedings #87, (1981).

5. R. D. Ruth, IEEE Trans. Nucl. Sci., NS-30, No. 4, pp.2669-2671, (1983).

6. F. C. Iselin, Private Communication.

7. F. C. Iselin, "The MAD Program", 12th International Conference on High Energy Accelerators, Fermilab, (August 1983). (CERN-LEP/TH/83-30).

8. F. C. Iselin, "The MAD Program: Reference Manual", CERN, Geneva, Switzerland, (October 20, 1983).

9. F. C. Iselin, Private Communication.

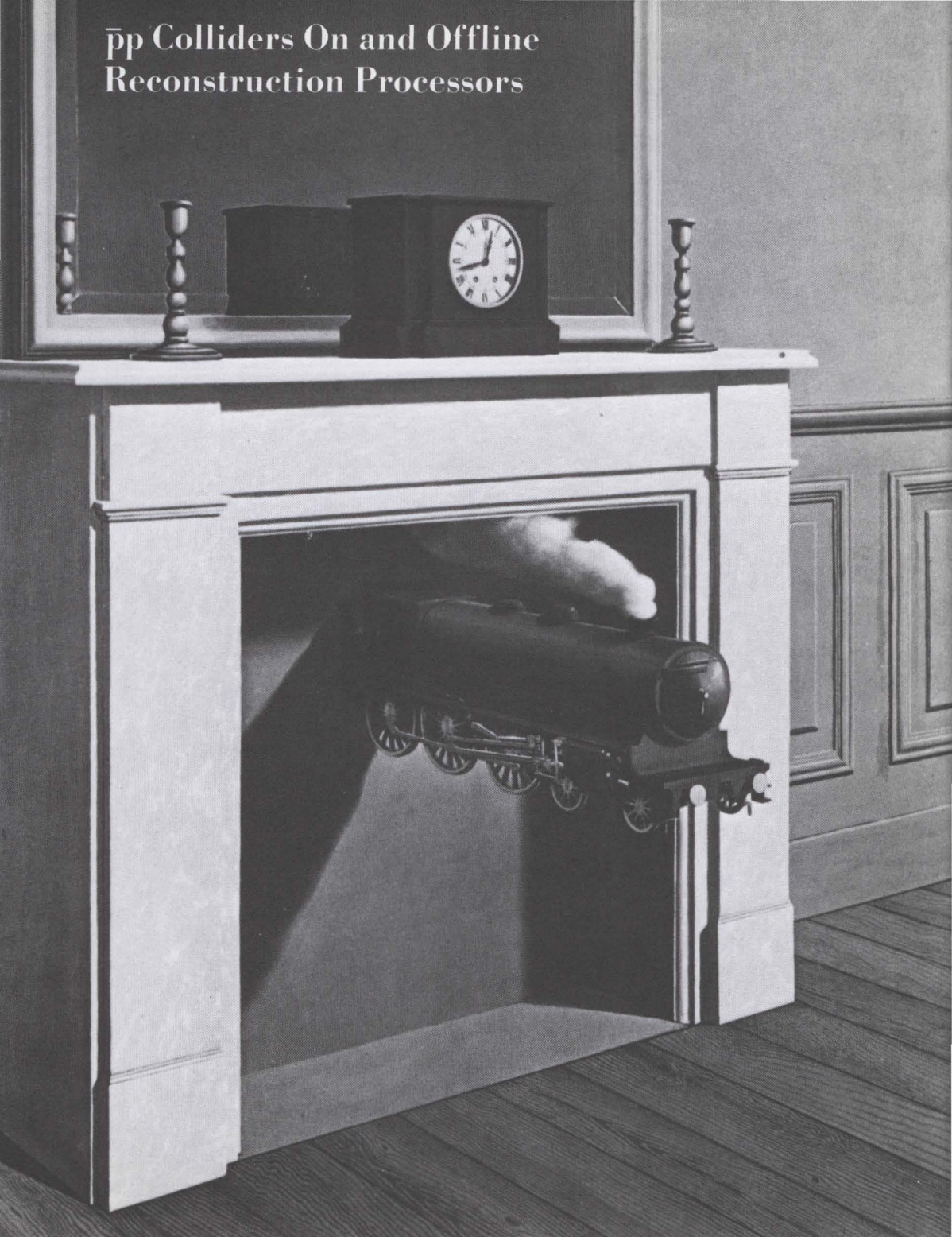
10. B. D. Cox, Message/Object Programming: An Evolutionary Change in Programming Technology, IEEE Software, Vol. 1, No. 1, pp.50-61, (1984).

11. J. Niederer, Relational Data Bases in an Accelerator Environment, BNL, (1984).

12. Goldberg and Robinson, Smalltalk 80, the Language and Its Implementation, Addison Wesley, New York, (1983).

13. D. S. Appleton, Data Driven Prototyping, Datamation, pp.259-268, (Nov. 1983).

$\bar{p}p$ Colliders On and Offline Reconstruction Processors



[La Duree poignardee (Time transfixed), 1939, René Magritte, 1898-1967]

53 MHz DIGITAL PROCESSOR FOR REAL TIME CALCULATION OF BEAM ORBIT CORRECTIONS IN THE FERMILAB TEVATRON

Marvin Johnson* and Louis Rolih
Fermi National Accelerator Laboratory†
P. O. Box 500
Batavia, Illinois 60510

Abstract

We have built a pipelined processor which takes the measured position and intensity of each radio frequency bucket, computes deviation from the desired orbit and then computes the voltage to be applied to a fast kicker to correct the orbit. The device also computes the average positions of the beam and the fractional part of the betatron tune. All computations are done in real time at a 53 MHz rate.

Betatron oscillations in the Tevatron ring are a primary limitation on the intensity of the Fermilab accelerator. These oscillations are caused by magnetic fields from eddy currents induced in the stainless steel bore tube by the passage of particle bunches themselves. This is commonly referred to as the resistive wall affect.

To combat this affect, we have built a device which measures the beam position of each particle bunch, calculates the amount of displacement, and then provides a kick in the opposite direction. All the calculations are done with a pipelined processor using Motorola 10K and 10KH integrated circuits. Since the separation between bunches is only 18.9 ns, the digital processor part of this system was designed to run with a clock period of 17 ns. The boards are all hand wired using insulation displacement prototype boards from Robinson-Nugent Co. The system had to be compatible with the Fermilab control system so it is packaged in CAMAC. The design makes heavy use of pipelining to achieve these clock speeds.

*Presented by Marvin Johnson.

†Operated by Universities Research Association, Inc., under contract with the United States Department of Energy.

A block diagram of this device is shown in Fig. 1. A four foot long, two plate detector measures either the horizontal or vertical beam displacement. The signals from the two plates (A and B) are summed to give the total intensity (A+B) and subtracted to give the displacement (A-B). This is done by passive power splitters, inverters and combiners. The signal is then integrated over the length of a bunch (2 ns.) and digitized by a flash analog to digital converter (ADC). The ADC consists of four 6 bit ADC's (Analog devices 5010s) stacked to give ± seven bits of resolution.

This system needs a position resolution of approximately ±64 and a total intensity variation of 20. This gives a total dynamic range of ±1200 which is much greater than our 8 bit system can provide. We solved this problem by using 2 sets of ADCs. The first set has an additional analog gain of 10. When the first set overflows, the input to the rest of the system is automatically switched to the second set. This gives an overall dynamic range of ±1280.

The effect of intensity variations is removed by dividing the (A-B) signal by (A+B). This is done digitally by using PROMs to find the log(A-B) and -log(A+B). These signals are added and another set of PROMs is used to exponentiate the sum. Base 2 logarithms are used. We obtained a maximum error of 3 per cent with 8 bit log tables and 8 bit exponential tables. Going to 9 bit log tables increases the accuracy to about 1.5 per cent.

Although the use of logarithms allows fast computation, they have significant draw backs. The log of zero is not defined. The log of a negative number is complex. Even though (A-B) is in principle always less than or equal to (A+B), our system has some noise in it so we must take care of the case where (A-B) is greater than (A+B) also. Each of these problems can be solved rather easily but when taken together they add circuit complexity. Because of this and the availability of fast, 16K static RAMs, we are designing a board that simply uses a 14 bit wide table look up. Negative numbers are obtained by using PROMs to convert the 8 bit quotient to twos complement.

Since the particles travel at the speed of light, the results of the calculation must be delayed until the particles come back around again. This delay is about 21 ms. We achieved this delay by using RAMs to implement a long shift register. A straight forward shift register cannot be implemented at these frequencies since this would require both a read and a write to the same memory chip in one 19 ns. cycle. To solve this problem, we divided the memories into two parts - an even one and an odd one. Each memory is 1K long by 12 bits wide. An 11 bit counter is used to determine the memory address to read. The write address then determines the delay. It is computed by adding the delay (set in switches) to the counter address. That is, if the delay

is 600 clock steps and the counter is set to 1, the current result is written to address 601 and the delayed result is read from address 1. Six hundred steps later, the delayed result would be read from address 601 and the current data would be written to address 1201. In order to avoid the memory conflict mentioned earlier, the delay address is forced to be odd always. Thus, if one is reading from an odd address, one is always writing to an even address and vice versa. This means that the delay increment is 2 clock steps, but this is not important to us.

The delayed quotient is then fed into a module that computes the damping voltage that is to be applied to the given bunch. This module subtracts the average position of the beam and then uses a look-up memory to determine the amplitude of the kick. During the acceleration cycle the beam energy and the magnetic fields do not track precisely. Thus the average beam position varies throughout the cycle. To correct for this, a module (described below) calculates the average position of the bunches in the preceeding turn. This is subtracted from the present bunch position before the data is sent to the look up memory.

The look up memory is 12 bits wide and the position is only 8 bits wide. This gives 16 different operations that can be applied to a given bunch. Which one of the 16 to apply to a given bunch is determined by a function select module. There is one 4 bit function code for each of the 1113 bunches in the machine. Eight of the sixteen functions can be gated by an external clock. An example of a special function is the set-up for proton anti-proton collisions. One would select normal damping on 1 bunch and anti damping on all the others. This would remove all but one bunch from the machine.

The output of the lookup memory is then fed to a high speed Digital to Analog Converter(DAC) and then to a power amplifier

The average position of the beam is determined by summing selected bunches and dividing by the number of bunches. Since the number of bunches is not crucial, they are restricted to a power of 2 so that the division is just a shift. Bunches are again selected by a memory and are accumulated by a recursive adder, i.e., the output of the adder is connected (via a latch register) to its input. A new average position is output at the end of every turn. Because of the speed limitations of fast adders, this board skips every other bunch.

We hope to use the damper for continuous tune measurements of the machine. One of the 16 functions will be programmed to anti-damp a bunch if the position amplitude is small and to damp if the amplitude is large. This will then excite betatron oscillations of roughly constant amplitude. This function will be selected for only one bunch in the machine so that we don't cause the entire beam to blow up. In principle, the position of this

one bunch needs to be Fourier analyzed to obtain the betatron frequency. However, the betetron frequency is a single frequency so all that is required is to count zero crossings. We are designing a module to do this. The module's input will be the bunch position corrected for the average offset.

Finally, a large diagnostic memory has been built which will record the position of the last 64K bunches. This can be all the bunches for approximately 58 turns or 1 bunch for 64K turns or any combination in between. The memory is constructed from 16K static rams interleaved 4 ways. This will be quite important for machine studies and for finding problems in the machine.

Since this processor cannot test itself, we developed a computer assisted test setup using a 60 MHz. random number generator. This system has 16 input bits of 65000 different states. We also wanted to test the affect of the previous state on the present state. This gives 65000 squared different states which is about 4 billion.

The random number generator is a Tausworthe[1] generator based on Mersenne primes. We used the trinomial generator x**31+x**13+1 which can be shown to have a period of 2**31[2]. This generator passes all first order Chi-square tests for randomness but fails second order tests (pairs of numbers are not random). This problem could be solved with more hardware but first order randomness is adequate for testing.

Next, one needs to know how long to run so that every interval of a given size has been tested at least once. With random numbers this becomes a statistical question. For a small fraction of the total interval the distibution of uniform random numbers follows a Poisson distribution. Thus the probability of having no number in the interval is just exp(-m) where m is the average number that should occur in the interval. Therefore, to find the necessary running time, one selects the minimum size of the interval to probe and the probability of testing each interval of this size at least once. The natural logarithim of this probability gives the average number of tests for this interval.

[1]Tausworthe,R.C. "Random Numbers Generated by Linear Recurrence Modulo Two,"Math. Comp. 19(1965),201-209.

[2]Bright,H.S. and Enison,R.L.,"Quasi-Random Number Sequences from a Long-Period TLP Generator With Remarks on Application to Cryptography," Computing Surveys,11, 4(1979),357-370.

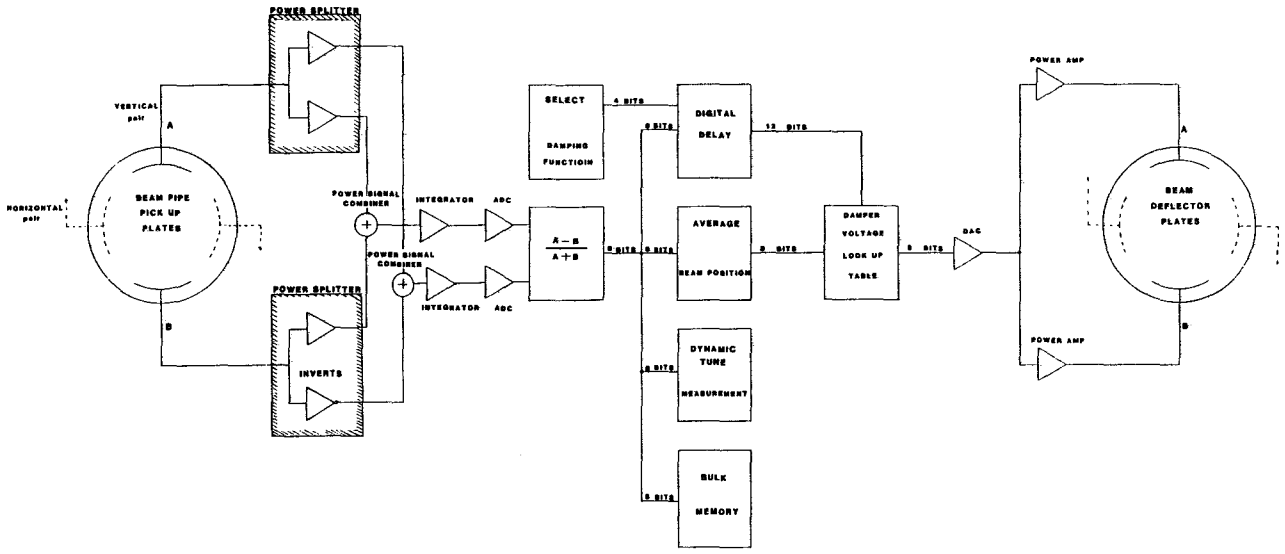


Fig. 1. Block diagram of the beam damper system for the Tevatron.

THE DESY BEAM ORBIT PROCESSOR

W.Neff, H.Quehl, H.J.Stuckenberg,
Deutsches Elektronen-Synchrotron DESY, Hamburg,

P.Leu*, E.Lohrmann, P.Wilhelm,
II.Institut für Experimentalphysik, Univ. Hamburg, Germany.

1. Overview of the Problem

Computing the motion of particles in a storage ring is an important task in accelerator physics. In realistic studies one has very often to go beyond the approximation of linear optics, and consider the effect of higher order multipoles and other deviations from a simple machine consisting only of dipoles and quadrupoles. A standard approach consists of tracking one particle at a time through the magnetic elements of the storage ring by explicit computation of particle motion in each element of the storage ring.

In order to determine important properties of the ring like its aperture and the stability of the beam, one has to consider a rather large number of particles, whose initial conditions are chosen to cover the expected aperture of the storage ring. Moreover, in order to study instabilities one has to track particles through many revolutions around the ring.

This task can be very costly in computing time. As a specific example we take the program RACETRACK[1], which is widely used at DESY for studies of the HERA proton ring, and which we implemented on the DESY Beam Orbit Processor (DBOP). In simulating the HERA proton ring, the tracking part of the program has to consider 850 linear beam elements (quadrupoles and dipoles). The nonlinearities are taken into account by describing them in the form of 800 nonlinear elements placed around the ring. For one revolution one has to carry out about 0.25×10^6 floatingpoint operations (addition, subtraction, multiplication). A study of the necessary numerical accuracy shows, that the floating point operations have to be carried out in double precision (64 bit word), if one wants to go to 10^5 revolutions. Under these conditions one revolution needs about 50 ms of CPU time on an IBM 3081D.

The structure of the tracking program is simple. The particles are considered independently one at a time. Most of the CPU time is used for evaluating polynomials in double precision. The program contains about 180 FORTRAN statements and uses about 100 k bytes of core storage. The relatively small size of the tracking part of the program suggest the parallel use of many microprocessors, each of which contains the same tracking program and works on one particle with specific starting conditions.

2. System Design

Some general requirements had to be taken into account when designing the processor system:

- Execution speed: For the RACETRACK tracking program the complete system should have adequate computing power as compared to a major computer like the IBM 3081D.
- Programming: FORTRAN programs must run on the system, program modifications must be easy.

*Presenter

- Time scale: To make maximum use of the machine for HERA construction, prototypes should be finished within 1/2 year and a first system of 8 processors run within about 1 year.

In Fig. 1 the conceptual design showing a microprocessor (μP) and the host is shown. The system is controlled by the host computer which is master on the global bus. All microprocessors are interfaced to the global bus as slave devices. The local bus is the μP bus itself. The μP is connected via the local bus to various interfaces, program memory and the floating point unit (fpu). The fpu consists of a fast dual ported data memory of 128 k byte and the floating point processor. As a further extension a hardware subroutine for the evaluation of polynominals is foreseen. The floating point modules are interconnected by the floating point bus which has 2 x 64 bit data lines.

For the μP we have chosen the Motorola MC 68000 with 12 MHz clock. The program memory is 128 k byte of 8 k byte static RAMs. There are two serial line interfaces and one 8/16 bit parallel I/O port connected to the μP. The μP, program memory and interfaces for the prototype processor are commercial products[2].

Each microprocessor has the same version of the tracking program. The host computer supplies the same load module to each μP and then transmits specific starting values of the particle orbit to be computed. The computing time is mainly determined by floating point operations. Since μP with fast floating point hardware for double precision are presently not available, we have built a special floating point unit.

One of the basic ideas of the system was to avoid any cross software complications by using a host computer with a MC 68000 CPU. The host then also serves as a software development machine and edited, compiled and linked programs can run directly on the microprocessors. A workstation[3] is used running under the UNIX operating system. It supports C and FORTRAN 77.

To support the requests for operating system services embedded in the linked program modules we have developed a supervisor program which is permanently loaded on each of the microprocessors. This supervisor intercepts and services all necessary system calls.

The host compiler generates floating point instructions which meet the requirements of the Motorola software coprocessor[4] MC 68341. This emulation software package is also installed on each μP so that programs using floating point arithmetic can run on the system without the hardware fpu.

Having on the assembler level explicit floating point instructions instead of subroutine calls makes interfacing to the hardware fpu straightforward: All floating point instructions which are to be executed by our special hardware are replaced on assembler level.

The speed of the MC 68000 gives an upper limit on the floating point computing speed. Neglecting the time required by the fpu itself the maximum computing speed is $0.2 - 0.7 \times 10^6$ floating point calculations per second. This depends mainly on the length of the arithmetic expression to be evaluated, where

long calculations are favoured.

Assuming such a machine, the shortest possible execution time for RACETRACK can be estimated to be about 0.5 sec per revolution per particle. This is about 10 times more than on a IBM 3081D or about as long as on a VAX 780 with floating point accelerator.

To achieve this fast execution time the fpu must be able to transfer 64 bit operands from/to memory within about 400 nsec and to perform 64 bit arithmetic calculations within about 800 nsec.

3. Floating point hardware

Since there are no 64 bit floating point units for microprocessors available which fullfill our needs for execution speed we have built a fast 64 bit floating point processor for addition, subtraction, multiplication and operand compare instructions. Division, which is a very rare operation, is carried out by software. The fpu is shown in Fig. 2. The µP communicates with the floating point controller and data memory via the local bus. An essential property of the fpu are the internal 64 bit data paths for operand transfer. Otherwise passing the 64 bit operands via the 16 bit µP data lines would require 4 read/write processor cycles and thus be much too slow. The floating point controller is a microprogrammed device with 55 nsec cycle time to match the data memory access time. The controller has 16 floating point registers, of which 8 are used by the software. The additional 8 registers and sufficient space for microprogramming is foreseen for an implementation of hardware subroutines. The arithmetic units get their input from two 64 bit busses and pass the result back on one of them.

4. Present status and Outlook

At present we have one microprocessor running. The floating point unit uses 250 nsec for an add instruction and 1200 nsec for a multiply instruction in double precision. The computer needs 0.8 s to track a particle once around the HERA storage ring and thus performs at about 6% of the speed of an IBM 3081D. We now plan to build 8 identical microprocessors to bring the system to full capacity.

References:

1) A.Wrulich, DESY Report 84-026
2) KWS Computersysteme GmbH, Ettlingen, West Germany
3) PCS - Periphere Computersysteme GmbH, Munich, West Germany
4) M 68 KFPS, Motorola.

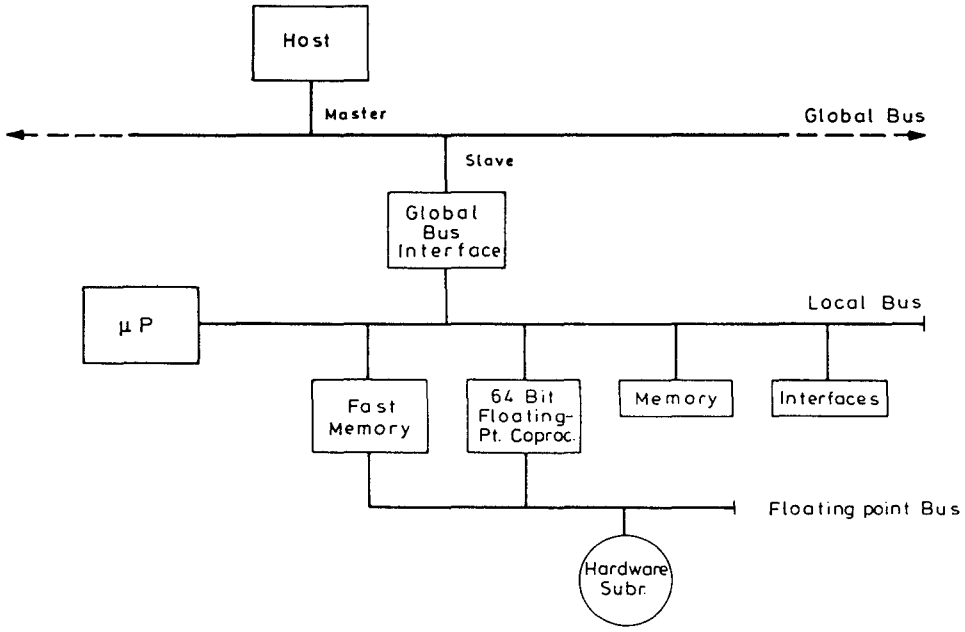


Fig. 1. Design of host and one microprocessor

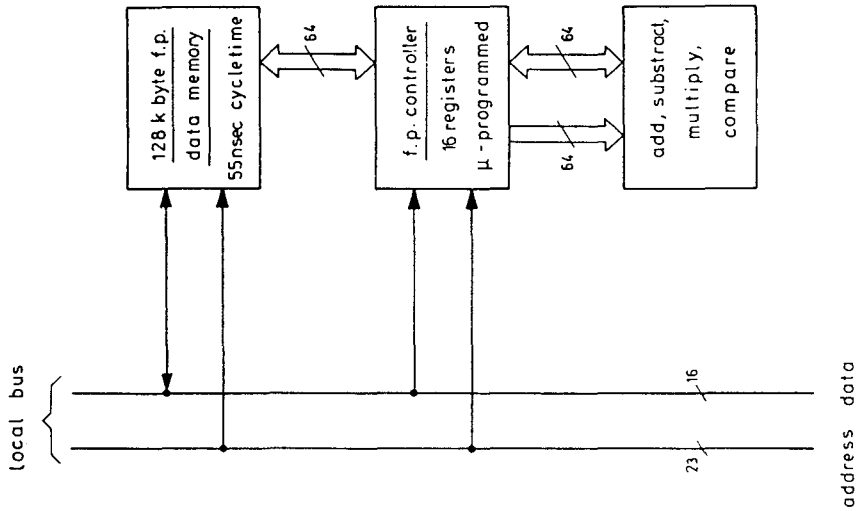


Fig. 2. Floating point unit

QUESTIONS AND ANSWERS

Q: Do you have plans to expand the system beyond 16 processors, and what about interactions among the particles?

D. Kaplan

A: Yes, if we get more money. The global bus is low bandwidth, but interfaces among processors are easy to build to allow dealing with interactions.

Q: How many boards and how expensive is your Floating Point?

M. Fischler

A: 3 boards (+1 for memory). (No cost number recorded.)

ON-LINE USE OF MICE FOR MONITORING $\bar{p}p \rightarrow J/\Psi$ AND FILTERING $\bar{p}p \rightarrow \eta_c$ EVENTS IN THE R704 EXPERIMENT

J.P. Guillaud, Laboratoire d'Annecy-le-Vieux de Physique des Particules (LAPP), Annecy-le-Vieux, France.

ABSTRACT:

MICE, a fast user-microprogrammable processor is part of the on-line system of the R704 experiment. Two different uses have been implemented: monitoring the detection of J/Ψ decays in $\bar{p}p$ annihilations and $\bar{p}p \rightarrow \eta_c$ event filtering. In both aspects, MICE has proved to be efficient, flexible and reliable.

1) MICE

MICE is a fast user-microprogrammable processor which emulates the PDP11 fixed point instruction set. It has been built at CERN with MOTOROLA ECL 10800 bit-slice family and ECL 10146 memory chips (1). Its block diagram is shown in figure 1, a detailed drawing of the CPU architecture can be found in ref. 1. Its micro cycle time is 105ns. In our configuration, MICE has 2 separate memories (fig. 1):

- the Target Memory (TM), 16 bit-wide with a total size of 20kw split in 4kw of fast ECL memory (access-time ~40ns) and 16kw of MOS-memory (access-time ~100ns). The maximum possible TM size is 28kw. TM contains the PDP11 code, program and data.
- the Writeable Control Store (WCS), 1 kw of 120 bits, where the emulator (400w), the diagnostic routine (50w) and the microcode reside. The maximum WCS size is 4kw.

A fast multiplier (~210ns multiply time) is connected to the CPU via extensions of the 2 internal buses (TTL extended buses XIB,XOB). MICE communicates with the external world through 3 interfaces:

- a simplified Unibus adapter for connecting standard PDP11 peripherals
- a CAMAC interface connected to the CAMAC Dataway. It allows the control of MICE from the main computer: loading and dumping of the 2 memories, Program-Counter loading and various control actions as "start", "halt", "dump register content" ... It allows to send and receive 4 interrupts to and from the experiment.
- a cycle stealing Direct-Memory-Access (DMA) interface with 2 different inputs:
  1) a Receiver-Memory-Hybrid (RMH) interface that reads the data from multi-wire proportional chambers (MWPC) and counter hodoscopes at ~410ns per word (due to the limited number of words ($\lesssim$ 50w), no effort was made to decrease this time to the possible ~150ns figure),
  2) a Romulus-Remus interface which allows, through a specially built interface (2), (IDC), to spy the content of the System Crate Dataway during the transfer of the data from the experiment to the main computer. The cycle stealing mechanism allows to interleave this read-out with the processing of the data previously read from the RMH.

2) THE R704 EXPERIMENT

The principal aim of the R704 experiment (4) is the study of the Charmonium states directly produced in $\bar{p}p$ annihilations obtained in the interaction of an intense $\bar{p}$ beam (~5mA), circulating in the ring 2 of the CERN ISR, with a molecular Hydrogen "jet target". This provides a high luminosity ($\gtrsim 10^{30} cm^{-2} \cdot sec^{-1}$), a very good definition of the center of mass energy with the momentum cooling of the ISR ($\Delta m \lesssim 0.3 MeV$) and a favourable geometry of the detection of

the final states. The experimental apparatus has been designed to select the $\bar{p}p$ collisions with final states e^+e^-, $e^+e^-\gamma$, $\gamma\gamma$, $\gamma\gamma\gamma$. These events correspond to the production of the states J/Ψ, η_c, η'_c, χ_0, χ_1, χ_2 ... (for ex, $\bar{p}p \rightarrow \chi_2 \rightarrow e^+e^-\gamma$). It is based (fig. 2) on 2 symetrical detection arms arranged in a trapezoidal shape following the θ/ϕ acceptance in the lab system. Each arm consists of charged detectors (for e^+e^- pairs) and calorimeters (for e and γ shower conversion). The charged part consists of hodoscope counters, MWPC and a freon Cerenkov counter. The calorimeter is divided into 4 parts: a pre-calorimeter (PC) a set of analog chambers (AC), a shower hodoscope (SH) and a lead-glass wall (LG). Extra counters, charged and neutral guard detectors, cover a large part of the remaining solid angle (Veto's).

The general flow of data from CAMAC modules to magnetic tape is supervised by a NORD-10 computer. It provides the data acquisition, through Direct-Tasks, the steering of the experiment and the sample analysis, through Real-Time-Tasks. Before being read out by the NORD-10, the calorimeter information (ADC's) is pre-processed: 2 LRS 2280 μ-processors are used for pedestal subtraction and compaction of the analog chamber ADC's, a CAB μ-processor rejects events with too low energy deposit in the calorimeter ADC's (PC+LG). The total read-out time is ~8msec.

3) MICE AS A J/Ψ DETECTION MONITOR

The RMH information (3), MWPC and Charged hodoscope registers, is read out by MICE via its RMH interface (fig. 3). Consistency checks (in assembly code) are performed within 50μsec, the spatial reconstruction (track and vertex finding, coplanarity, written in Fortran) is completed within 2.2msec. MICE has even enough time left to spy via its Romulus-Remus interface, the information read by the NORD-10: when the data from the calorimeters is available

on the System Crate Dataway, MICE reads it in parallel with the NORD-10, using the IDC mechanism (Indirect Data Channel). The energy deposit in the calorimeters, inside roads centered on the electron tracks, is calculated and a fast kinematical analysis (Fortran) is performed giving the effective mass of the e^+e^- pair (3.3msec). An event is thus fully reconstructed within 5.5msec. Due to the very small width of the J/Ψ (60keV) and to the very good momentum resolution of the $\bar{p}$ beam, the on-line J/Ψ reconstruction allows to optimize the mass scanning operations leading to the absolute momentum calibration of the ISR.

4) MICE AS A $\bar{p}p \rightarrow \eta_c$ EVENT FILTER

The trigger logic of the R704 experiment is mainly based on a two level system:

- a system of ECL discriminators and coincidence matrices followed by a programmable logic unit (PLU LRS4508) selects different types of triggers based on topological θ/ϕ considerations (for charged hodoscopes: θarm1 ⊗ θarm2, ϕarm1 ⊗ ϕarm2, for shower hodoscopes: SHarm1 ⊗ SHarm2).
- a second level of decision is performed by a Majority Decision Unit (MDU) whose selection criteria are based on the multiplicities in the charged hodoscopes, shower and veto counters.

This system has proved to be very selective. But in the case of the decay $\eta_c \rightarrow \phi\phi$, each ϕ decaying in 2 Kaons, it is difficult to distinguish a single track (coming from elastic $\bar{p}p$) from the 2 nearby tracks of the Kaons. The resulting trigger rate is too high (~10Hz/1mA). MICE has thus been inserted in the trigger as event filter.

The RMH is read out (fig. 4) and consistency checks on MWPC are performed as in 2) within 50μsec. A very limited spatial reconstruction (Fortran) requir-

ing at least 2 tracks per arm is completed within 800μsec. In the case of a "bad" event, bad content in MWPC or wrong topological configuration, a RESET is sent to the fast logic, giving a dead-time between 50 and 850μsec (depending on how close to the 4-prong configuration the event is). This decreases the trigger rate by a factor 10.

5. CONCLUSIONS

In the R704 experiment, two important roles are assigned to MICE: monitoring and event filtering.

Using the device to perform track finding and event reconstruction (in $\bar{p}p \rightarrow J/\Psi \rightarrow e^+e^-$, $\bar{p}p \rightarrow \chi_2 \rightarrow J/\Psi \rightarrow e^+e^-\gamma$ ...) has been very easy due to the possibility of programming it in high-level languages such as assembly code or Fortran. As the reconstruction time is not too long, there was no need of using the microprogramming facility. The on-line reconstruction of the J/Ψ decay optimizes the mass scanning operations.

On the other hand, the role of event filtering devoted to MICE in $\bar{p}p \rightarrow \eta_c \rightarrow \phi\phi \rightarrow K^+K^-K^+K^-$ is even more important as the event rate would have been an order of magnitude too high with only ECL fast trigger logic.

Finally, the software environment of MICE was found to be so "user friendly" that program development could be done with minimal effort on the NORD-10 host computer. MICE has proved to be a valuable, flexible and efficient device in both monitoring and event filtering tasks.

6) ACKNOWLEDGEMENTS

The author wants to thank the members of the R704 collaboration, especially C. Baglin, for their encouragement and helpful discussions as MICE was brought on-line. He is much obliged to A. Bazan, M. Feuillet and M. Moynot from Annecy

(LAPP) for their technical assistance. Special thanks are due to J. Anthonioz-Blanc, J. Joosten, M. Letheren and A. Van Praag from CERN (Data Handling Division) whose support and help were invaluable.

REFERENCES

1- C. Halatsis, J. Joosten, M.F. Letheren and A. Van Dam.
Architectural considerations for a microprogrammable emulating engine using bit-slices. Proceedings of the 7th Annual Symposium on Computer Architecture, May 6-8 1980, La Baule, France and CERN Internal Report CERN-DD-79-7.

J. Anthonioz-Blanc, J. Joosten, M. Letheren and A. Van Praag.
User aspects of MICE. Proceedings of the topical Conference on the Application of Microprocessors to High Energy Physics Experiments, May 4-6 1981, CERN, Geneva, Switzerland. CERN 81-07.

2- A. Bazan, Annecy (LAPP).

3- J.B. Lindsay, C. Millerin, J.C. Tarle, H. Verwey and H. Wendler.
A fast and flexible data acquisition system for multiwire proportional chambers and other detectors, Nucl. Instr. Methods, 156 (1978) 329.

4- Annecy (LAPP), CERN, Genova, Lyon, Oslo, Roma, Torino R704 Collaboration:
Charmonium Spectroscopy at ISR using an antiproton beam and a H2 gas "jet target". CERN EP/0011P 8/2/84, CERN Internal Report to be published.

FIGURE CAPTIONS

Figure 1.- MICE block diagram.

Figure 2.- Set-up of the R704 experiment.

Figure 3.- MICE as a monitor in the R704 on-line system.

Figure 4.- MICE as event filter in the R704 on-line system.

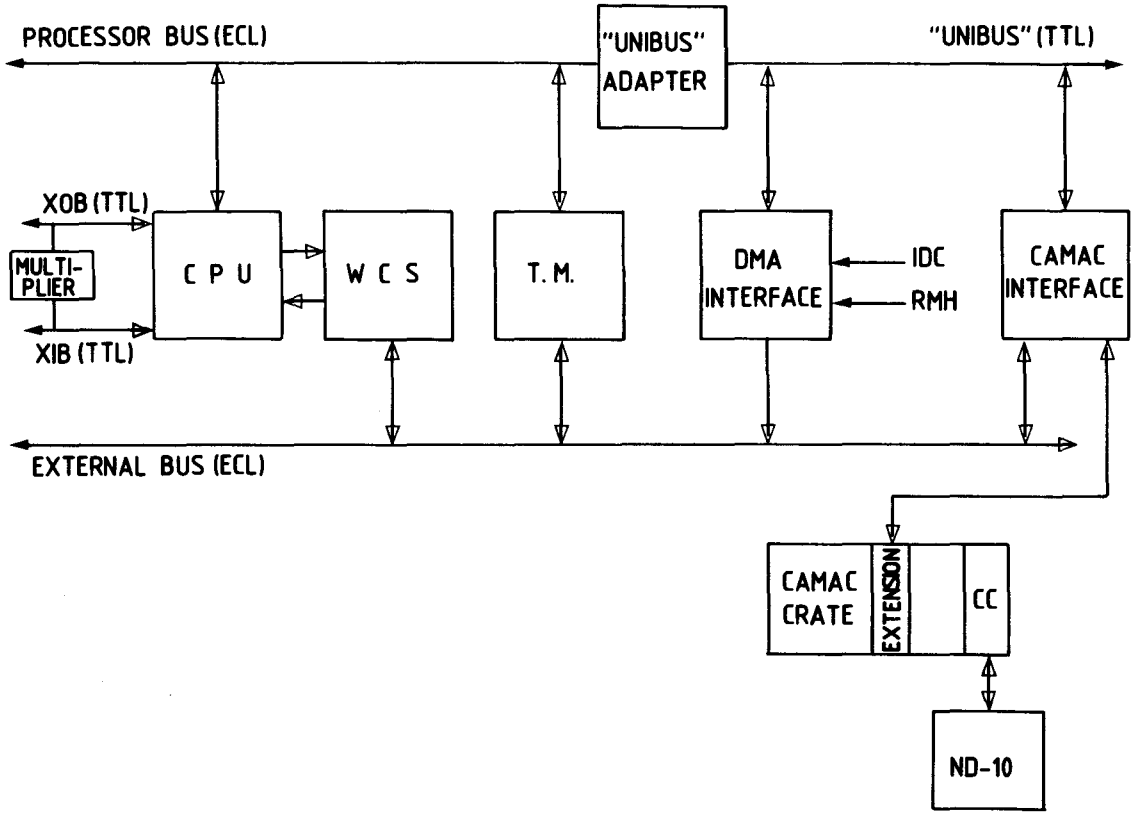

PROCESSOR BUS (ECL)
"UNIBUS" ADAPTER
"UNIBUS" (TTL)
XOB (TTL)
MULTI-PLIER
C P U
W C S
T. M.
DMA INTERFACE
IDC
RMH
CAMAC INTERFACE
XIB (TTL)
EXTERNAL BUS (ECL)
CAMAC CRATE
EXTENSION
CC
ND-10

Fig.1

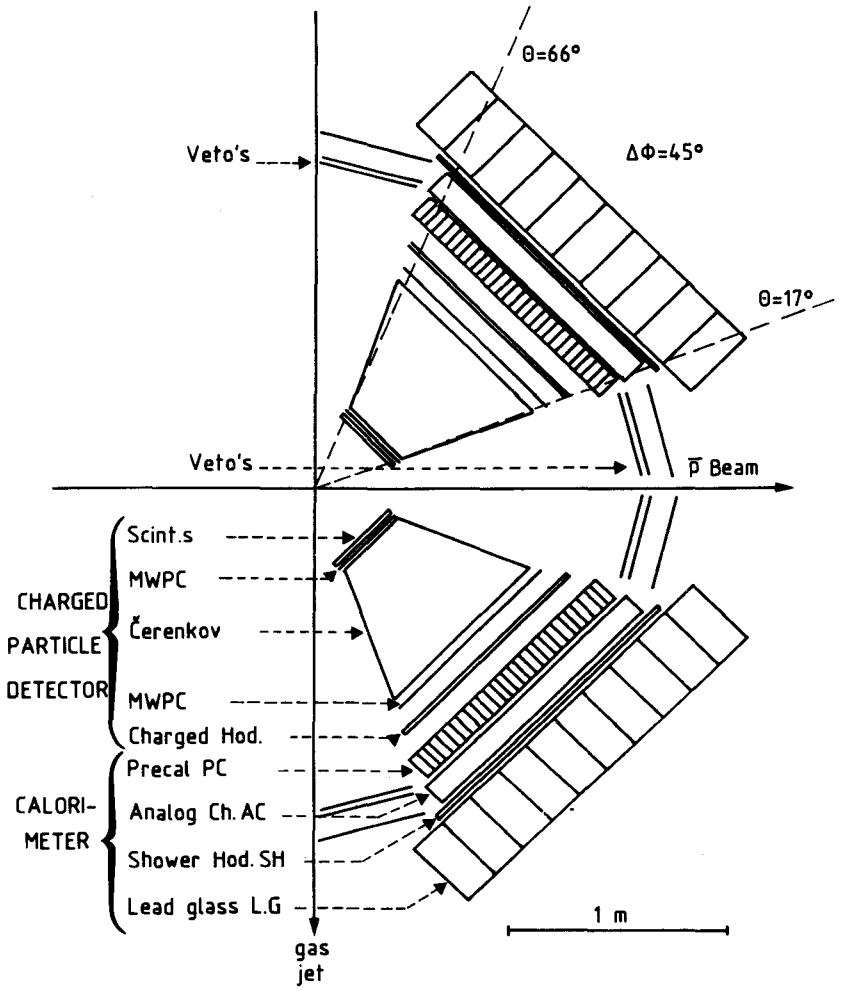

θ=66°
Veto's
ΔΦ=45°
θ=17°
Veto's
p̄ Beam
Scint.s
MWPC
CHARGED
PARTICLE
Čerenkov
DETECTOR
MWPC
Charged Hod.
Precal PC
CALORI-
Analog Ch. AC
METER
Shower Hod. SH
Lead glass L.G
1 m
gas
jet

Fig.2

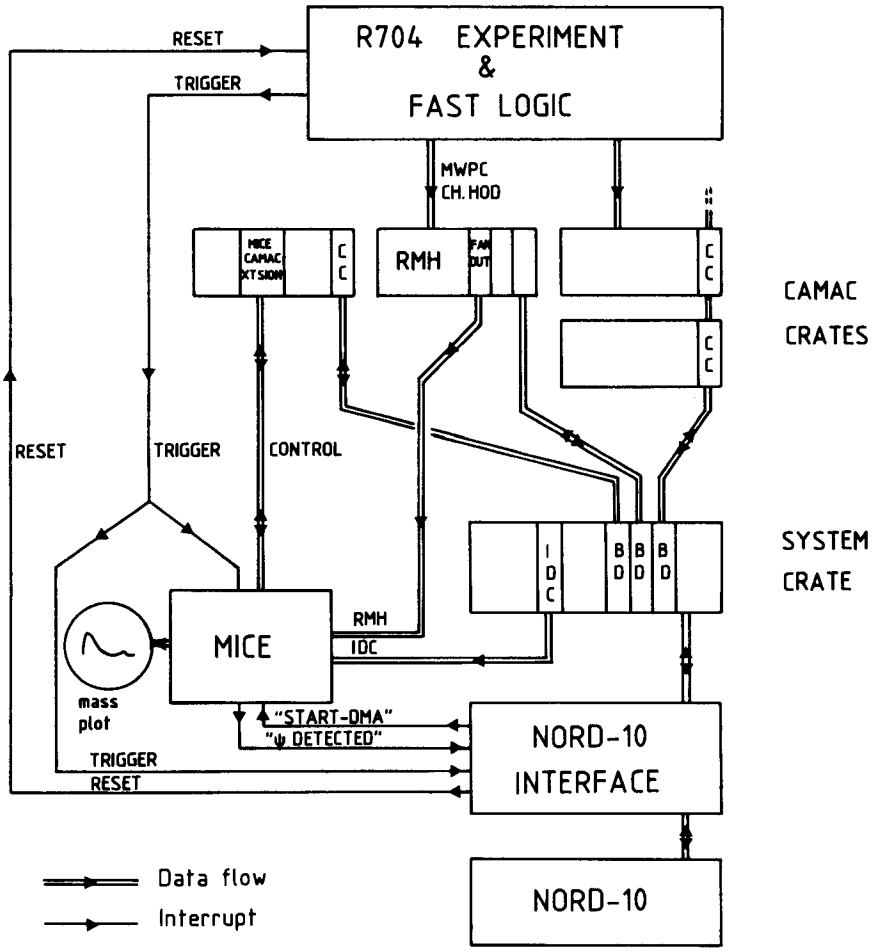

R704 EXPERIMENT
&
FAST LOGIC
RESET
TRIGGER
MWPC
CH. HOD
MICE CAMAC XTSION
C C
RMH
FAN OUT
CAMAC
CRATES
RESET
TRIGGER
CONTROL
I D C
B D
SYSTEM
CRATE
MICE
RMH
IDC
mass
plot
"START-DMA"
"ψ DETECTED"
NORD-10
INTERFACE
TRIGGER
RESET
Data flow
Interrupt
NORD-10

Fig.3

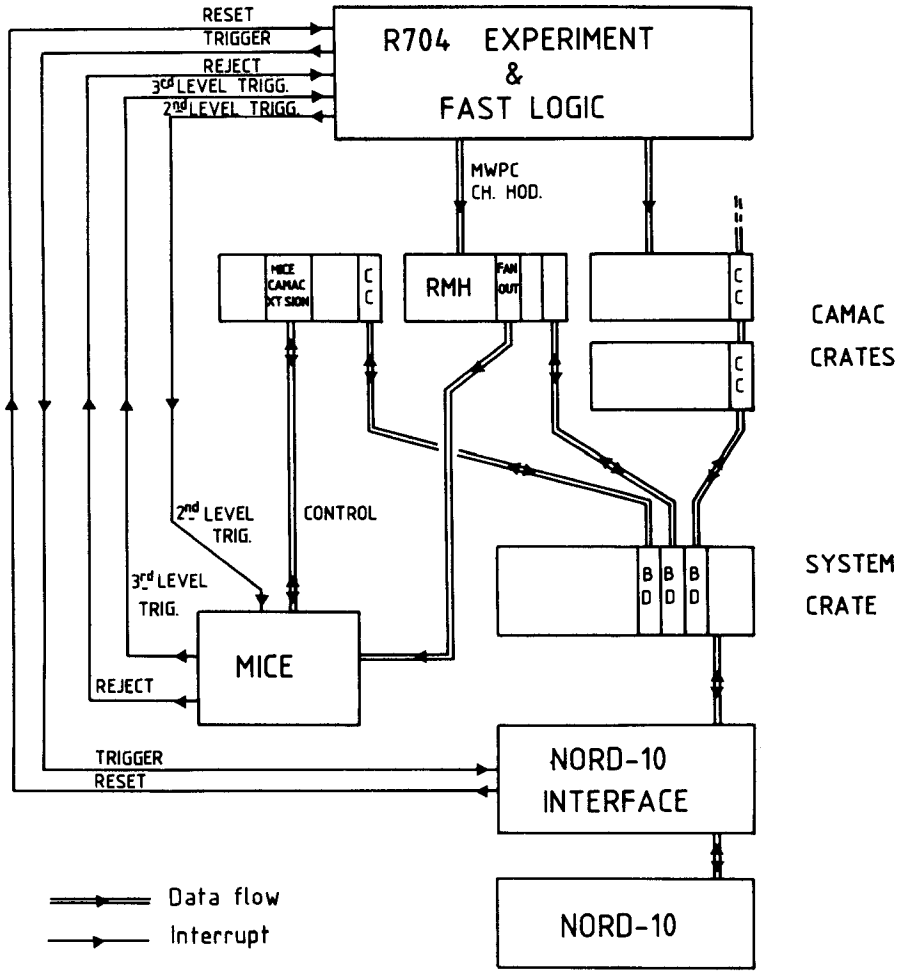

RESET
TRIGGER
REJECT
3rd LEVEL TRIGG.
2nd LEVEL TRIGG.
R704 EXPERIMENT
&
FAST LOGIC
MWPC
CH. HOD.
MICE CAMAC EXT SION
C C
RMH
FAN OUT
CAMAC
CRATES
2nd LEVEL
TRIG.
CONTROL
3rd LEVEL
TRIG.
B D
SYSTEM
CRATE
MICE
REJECT
TRIGGER
RESET
NORD-10
INTERFACE
NORD-10
Data flow
Interrupt

Fig.4

QUESTIONS AND ANSWERS

Q: Was the reconstruction program written in FORTRAN?

R. Verkerk

A: Half in FORTRAN. - for spatial reconstruction and kinematics; half in PDP Assembly code - for key routines (interrupt handler and square root calculation, for example).

TRIGGER AND SPECIALIZED COMPUTING HARDWARE FOR THE COLLIDING DETECTOR AT FERMILAB

Myron Campbell
The Enrico Fermi Institute, The University of Chicago

I. Detector Description

The collider detector at Fermilab will provide the first look at 2 TeV center of mass $\bar{p}p$ collisions. The detector has complete calorimetry coverage from 2° to 178°. The central region is instrumented with a silicon vertex detector, a vertex time projection chamber and a large central tracking chamber. A central muon chamber covers the region between 51° and 129°. Muon coverage in the forward region is provided by toroids from 2° to 10° and 170° to 178°. A more detailed description of the detector can be found in the CDF design report.[1]

The $\bar{p}p$ collider at Fermilab will have up to 6 bunched beams with 3.5μsec between crossings. At the design luminosity of $10^{30} cm^{-2} sec^{-1}$ and at 2 TeV center-of-mass energy we expect an inelastic interaction rate of 50,000 events per second. The total number of detector elements is approximately 75,000 with 10% occupancy for typical events. The expected rate for events being written to tape is between one and five Hertz. The trigger system and online processes must perform this factor of 10,000 reduction in event rate.

II. Overview of Data Acquisition System

The trigger electronics and data acquisition system are built to allow the detector to be partitioned into several independent systems. This will be important in the early stages of detector operation when the several different parts of the detector are being brought online. Also operation of the majority of the detector will not be inhibited by the failure of one part. The partitioning scheme will be useful later for maintaining calibrations of the various parts of the detector.

The detector can be divided into a maximum of 64 subunits, each of which can be assigned to any of up to 16 independent partitions. The large number of subunits allow for minimal disruption of the main detector when a piece must be removed for testing or calibration. The limit of 16 independent partitions is set somewhat arbitrarily by FASTBUS broadcast protocol.[2] The cleavage lines for splitting the detector are determined by the scanner and front end card assignments: a scanner and its associated RABBIT crates[3] can be in only one partition. The central calorimeter is divided into 48 wedges each wedge containing the electromagnetic and hadronic calorimetry, strip chambers, and muon chambers. The electronics for the rest of the detector is divided mainly along detector component lines.

CDF is designed to be a multi-purpose, multi-use detector. During data taking there can be several processes active each looking for different types of physics processes and each having different types of signatures. All detector components will be then in one partition. The different classes of events will be selected and tagged by separate trigger algorithms. The events will then be routed to the appropriate consumer process. The consumer process could be in the form of a level 3 trigger which would either reject or pass the event on to be written to tape or an analysis process which maintained online tables and histograms.

The computing power needed to analyze the data acquired from a one month run at 5 Hz is estimated to be one year of available offline resources. It is important to be critical in the event selection to reduce the number of events needing reconstruction.

Dedicated hardware for event reconstruction in the data acquisition system would also greatly reduce the offline computing requirements.

III. Trigger

The trigger system is divided into three levels.[4] The first level examines the signals from selected parts of the detector via dedicated cables. A fast decision based on analog processing and coincidence logic is made in the 3.5 microseconds between crossings. The level one trigger is therefore deadtimeless. Upon receipt of a level one accept the front end electronics is prohibited from resetting and hold the event data in analog sample and holds. The level two trigger then generates a more detailed description of the events. The list used by the level two processors contain information on calorimetry energy clusters, muon candidates and high momentum tracks from the central chambers. These lists are scanned by up to 4 independent trigger processors each of which can be programmed to look for different signatures. Upon the generation of a level two accept the event readout is started. The third level trigger operates on all of the information from the event after it is digitized and read via the scanners.

During the level two trigger and event readout the detector, or the partition involved if multiple partitions are running, incurs deadtime. The goal is to keep the total deadtime at 10%. Contributions to deadtime due to the trigger and the readout are shown in figure 1. Level one reduces the event rate by a factor of ten without deadtime. During the early running of the collider when the luminosity is $10^{28} cm^{-2} sec^{-1}$ level one will be sufficient to reduce the event rate to 50 Hertz. At the design luminosity of $10^{30} cm^{-2} sec^{-1}$ two options are shown for different types of track finders. A simple hardware straight road track finder will have results at the start of level two such that cuts on muon and electron candidates can be made early. A more sophisticated track reconstruction processor using curvature modules incorporating shift registers will not have results available until later in the level two analysis. The difference in event rate is from the expected muon and single electron backgrounds which cannot be rejected without central tracking information.[5]

The trigger system is based on the projective calorimetry of the detector. The electromagnetic and hadron towers are divided into a logical 24 x 42 array of $\Delta\phi = 15°$ and $\Delta y = 0.2$. The energy deposited in each tower is represented by a D.C. voltage and transmitted to the trigger system by dedicated cables (figure 2). The receiver circuits have programmable gains and offsets. The gain is used to convert energy to transverse energy, E_t, by multiplying each signal by $\sin\theta$ where θ is the angle from the proton direction. The offset is used for removing biases during running and injecting test signals during testing. The digital values for $\sin\theta$ and offset are loaded via FASTBUS.[6]

The resulting E_t signals are used in analog processing. The level one trigger sums together all towers in the detector over a preset threshold to generate the total transverse energy. The towers are also weighted by $\sin\phi$, $\cos\phi$, and y to produce the ϕ and y components of energy to be used for a missing energy trigger. The resulting sums, ΣE_t, $\Sigma E_t \sin\phi$, $\Sigma E_t \cos\phi$, $\Sigma E_t y$, are compared to global thresholds and used to generate level one accepts. There are four independent sets of analog summers. The threshold for including a tower in the global sum can be set differently for each of the analog sums. This will allow the total energy threshold for a level one accept to be set high for diffuse events which include a large number of relatively soft particles and still accept events which have a lower total energy but where the energy is concentrated in one or two regions.

After a level one trigger is generated and the front end electronics have been prohibited from resetting, the level two trigger will start. The first task of the level two trigger system is to generate a brief list or table which describes the event. The information generated for the calorimeter is the total energy, position, and width of each cluster

of energy. These are described by $E=\Sigma E_t$, $\langle y\rangle=\frac{\Sigma Ey}{\Sigma E}$, $\langle \sin\phi\rangle=\frac{\Sigma E\sin\phi}{\Sigma E}$, $\langle \cos\phi\rangle=\frac{\Sigma E\cos\phi}{\Sigma E}$, $\langle y\rangle^2=\frac{\Sigma Ey^2}{\Sigma E}$, and $\langle\theta\rangle^2=1-\frac{(\Sigma E\cos\phi)^2+(\Sigma E\sin\phi)^2}{\Sigma E^2}$.

The procedure used for finding and summing clusters is now described. The value representing E_t from the receive and weight card is compared to a programmable reference. The output of the comparator is sent to the cluster finder via a digital bus. The first reference level is set high and the towers over this threshold are latched in the cluster finder as seed towers. The threshold is then lowered to a minimum value and towers over this low threshold are stored in the cluster finder's second set of latches. The bus drivers in the compare and sum card are turned off and the cluster finder now returns sets of connected towers. A tower is in a cluster if it is either the selected seed tower or a low threshold tower adjacent to a tower already included in the cluster. The time needed to locate and define a cluster is about 100 nanoseconds and the time needed to make the analog sums for a cluster is about 600 nanoseconds. In order to provide a better match between these two sets of hardware four independent summers are used working in parallel. The throughput for the analog sums is then one cluster per 150 nanoseconds.

The 2016 towers of electromagnetic and hadronic calorimetry are divided between ten FASTBUS crates of analog sum modules. The results of each crate are digitized by the crate sum module and sent to the list makers. The list makers take the serial information coming from the crate sum modules and enter it in a master list. When a cluster crosses a crate boundary the list makers must combine the information into a single entry. The master list is made available to the level two processors.

It is at this master list that all information to be included in the trigger must come together. Muon candidates from the muon chamber and high P_t charged particle track from the central track finder together with the calorimeter information proved a powerful handle on selecting specific signatures.

The muon candidates are generated by requiring a coincidence in two sets of drift cells such that a line pointing to the interaction region is reconstructed. The P_t cutoff and multiple scattering limit can be controlled by adjusting the width of the overlap coincidence in drift times. The signal used for level one will be the OR of all coincidences indicating a muon candidate. During the construction of the calorimeter list the ϕ and Y coordinates of the muon candidates will be generated. The energy in the corresponding electromagnetic and hadronic towers can then be examined, checking that they contain only energy equivalent to a minimum ionizing particle.

There are two methods proposed for producing the list of high P_t tracks from the central tracking chamber. The first method is a sophisticated processor using curvature modules similar to the kind of track finder used at MARK II. This device would reconstruct all tracks, not just the high momentum tracks. The tracking information would not be available to the level two processors until after the lists were constructed. The high multiplicity of particles in an event in a hadron collider would seem to prohibit being able to trigger on low momentum particles - unlike electron-positron colliders where soft pions can be used to tag ψ' or D^* decays. The second type of track processor would look at the first wire in each of nine superlayers which collected charge[7]. A coincidence of wires matching one of 504 predefined roads determined by the outer drift tubes indicates a high momentum particle. The results of this track finder would be available at the beginning of level two processing.

The level two processors are dedicated to the trigger system. The internal architecture is ECL based with the micro sequencer using a bit slice sequencer sequence, the MC10801. The processing element include FASTBUS IO controls, special hardware for manipulating the cluster sum information, and a general arithmetic and logical

processing unit based on the MC10900. The micro- instruction cycle time is 12 nanoseconds. The output of the level two processors will be a level two accept or reject. A level two accept will result in the event being digitized.

IV. Data Acquisition

The majority of the front end electronics is in the RABBIT system (Redundant Analog Bus Based Information Transfer system). There are two separate and independent analog busses in a RABBIT crate, each controlled by a separate module, the EWE. The EWE's are controlled by the scanner (MX)[8] via independent communication links. Each analog channel is then accessible through two separate paths. During normal operation the two paths will be operation in parallel, however, in the event of a component failure in one path all information is accessible through the remaining link. This is important because the front end electronics is located in the collision hall which will have only restricted access. The analog cards designed for the RABBIT system are the strip chamber ADC, muon chamber ADC/TDC, hadron calorimeter ADC/TDC, and electromagnetic ADC.

The scanner used to control and read the RABBIT system via the EWE is called the MX. This is a micro-coded ECL machine specifically designed for this task. The data collected from the EWE is corrected for gain shifts and offsets then stored in one of four buffers from which the event data is assembled by the event builder. The MX has six separate memories (figure 3). Three memories, the DMA, DMB, and DMC, store the operands needed for the a + b*c correction. The list of commands needed to control the EWE is in the UM memory. The event memory, EM, holds the assembled event. The MX's micro-instructions are stored in the 2048 word by 64 bit IM memory which can be loaded via FASTBUS. The instructions are 64 bits wide; the first 32 bits specify the op codes and the second 32 bits specify the operand addresses.

The synchronization between the trigger system and the MX is managed by the trigger supervisor. The start scan message is delivered via a FASTBUS broadcast and the MX done signal is returned via dedicated cables to the trigger supervisor.

Other non-RABBIT electronics must obey the scanner protocol involving receiving a broadcast message, storing data in quadruple buffers and returning done signals. The LeCroy 1800 FASTBUS system[9] is being considered for the tracking readout system. However, since the 1800 system does not obey the scanner protocol a FASTBUS to FASTBUS interface is required. A device under development at SLAC, the SSP[10] (SLAC Scanner Processor), is being considered for use as the 1800 system scanner. This module implements the IBM integer instruction set and has a cycle time of 150 nanoseconds for most instructions. It is a FASTBUS master and slave on both a cable segment and crate segment[11].

V. Level Three Processors

The level two trigger system receives incomplete detector information via dedicated cables. After the event is constructed by the event builder the full data set, gain and offset corrected, in a formatted form is available. A third level trigger using the complete event data will be needed to reduce the rate to a manageable level.

The third level trigger must provide flexible event selection using algorithms developed in a high level language. A processor for level three must have enough memory to contain the 100 to 200 kilobytes of data from an event. The processor must be able to accept the data at 20 megabytes per second from FASTBUS. Current plans for the level three system is a collection of identical processors which are loaded by the buffer manager and pass accepted events to the host VAX[12]. CPU's under consideration are the 3081/E currently under development by a CERN-SLAC collaboration[13], the

168/E and the 370/E. All of these depend upon having FASTBUS interfaces; the choice remains to be made.

VI. Summary

The high event rate and large multiplicity of events expected at CDF present new challenges in online triggering and processing. To meet this challenge an array of new dedicated analog and digital processors are being built. The trigger involves a combination of fast analog and digital techniques. The data acquisition system is designed to provide corrected data from the redundant analog front end.

REFERENCES

1. Design report for the collider Detector Facility (CDF), D. Ayres, et al., Fermilab internal document, CDF 111. August (1981)
2. Design of the Collider Detector Facility Data Acquisition system, A.E. Brenner et al., Fermilab internal document, CDF 108. December (1981)
3 RABBIT System Specifications, T. F. Droege, K. J. Turner, T. K. Ohska, Fermilab internal document, PIN-52. July (1983)
4. Status of the CDF Trigger, M. Campbell, et. al., Fermilab internal document, CDF-157. February (1983)
5. Central Tracking for Electron Triggers, M. Shochet, University of Chicago internal document, TGN-13. March (1983)
6. FASTBUS Modular High Speed Data Acquisition and Control System for High Energy Physics and Other Applications, U.S. NIM Committee, DOE/ER-0189. December (1983)
7. Charged Particle Tracking in CDF, M. Atac, et. al., Fermilab internal document, CDF-178. September (1983)
8. The MX Hardware Description, K. J. Turner, Fermilab internal document, PIN-116. February (1894)
9. System 1800 Data Acquisition System, LeCroy Research System Corporation, April (1983)
10. FASTBUS Interface for TDC Crates, T. Carroll, Fermilab internal document, CDF-214. February (1984)
11. SLAC Scanner Processor, T. Glanzman, SLAC internal document, MKII/FASTBUS Note #2. October (1983)
12. Level 3 Processors, T. Carroll, Fermilab internal document, CDF-188. October (1983)
13. The 3081/E Processor, P. E. Kunz, et. al., paper presented at Impact of Specialized Processors in Elementary Particle Physics, Pavoda, Italy. (1983)

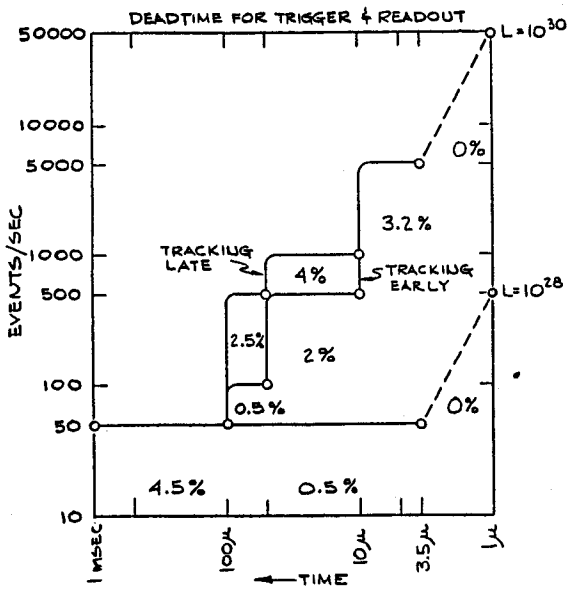


Figure 1. Dead time for trigger and readout for various event rates. Note time increases from right to left.

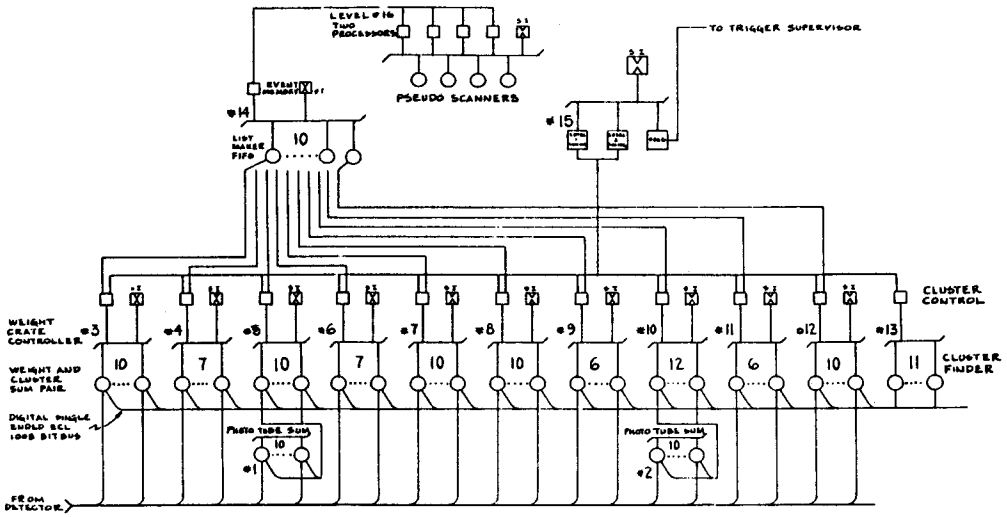


Figure 2. Diagram showing all crates involved in the calorimetry trigger. S.I.'s are segment interconnects.

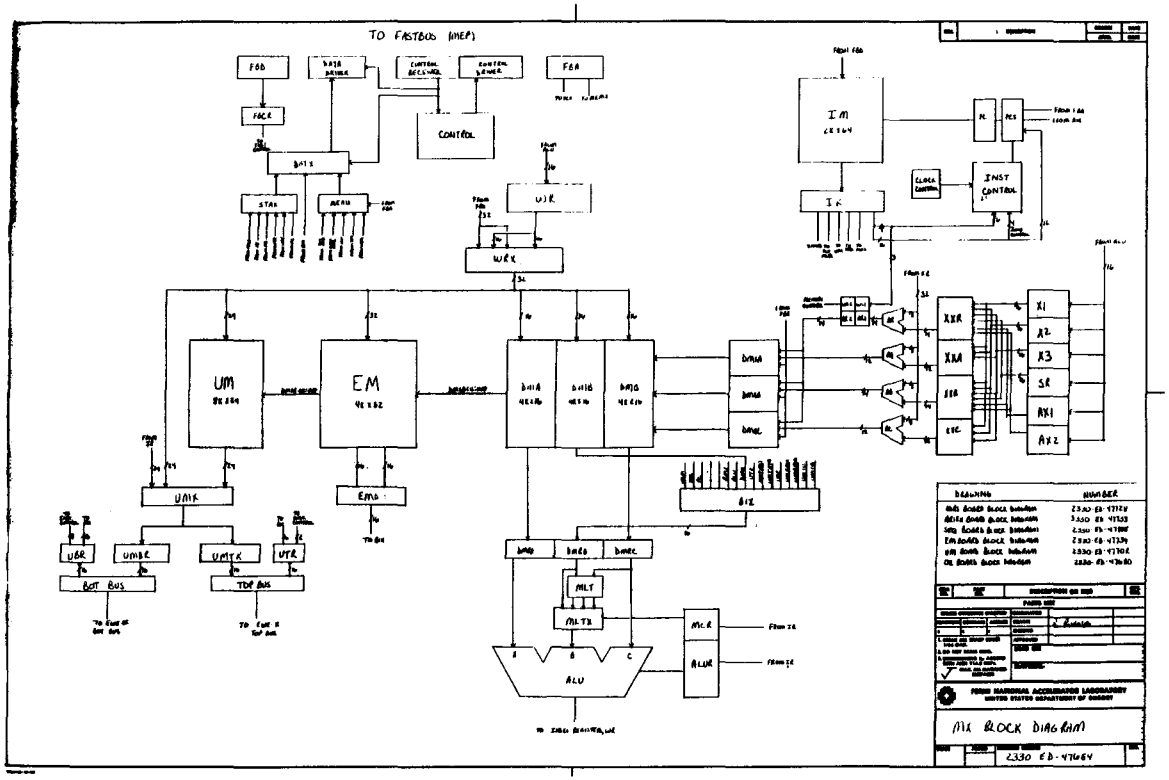


Figure 3. Scanner (MX) block diagram

QUESTIONS AND ANSWERS

Q: How does MX compare with, say, a VAX? Could the MX have been replaced by a commercially available processor?

D. Kaplan

A: Simulation indicates the MX is 2-3 times as fast as a 3081E, or around 10 VAX 780's. But probably the main motivation for building was that it was fun for the people who did it.

THE D0 EXPERIMENT : ITS TRIGGER, DATA ACQUISITION, AND COMPUTERS*

Authors:
D. Cutts,† R. Zeller, Brown University; D. Schamberger, S.U.N.Y. - Stony Brook;
R. Van Berg, University of Pennsylvania (presented by D. Cutts)

Abstract:

The new collider facility to be built at Fermilab's Tevatron-I D0 region is described. The data acquisition requirements are discussed, as well as the hardware and software triggers designed to meet these needs. An array of MicroVAX computers running VAXELN will filter in parallel (a complete event in each microcomputer) and transmit accepted events via Ethernet to a host. This system, together with its subsequent offline needs, is briefly presented.

THE D0 EXPERIMENT

The D0 project is an experiment to study pp interactions at the D0 intersection region at Fermilab's Tevatron-I colliding beam accelerator. The collaboration at present consists of 12 institutions[1] with Paul Grannis of Stony Brook as spokesman. It is fair to say that this experiment is the "most approved" of any current project. Because of funding uncertainties we have over the past year been repeatedly reviewed, always with positive results; and I'm happy to say that finally appropriate commitments are forthcoming.

At the Tev-I, the expected signatures of new physics at 2 TeV involve high p_t leptons and jets (as from massive intermediate states) or large missing p_t (carried off by neutrinos, say). As will be seen, these event characteristics will be used by the hardware trigger; but more fundamentally they provide the basic goals for our detector design; to optimize the lepton (electron and muon) and photon identification and energy measurements at the same time as maintaining the best possible energy and angular resolution for jets.

To be both competitive with and complementary to the other detector at Tev-I (CDF, at intersection region B0) we have concentrated on a design with superior lepton measurements and calorimetry, while doing without a central magnetic field. The detector will have three basic divisions, beginning with a central tracking section surrounding the interaction point and extending radially to 70 cm. This device will include a transition radiation detector providing an independent hadron/electron rejection of order 50/1. Surrounding the central tracking will be a uranium (and copper)/liquid argon calorimeter extending over the full angular range, to within 1 degree of the beam. This calorimeter will be divided in two sections: an electromagnetic region with fine sampling and a hadronic part with coarser sampling. Outside the calorimeter will be a muon detector, consisting of planes of proportional drift tubes spaced about three magnetic iron torroids. A full description of the detector is available[2].

To give a measure of the real strengths of the detector, some parameters for the central calorimeter and the muon detector are shown in Table 1. Note the superb calorimeter energy resolution, the fine segmentation in towers, with multiple samplings, and the near equality of electromagnetic and hadronic response, a feature of this type of calorimeter. The muon detector extends over nearly the complete angular range; with the large total absorption length, hadronic punch-through is minimized.

* This work has been supported in part by grants from the Department of Energy and the National Science Foundation.
† Presenter

DATA ACQUISITION OVERVIEW

The D0 detector, with approximately 100,000 separate channels of electronics, will reside in a region where the interaction rate is of order 50 KHz. In the detector, the typical size of an interaction event will be rougly 60 K bytes, including both digitized data and address. We will handle this data flow with two sequential trigger systems, the first done quickly with hardware and the second more slowly with software. Given an interaction, fast output signals from summed elements of the detector will flow into custom designed hardware capable of filtering events with a rejection factor of roughly 250 within the time between beam crossings (3.5μsec). Although complicated, such an electronic trigger is standard and straight forward to build (see Figure 1).

In order to reduce the number of events saved on tape or disk to one or two per second, a further selection will be necessary, with a rejection factor of order 100/1. Implementation of this step will also be straight-forward, since it will be based on commercially available (and supported) hardware and software. At this stage, event selection will be made with a filter program, with of order 100K - 200K instructions on average needed per event. Several microcomputers, capable of handling the filter requirements, are commercially available now or in the near future. These include projected MicroVAX units as well as those based on the Intel 286/386 series and on the Motorola 68000 series. Analysis in these microcomputers will be done in parallel, each essentially an isolated system receiving an event directly and analyzing in completely. We have chosen as this microcomputer the MicroVAX[3] ;for the Level-2 trigger we will have an array of MicroVAXes each running VAXELN and each linked via Ethernet to a central host VAX, which will collect those events surviving all filtering.

The structure of the data acquisition system is shown in Figure 2. The fast electronics digitization and readout crates will be grouped into 7 sections; each section reading out on its own data cable, which will be daisy chained to each microcomputer in the array. A separate MicroVAX, the "Acquisition Supervisor", will control the event flow - allocating the data cable within the readout section and assigning which analysis MicroVAX will receive the event. For its control functions, the Supervisor will be interfaced to all elements of the acquisition system: the Level-1 trigger, the readout crates, and the Level-2 analysis units.

LEVEL-1 TRIGGER

The hardware trigger decision will be made in the 3.5 microsecond interval between beam crossings. The fast response together with the double-buffering of the input analog signals will insure that deadtime associated with this trigger will be minimal. As input to the Level-1 trigger, signals from the calorimeter towers will be grouped in towers 4x4 laterally and summed in depth; thus there will be available a fast calorimeter signal from each dimension of 22.5 degrees in azimuth and 0.4 units of rapidity, for a total of 16x25 = 400 e.m. and 400 hadronic trigger towers. Also present will be data from the TRD, from the muon detector, from the interaction tagging scintillation counters (the "Level-0" trigger) and from accelerator monitors, as well as a 3-bit vertex position measurement from the central detector. This last data may not be available within the initial decision time, but may be used as a "Level-1.5" since a corrected angle can make a significant improvement in transverse energy calculations.

Figure 1 shows the design of the Level-1 trigger[4]. In addition to the detector data, various signals which will be preset externally control the response of the trigger hardware to each event. Listed in Table 2, these signals define such parameters as thresholds for

transverse energy or transverse momentum in the electromagnetic and hadronic calorimeters, and minimum energies for the "jet finder". These signals also will allow the definition of particular trigger types as some combinations of the separately-tested event characteristics.

LEVEL-2 TRIGGER

The Level-2 trigger will be accomplished by routing the complete data for one event from the digitization crates to one microcomputer which runs the entire filter program. In this mode the micros will operate in an independent but parallel fashion. As discussed previously, of several commercially available microcomputer systems, we have chosen for these units the QBUS based MicroVAX family. We expect that its performance will be comparable to that of a VAX-11/780, that it will have sufficient direct memory for the complete analysis program and event data, and that it will be available on a time scale suitable to our experiment. To provide a means of loading the data into the analysis unit, one of us (R.Z.) has designed a dual-port memory board using 64K static rams. The external port cycles at 100 nsec. with a 32-bit wide input, for a bandwidth of 40 Mbytes/sec. We plan to have eight dual-port input channels per Level-2 box loaded in parallel with event data and accessed by the MicroVAX via QBUS block mode transfers.

The MicroVAX units will be interfaced to the digitization crates via data cables which carry a few control lines supporting a simple protocol (no handshaking) as well as the 32-bit wide data lines. Each of the units will be linked to a host VAX via Ethernet for shipment of events which pass the software filter as well as for initial program downloading. To enable real-time control of the acquisition process, each unit will also be linked (to the Supervisor) via "extended registers". We expect to incorporate in the system approximately 50 MicroVAX processors to meet the Level-2 trigger requirements.

As indicated above, the Supervisor will be a separate MicroVAX (running a separate VAXELN task) which will exercise the data acquisition control functions, such as deciding:

1. which Level-2 unit is free and should receive the next event;
2. which Level-1 triggers should be enabled;
3. given a Level-1 trigger, which electronics readout crates should initiate digitization;
4. as digitizations complete, which crate should use the data cable associated with its readout section.

In addition to the seven data cables associated with the readout crates, one cable will be used by the Level-1 trigger to ship the fast trigger data (about 4 Kbytes) directly to the Level-2 analysis unit. Preliminary analysis of this data can give rise to a fast abort signal communicated to the Supervisor, which in turn could terminate unfinished readouts and reset the digitization buffers.

Another way in which the Supervisor will exercise a central control over the data taking is its loading an external "fast trigger memory". This simple interface acts as a matrix, with one dimension being lines associated with each of the different hardware triggers and the other dimension providing the corresponding patterns of electronics crates which should initiate digitization. For calibration and test data, for example, the Supervisor will be able to allocate a specific, single crate to digitize and, if useful, to route the data

to a specific analysis unit. Indeed, for testing, the Supervisor will also be able to direct a given event to be read into a number of Level-2 systems and analyzed identically. Because of its central role in the acquisition, the Supervisor will be connected to the host VAX via a separate Ethernet line.

A number of additional interfaces will be developed for the acquisition system. The various detector readout devices will transmit data via data cables described earlier. The simple protocol of these cables will allow straightforward interfacing of the special calorimeter electronics[5], CAMAC, and FASTBUS. To enable control of the digitization and readout process, and to permit downloading to local crate intelligence, interfaces similiar to the "extended register" used in the analysis units will couple each of the electronics crates to the Supervisor.

A key element in our design is the use of commercially available system software. With our choice of the microcomputer system, the new DEC software package VAXELN will provide the complete framework for development and operation of the Level-2 analysis software, encompassing the host VAX and all the satellite MicroVAXes. This "software product for the development of dedicated real-time systems for VAX processors"[6] runs as a task under VMS on the host VAX. The task image is then downloaded to the appropriate MicroVAX using Ethernet. The analysis software can be written entirely in high level languages: "EPASCAL" for the control functions and FORTRAN for the Level-2 filter code. Downloading from the host to the MicroVAXes is supported as is remote symbollic debugging. The VAXELN program in each of the MicroVAXes has little system overhead, yet is multitasking (each interrupt service routine can be a separate task), and has transparent Ethernet support. Thus a user at a terminal on the host can access any MicroVAX, and the VAXELN program in a MicroVAX can read and write the host's disks and tapes. This latter feature might make MicroVAXes an ideal choice for an offline analysis farm: events can be read from the host's data tape by VAXELN analysis packages running in each MicroVAX. All these aspects of VAXELN make it very desirable, especially since it is a currently available and supported product.

ONLINE AND OFFLINE COMPUTING

We plan to incorporate a large VAX (hopefully the "VENUS" will be available) as the central online computer for the experiment. This machine will serve as a host, for program downloading and reception of events passing the Level-2 filter. The host will maintain direct overall control over the data taking (via the separate Ethernet link to the Supervisor). This VAX will carry a large assortment of necessary peripherals and support the usual online functions such as displaying data.

In an experiment of our size it is important to consider the implications of saving events. Even with an event filter having an overall rejection factor of 25000/1 we will still collect events at a rate of 2 Hz. During the roughly 3 months per year that D0 operates, we will then require of order 8 Cyber 175-equivalents per year for the complete analysis. This result is based on an estimate[7] of 16 sec. of Cyber 175 time per event; adding in other offline tasks (Monte Carlo, reanalysis, development, etc.) yields roughly 16 Cyber 175-equivalents per year needed by our experiment. Still, this translates to roughly 100 MicroVAX-years, which may in fact be obtainable by microcomputer farms similiar to that described here.

SUMMARY

The D0 experiment is facing data acquisition requirements that are extreme, forcing the use of parallel processors as the only commercially available means to obtain the necessary reduction in trigger rate. There is an extremely attractive option for parallel processing which will exist soon, an option with both hardware and software fully supported: the MicroVAX and VAXELN. We anticipate having a prototype acquisition system based on these products operational this year.

REFERENCES

1) Univ. Arizona, Brookhaven Nat'l Lab, Brown Univ., Columbia Univ., Fermi Nat'l Lab, Florida State Univ., Univ. Maryland, Michigan State Univ., Northwestern Univ., Univ. Pennsylvania, S.U.N.Y.-Stony Brook, Virginia Polytech. Inst.

2) "DESIGN REPORT, An Experiment at D0 to Study Antiproton-Proton Collisions at 2 TeV." The D0 Collaboration, December 1983 and later updates.

3) Units will be the "next in the family" after the currently available MicroVAX-I.

4) Questions about the design of the D0 hardware trigger should be referred to Maris Abolins or Dan Edmunds at Michigan State University.

5) Designed by the Columbia-Stony Brook (CUSB) group.

6) Digital Equipment Corp.: "VAXELN PROGRAMMING" page 1.1 (AA-Z451A-TE)

7) Bruce Gibbard, Brookhaven Lab. Questions about the analysis package or the central VAX should be referred to Bruce, while inquiries about the offline package should be made to Serban Protopopesou, also at Brookhaven.

TABLE 1. Some parameters of the DØ detector

Central calorimeter	Muon detector
depth: 7λ	coverage 8° -172°
energy res.: $.37/\sqrt{E}+.005$ (had)	$dP/P \approx 17\%$
$.11/\sqrt{E}+.005$ (em)	thickness: 13λ (90°)
seg.: 576 towers/4 sampl. (had)	18λ (10°)
3360 towers/4 sampl. (em)	punchthrough (min. E_π)
em/had. response ratio: 1.1	600 GeV (90°)
	5000 GeV (10°)

TABLE 2. Control Signals for the Level-1 Trigger

E_t electromagnetic thresholds	e.m. jet low energy cutoff
E_t hadronic thresholds	had. jet low energy cutoff
P_t electromagnetic thresholds	jet counter energy limits
P_t hadronic thresholds	AND-OR network programming
specific trigger prescaler ratios	
specific trigger enable register	

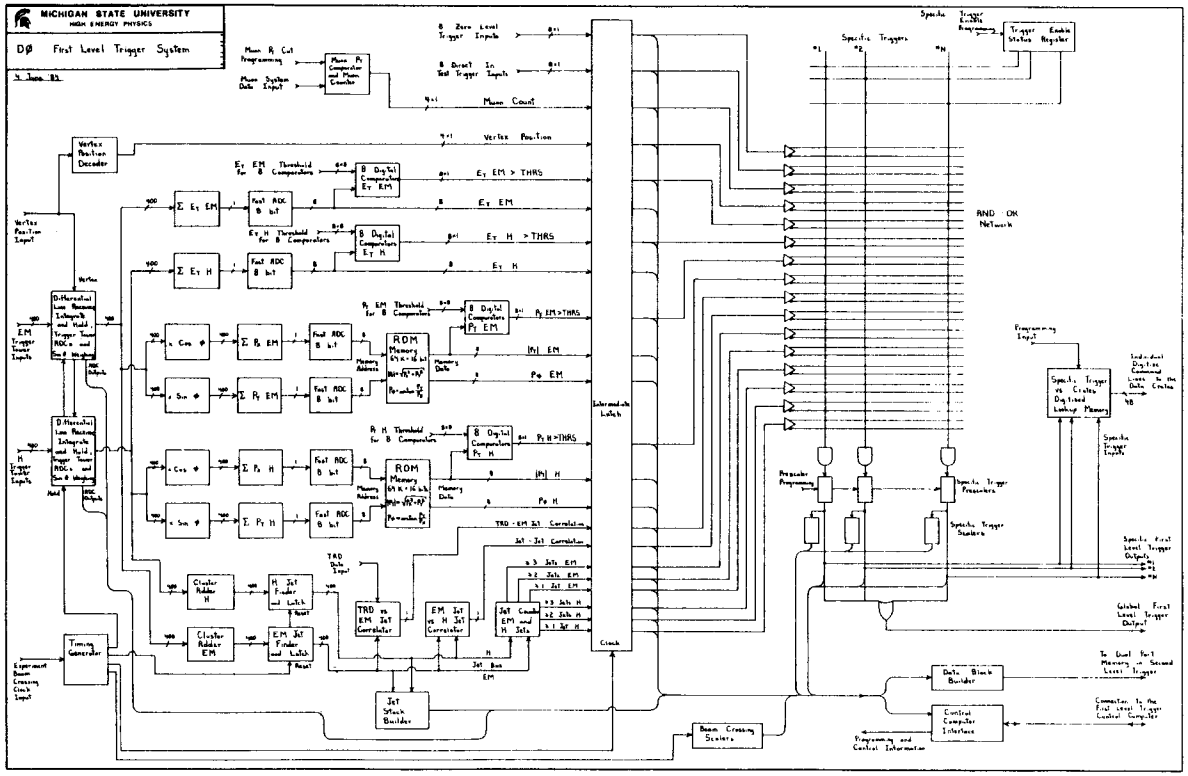

MICHIGAN STATE UNIVERSITY
HIGH ENERGY PHYSICS
DØ First Level Trigger System
Muon Count
Vertex Position
Et EM > THRS
Et EM
Et H > THRS
Et H
Pt EM > THRS
|Pt| EM
Pt EM
Pt H > THRS
|Pt| H
Pt H
TRD - EM Jet Correlation
Jet - Jet Correlation
AND OR Network
Intermediate Latch
Clock
ROM
Fast ADC 8 bit
Timing Generator
Vertex Position Decoder
Muon Count
Beam Crossing Clocks
Data Block Builder
Control Computer Interface
Beam Crossing Scalers

Figure 1.

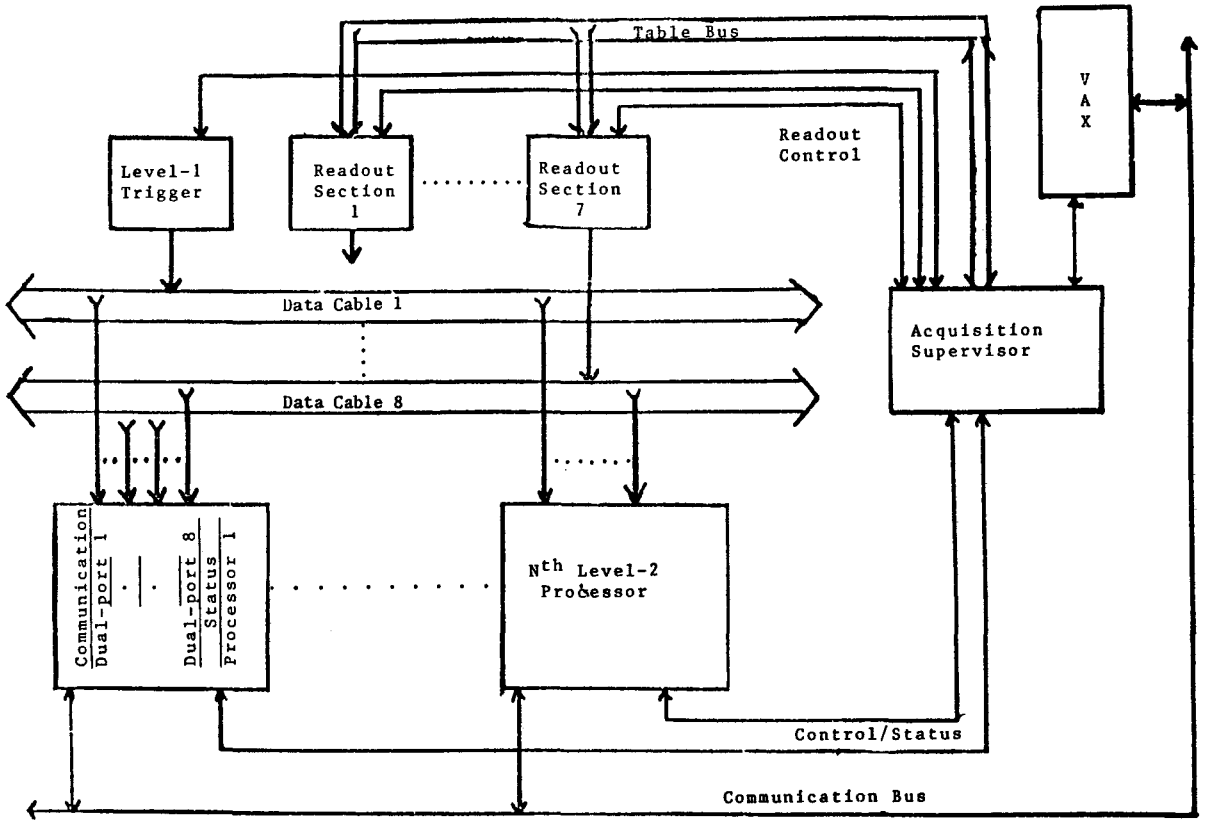


Table Bus
VAX
Level-1 Trigger
Readout Section 1
Readout Section 7
Readout Control
Data Cable 1
Data Cable 8
Acquisition Supervisor
Communication Dual-port 1
Dual-port 8
Status Processor 1
Nth Level-2 Processor
Control/Status
Communication Bus

Figure 2

QUESTIONS AND ANSWERS

Q: We have heard during the conference that DEC did not manage to achieve parallel processing on VAX's. What does that mean in your context?

H. Kasha

A: Each of our VAX's will process an event independently of the others. There will be no shared code. This works today.

Q: Does FORTRAN work under VAX ELAN?

A. Brenner

A: By writing a short main program in PASCAL a FORTRAN subroutine can be linked and run under ELAN.

THE UA1 VME-BASED DATA READOUT AND MULTIPROCESSOR SYSTEM

S. Cittolin, M. Demoulin, W.J. Haynes, W. Jank, E. Pietarinen, P. Rossi
CERN
1211 Geneva 23, Switzerland

1. Introduction

The UA1 experiment is a large multi-purpose particle physics detector system. It was designed to provide full solid-angle coverage around the LSS5 intersection region of the CERN proton-antiproton collider operating at a centre of mass energy of 540 Gev. The experiment has been operational since November 1981 and the next data-taking period is scheduled for September 1984. At present an upgrade program (4) involving the installation of new detectors and the improvement of the data acquisition system is in progress.

This paper describes the current upgrade of the data readout and online data handling facilities by means of a multicrate and multiprocessor system based on the industrial standard VME/VMX bus (1) and the M68000 microprocessor family.

The various parts of the detector and the data acquisition system have already been described elsewhere (2,3). In short, the apparatus consists of a central detector built of drift chambers with image readout to provide particle tracking, surrounded by a variety of complementary electromagnetic and hadronic calorimeters in both the transverse and longitudinal directions of the beam. A dipole magnetic field provides momentum analysis, and the whole detector is surrounded by an iron shield, which is being instrumented with multiple planes of Iarocci tubes, and a matrix of muon chambers (2). Additional drift chambers, in the forward and backward directions, complete the detector's 4π coverage. It is planned to insert a high resolution cylindrical drift chamber in the beam pipe in order to improve track reconstruction around the collision point.

The data digitization and formatting are performed by about 200 Camac crates grouped in 28 Remus branches (5). These are read in parallel into a multievent buffer system. A variety of specialized processors such as ROP (6) , FAMP (7), 168/E (8), Super Caviar (9) and NORD computers, share the tasks of event data compression, digital trigger, readout control and mass storage. The number of channels and the amount of data generated by the digitizing electronics for each event are given in Table 1. In Table 2 are listed the main components of the UA1 data acquisition system.

2. Trigger Levels and Data Acquisition Phases

At the design luminosity of the SPS proton-antiproton collider (10'30/cm'2 sec) the collision rate is of the order of 50 kHz, with a beam crossing every 3.8 μsec. The average luminosity obtained during the last data-taking period, in 1983, was 3 10'28/cm'2 sec, and an increase by a factor of 10 is expected for the next data-taking period starting in September 1984. This corresponds to a collision rate of about 15 kHz.

The maximum event readout rate is limited by the magnetic tape speed to about 4-5 Hz . In order to reduce the trigger frequency from several KHz to a few Hz, and maintain the overall system efficiency above 90%, the data acquisition operates in separate phases. Distinct levels of triggering take place before and during the data digitization, the data reduction, and the data readout phases.

A pre-trigger selects beam-beam interactions using standard NIM logic to demand a coincidence between hodoscopes in the proton and antiproton directions to within ± 20 ns. A first-level decision is then made between beam crossings, using a purpose-built processor system to fast identify energy distributions and prompt muon signals from the muon chambers. At this moment the data are in the phase of digitization and the data reduction and reformatting are enabled.

A second-level trigger decision can be activated only if a muon candidate is identified in the first-level, using M68000 microprocessors (7) to attempt to establish whether the muon candidate comes from the interaction region. At the same time data are formatted in parallel by a set of specialized processors and read into a multi-event buffer system. The system dead-time is determined mainly by the central detector data processing time (of the order of 35ms) However, the future incorporation of a hardware double buffer into each digitizer channel of the central detector readout will limit this dead-time to the ADC conversion time of 3ms. In order to reduce this dead time, further, to a few microseconds, the addition of an analogue double buffer to the calorimeter channels is also being studied.

Finally more refined decisions are made, based on the analysis of the event data by 168/E processors. These processors re-check the trigger condition matched in the preceding levels with greater precision and execute more sophisticated selection algorithms. This event filter is used either to flag or to reject an event. The maximum input rate to this phase of readout is about 30 Hz while the output rate, depending on the mass storage speed, is 4-5Hz.

3. Data Readout Structure

Figure 1 indicates the general structure of the data readout. Two systems

run concurrently during data acquisition: the parallel readout and the event data builder. They are controlled by two independent processor units: the readout supervisor and the event manager.

The Parallel Readout system consists of a set of detector busses autonomously driven by independent bus drivers associated with dual port memory units . These dual port memories are able to store several events, with a common port for data sorting. Data processing for reformatting and trigger selection can be executed in parallel by each driver unit accessing the data via a private bus. The readout supervisor handles the first level trigger signals, initializing the drivers for data readout, allocating the available memory for event buffering and controlling all the phases of data input. In addition this processor accomplishes the task of controlling and testing the digitizing electronics.

The Event Data Builder system accesses the parallel readout buffers and a set of event units, which are able to accept full event data for data acquisition and data sampling tasks. The event filter is part of this system. The event manager supervises the event building process, serving any event request coming from an event builder unit, initializing a multi-DMA transfer from the parallel readout buffers into the event unit memory, and starting and monitoring the event process. After the completion of an event transfer, for data acquisition, the event manager enables the corresponding multievent buffers for further trigger data input.

Such a structure was implemented from the beginning of the experiment in 1981, using Camac crates and Remus readout modules, with hardware FIFO memories for the parallel readout. The readout control processor was made of hardwired standard NIM logic. A Super CAVIAR microcomputer performed the task of event manager, controlling a stack of five 168/Es acting as an event builder unit. Other data sampling tasks were executed by a HP21MX and two NORD 100/500 computers, spying the data during acquisition. This solution was satisfactory in the previous data-taking periods with a trigger rate of the order of a few Hz, at first level trigger, and an overall system efficency of 90%. Its main limitations were the speed of the event readout at the third trigger level, at maximum 3 Mbytes/sec, and the difficulty of extending the system performance by including additional processing and control units. In order to cope with the higher rates expected in subsequent data-taking periods, and to allow for the modular expansion of the system in order to extend the second level trigger and the event data handling facilities, a hardware double buffer is being added to all the digitizer electronics channels that most affect the data reformatting dead time (namely the central dectector charge to time digitizers CTD (20)). In addition the data readout system is being partially rebuilt with the use of the general-purpose multiprocessor VME/VMX bus.

4. VME-VMX Bus

The VME bus is a 32-bit data and 32-bit address asynchronous multi-processor bus (20MHz bandwidth) introduced by industry (Mostek, Motorola, Signetics) in 1982 and now IEEE standard.

It is based on double size Eurocard mechanics. A large variety of general-purpose modules for processing, control, communication and display are commercially available. In addition, in order to improve the performance in applications where local data access and private memory extension are needed, a 32-bit data and 24-bit address local bus VMX is specified. Since the UA1 data acquisition improvement essentially involves the data readout part, and not the digitizer electronics system itself, the VME/VMX solution appears more attractive and suitable for this stage of the data acquisition system than a solution based on the high energy physics standard Fastbus. This does not exclude the use of Fastbus crates for digitizing and fast data compression in the front end of the data acquisition system.

The development of CAMAC-based equipment for UA1, and the monitoring and control of the experiment, made extensive use of CAVIAR microcomputers. With the introduction of the VME system, a successor to CAVIAR is being developed based on Apple Macintosh personal computers. This new system, called MacVEE (Microcomputer Applied to the Control of VME Electronic Equipment) allows up to eight VME crates and eight CAMAC crates to be directly memory-mapped into Macintosh address space.

The logical structure of the full readout system is shown in figure 9. It comprises the following four functional busses:

4.1.1) The Detector Bus links the digitizing electronics crates to a controller unit, the bus driver. It is a 16-bit data bus with a few control functions and sequential data access, its implementation depending on the standard of the digitizer electronics. For the old equipment of the UA1 experiment, the detector bus is a Remus vertical branch. For the upgraded detector program non-standard solutions have been chosen; these are the Iarocci STAR system readout (10) with an 8-bit data and 8-bit address special bus, and the microvertex detector LeCroy 1879 TDC readout, where a Fastbus crate with a VME/VMX interface is planned.

4.1.2) The Readout Control Bus is a general-purpose bus hosting all the detector bus driver units and the readout supervisor processor. The bus is used for readout initialization, calibration procedures and multi-processor control.

4.1.3) The Event Builder Bus performs the high speed data transfer between the multievent buffers and a requesting event builder unit memory, under the control of the event manager processor. Both the readout control and the event builder system are based on the VME bus. Physically they consist of several VME crates linked together with a crate interconnect system (figure 7).

4.1.4) The Local Bus is used for private data access between the driver unit, and the corresponding multievent buffer memory and the event builder processor, and its memory. The local bus is implemented by a VMX segment.

The system is composed of the following two modular elements:

4.2.1) The VME Parallel Readout Unit (VPRU) (figure 2) is the basic element of the parallel readout. It consists of a detector bus driver (DBD), a control processor unit CPU and a VME/VMX dual port memory (DPRX) of 128Kb size. This is suitable for storing several event data blocks coming from the associated detector electronics. The DBD and the CPU are connected via the VME port to the readout control bus and operate under the control of the readout supervisor processor.

The dual port memory VME port is connected to the event builder bus. The VMX port shares a VMX segment with the CPU and the DBD that act, respectively, as VMX primary and secondary master. An event unit may not have an associated CPU; in that case all the readout control is performed by the readout supervisor processor. During data acquisition each event unit takes care of the detector data readout and buffering. Data reformatting and/or second level trigger selection are executed at the end of readout by the CPU unit accessing data via its local VMX bus segment. The CPU is also responsible for the monitoring, calibration and system test tasks.

The detector bus driver can be a single VME module, such as the VME Remus branch driver used to read the old system, or it can have a substructure as in the case of the Iarocci STAR readout (figure 3) where an entire VME crate perfoms the function of the input driver. The full parallel readout system consists of 28 event units with VME Remus branch driver, a VME crate for the Iarocci detector and a LeCroy 1879 Fastbus VME/VMX Driver. Figure 7 indicates the physical layout of the VME crates.

4.2.2) The VME Event Builder Unit (VEBU) (figure 4) has a structure symmetric to the event unit. The event unit accomplishes the task of full event data analysis. It consists of a dual port memory accessible from the event builder VME system and from a VMX segment hosting a CPU. Optionally it can be accessed by an output driver unit for data communication. The dual port memory has a size of 256Kb, suitable for storing two full event records for double buffer operations. The CPU and output driver VME ports reside in an independent VME segment bus in order to avoid the event builder bus overloading during the event building data transfer. The task of an event builder is either to process a full event data record for data sampling, executing statistics, histogramming and display applications, or for data communication with the third level trigger and the mass storage. In general, a single event unit is associated with each task and several such units can be modularly inserted into the system without affecting the overall speed performance.

A special event unit is dedicated to the data acquisition and the third level trigger. In this case an output driver module acting as secondary master on the VMX bus accomplishes the data transfer (at 6 Mbyte/sec) between the dual port memory and the 168/E emulator stack. The latter is interfaced to a Camac system via an auxiliary crate controller (PAX) (8) (see figure 5). This represents a temporary solution to the event filter problem.

In the final configuration the 3081/E emulator (11) will be integrated into the system as a VME event unit processor (figure 6). Such a system can also be used for offline data analysis. Additionally new mass storage devices, such as digital video disk, can be included directly into the VME system when commercially available.

5. VME Components

Owing to the specialized functions of some of the units and to the recent introduction of the VMX local bus specification, not all of the VME modules needed for the implementation of this project were commercially available at the start of the data readout improvement program, in October 1983. Consequently an important development program has had to be pursued, partly within the UA1 collaboration and partly by commissioning specified projects to commercial firms. The module specification was finalised in December 1983 (12) and all the prototype modules where developed during the first half of 1984. At the time of writing production and installation are in progress.

The following modules were developed for the UA1 VME readout system:

5.1) Control Processor Unit CPUA1 (13).

The CPUA1 module is the basic processor unit used in all the VME readout subsystems. Its main features are: M68010 8MHz processor, VME/VMX bus master, 256Kb dynamic memory, 8Kb static memory dual ported CPU-VME, NS 16081 6MHz floating point processor, MK68901 peripheral controller (interrupt handler, timer, RS232 serial interface), a variety of control and status registers for CPU identification, address modifier control, VME bus arbiter control and VMX base address setting.

5.2) Dual Port Memory DPRX (14).

The DPRX is used both for the parallel readout and the event builder units. Its main features are: 128Kb/256kb static RAM VME/VMX dual port memory 32-bit data. 400 ns 32-bit word data transfer on both ports, programmable VME memory base address via a VME register, write broadcast mode implemented via address modifier selection (this mode allows the loading of several event builder units by a single DMA transfer, when all the corresponding memory units are set to the same VME address).

A version of the DPRX with 1Mbyte dynamic RAM is used as a local memory extension of an event builder CPU.

5.3) Remus VME/VMX Branch Driver RVMEX (15).

The RVMEX is used in the parallel readout unit as a detector bus driver. It is a multi-path REMUS-VMX, REMUS-VME and VME-VMX bus driver module.

It implements the Remus branch driver read/write functions and, acting as secondary master of the VMX bus, it handles autonomously the readout of a Remus branch and stores the data into an assigned VMX buffer. All of the module's functions are programmable via the VME port.

5.4) Crate Interconnect CI (16).

This module drives a high speed (10.7 Mbyte/sec) vertical bus allowing the linking of a VME crate (master) with up to 15 VME crates (slaves). The vertical bus master crate can operate in two modes (figure 8): the window mode and the DMA mode. In the window mode a master crate CPU can map 64Kb memory of a slave crate into a 64Kb segment of the master crate interconnect module. In the DMA mode any length of data block transfer can be executed under the master control between any two crates on the vertical bus.

The 32-bit data transfer has a three phase pipeline: the source data read (a VME cycle in the source crate), the vertical bus data transmission and the destination data write (a VME cycle in the destination crate).

5.5) 168/E VME/VMX-Camac Fast Data Link (17).

This system consists of two modules, a VME/VMX data output driver and a Camac data input driver. The VME module acts as a secondary master of a VMX segment and provides the DMA transfer between a VME/VMX dual port memory and a Camac module. The latter is read by the 168/E PAX auxiliary controller, linked to the 168/E emulator memory via a 'Greyhound Bus' (8). The maximum data rate is determined by Camac, and is of the order of 4Mbyte/sec. The modules are part of the VME event builder unit dedicated to the third level trigger and mass storage (figure 5).

5.6) VME/VMX Parallel Input Output VXPIO (18).

This is a general-purpose input-output module. It performs synchronous and/or asynchronous 16-bit TTL/NIM parallel I/O via a front panel connection and either VME or VMX ports. Both input and output are driven by a high speed FIFO of 512 16-bit words. This allows for the special application of the module as a data communication unit with an external system, or as a programmable output sequencer or as a logic/ state analyzer.

5.7) VME Interrupt Vector Generator IVG (19).

This module generates interrupts from 8 TTL/NIM front panel inputs at two presettable levels. For each channel the VME interrupt vector, the mask and a semaphore flag are programmable. Each input provides a channel status output signal and an internal counter allows the detection of double 'click' situations.

Conclusion

In an experiment such as UA1, the detector complexity, high trigger rates, and large data volume per event require distributed intelligence at many stages of the readout system. In addition a very modular and easily upgradable system structure is desirable, so that developments in technology can be readily applied to improve system performance.

It is believed that these aims can be achieved by the introduction of a readout system based on the industrial VME bus standard. This approach appears to be powerful, flexible, and cost-effective, and to prepare the way for the efficent exploitation of the enhanced potential of the UA1 detector in the future.

Acknowledgements

We would like to thank S. Centro, L.O. Hertzberger and B.G. Taylor for their useful contributions to the definition and specification of this project. The following CERN support groups and external organizations are collaborating in the design and development of the VME modules described: CERN EP/EL group for the VME Remus branch driver, Helsinki University for the crate interconnect module, Annecy LAPP electronics group for the parallel input-output module, CERN DD/ED group for the VME-CAMAC-168/E data link system and the assembly of VME crates, Saclay electronics group for the interrupt vector generator and Data Sud Systemes, Montpellier, for the CPU and DPRX modules.

Finally we are indebted to C. Rubbia for much encouragement and support in all the phases of this project.

References

1) VME Bus. MVEBS/D1, Rev. B Aug. 82 Motorola. VMX Bus, Rev. A Nov. 83 Motorola;
2) UA1 proposal: A 4π solid-angle detector for the SPS used as a proton-antiproton collider at centre-of-mass energy of 540 Gev, CERN/SPSC 78-6;
M. Barranco Luque et al., Nucl. Instrum. Methods 176 (1980) 175;
M. Calvetti et al., Nucl. Instrum. Methods 176 (1980) 255;
K. Eggert et al., Nucl. Instrum. Methods 176 (1980) 217,233;
A. Astbury, Phys. Scr. 23 (1981) 397;
M. Calvetti, The UA1 central detector, Proc. Intern. Conf. on Instrumentation for colliding beam physics (SLAC, Stanford, 1982) (SLAC-250, Stanford, 1982) p. 16;
M. J. Corden et al., Rutherford & Appleton Laboratory Report RL-83-116 (1983);
A. Astbury et al., Rutherford & Appleton Laboratory Report RL-84-025 (1984);
3) S. Cittolin, The UA1 data acquisition system, Proc. Intern. Conf. on Instrumentation for collidingbeam physics (SLAC, Stanford, 1982) (SLAC-250, Stanford, 1982) p. 151;
S. Cittolin, M. Demoulin, W.J. Haynes, W. Jank, J.P. Porte, P. Rossi, V. Vuillemin, The UA1 Data-Acquisition and Online-Monitoring System. To be submitted to Nucl. Instrum. Methods;
4) CERN/SPSC/83-48, SPSC/P92 Aug. 83;
5) P.J. Ponting, A guide to Romulus/Remus data-acquisition systems, (preprint CERN-EP/80-01 1984);

6) S. Cittolin and B. Lofstedt, A Remus crate controller for the autonomous processing of multichannel data streams. Proc. Topical Conf. on the Application of Microprocessors to High-Energy Physics Experiments, CERN 81-07 (CERN, Geneva, 1981), p. 91;
7) L.O. Hertzberger, J.C. Vermeulen, L.G. Wiggers, FAMP: an example of multiprocessing in High Energy Physics. Paper presented at: Three-day in-depth review on the impact of specialized processors in elementary particle physics (Padua 1983), p. 229;
8) J.T. Carrol, S. Cittolin, M. Demoulin, A. Fucci, B. Martin, A. Norton, J.-P Porte, P. Ross and K. M. Storr, Data acquisition using 168/E, Paper presented at: Three-day in-depth review on the impact of specialized processors in elementary particle physics (Padua, 1983), p. 47;
9) S. Cittolin and B.G. Taylor, CAVIAR: Interactive microcomputer control and monitoring of multi-crate CAMAC systems, Proc. Conf. on microprocessors in automation and communications, Univ. of Kent, 1978 (IERE No. 41, London, 1978), p. 309;
S. Cittolin and B.G. Taylor, Super CAVIAR: Memory mapping the general-purpose microcomputer. Proc. Topical Conf. on the Application of Microprocessors to High-Energy Physics Eperiments. CERN 81-07 (CERN, Geneva, 1981), p. 533;
10) S. Centro , STAR system description, (Padua INFN note 1984) PD. 29.5.84 ;
11) P.F. Kunz et al., The 3081/E Processor. Paper presented at this conference;
12) S. Centro, S. Cittolin, M. Demoulin, W.J. Haynes, L.O. Hertzberger, W. Jank, E. Pietarinen, P. Rossi and C. Rubbia, The UA1 VME readout system, UA1 technical note, (CERN UA1 TN-83/103) 1983;
13) The VME/VMX bus CPU module, DSSECPUA1 PHM 1.0 June 1984, Data Sud Systems S.A. 22, rue de CLaret VII 34007 Montpellier France;
14) The VME/VMX bus Dual Port Memory, DSSECDPRX June 1984, Data Sud Systems S.A. 22, rue de Claret VII 34007 Montpellier France;
15) C. Engster and L.G. van Koningsveld. REMUS-VME-VMX branch driver (RVMEX) type V385, CERN-EP 1984 note to be puplished;
16) E. Pietarinen. The VME Crate Interconnect module, UA1 technical note to be published
17) P. Gallno, B. Martin, VME-VMX-CAMAC high speed long distance data link, CERN-DD 1984 note to be published;
18) VME-VMX Parallel Input Output, LAPP 1984 note to be published;
19) A. Arignon, Ph. Farthouat, VME Interrupt Vector Generator, DPhPE-SEIPE CENS 91191 GIF sur Yvette Cedex (6).908.33.86;
20) B. Hallgren, H. Verweij, IEEE Trans. Nucl. Sci. NS-27(1980)33; see also CTD, CERN-EP 279 (1982);

QUESTIONS AND ANSWERS

Q:. How many man-years of effort in software for UA1?

J. Amann

A: 4 people for 6 years: 24 man-years.

Table 1. Number of Channels and Data/Event

Detector	No. Channels	Raw Data (Bytes)	Formatted Data (Bytes)
Central Detector	6200	1600000	≈80000
Hadron Calorimeter	1200	2400	2400
Electromagnetic Cal.	2200	4400	4400
Cal. Position Detector	4000	8000	8000
Forward Chamber	2000	32000	8000
Muon Chamber	6000	≈2000	2000
Iarocci Tube	40000	40000	≈4000

Average Event Size	100 Kb
Parallel Readout and Data Reduction dead time	3÷40 ms
Third Level Trigger Maximum Rate (90% Efficiency)	30 Hz
Event Data Mass Storage Maximum rate	4 Hz

Table 2. Data Acquisition Components

VME Modules	50	Dual Port Memory VME/VMX DPRX 128/256Kb Static RAM.	
	12	VME Crate Interconnect.	
	12	256Kb EPROM.	
	10	512Kb Dynamic RAM.	
	20	Parallel I/O, Interrupt Generator, Graphics	
Readout	200	Camac crates.	Data Digitization
	26	Remus branches.	Parallel Readout
	12	VME crates.	Readout Data Handling
8 bit Processors	200	M6800 µP .	Electronics Control
	110	Signetics 8X300 µP.	Data Reduction/Formatting
	20	Super Caviars.	Equipment Test and Control
16 bit Processors	7	FAMP M68000 10MHz.	Muon Second Level Trigger
	60	VME CPUA1 M68010 8MHz. 256Kb, NS 16081 FP.	Parallel Readout, Data Formatting and Event Data Sampling
32 bit Processors	6	168/E IBM Emulators.	Event Filter and Online Monitor
Main Computers	2	NORD 100/500 2 Mbyte	Data Acquisition and
	6	6250 BPI 125 IPS	Software Development

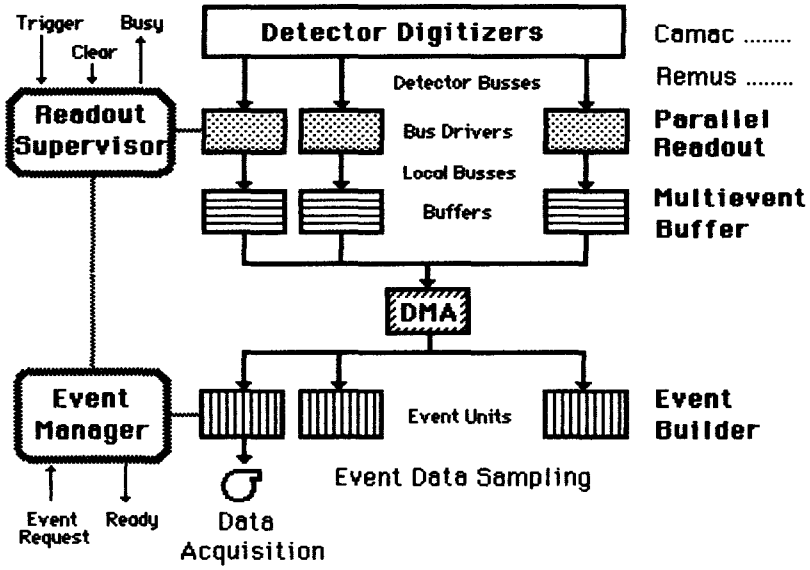


Figure 1. Data Readout Structure

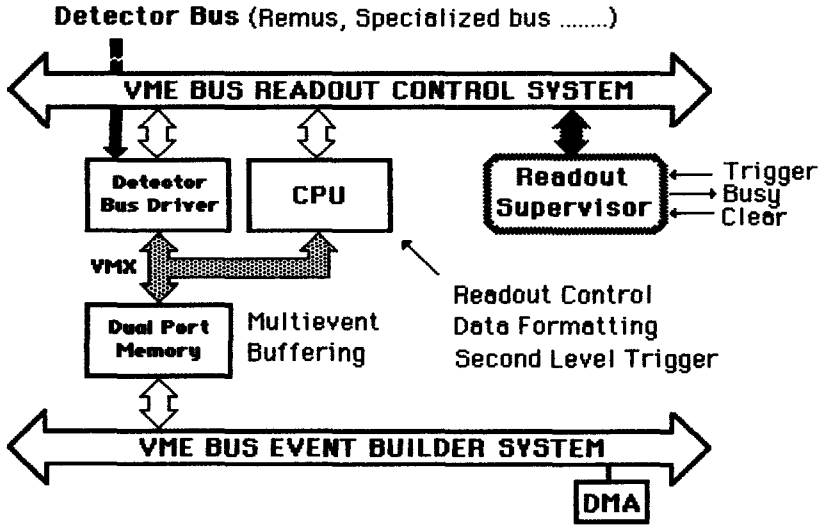


Figure 2. VME Parallel Readout Unit

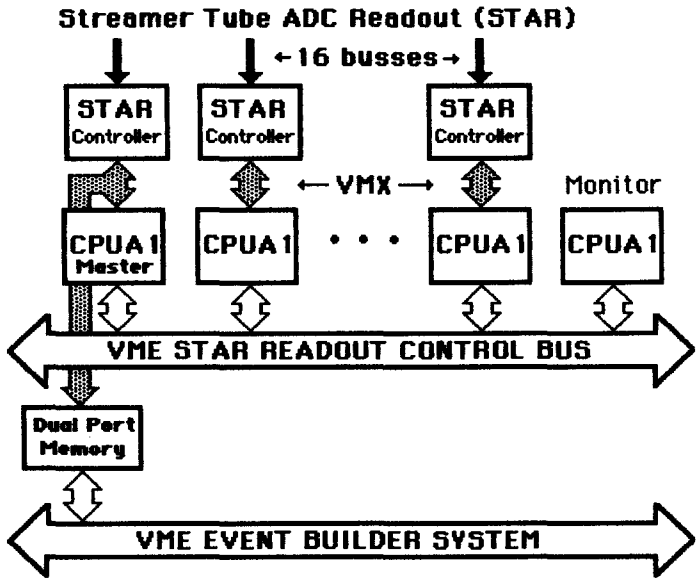


Figure 3. VME STAR Parallel Readout Unit

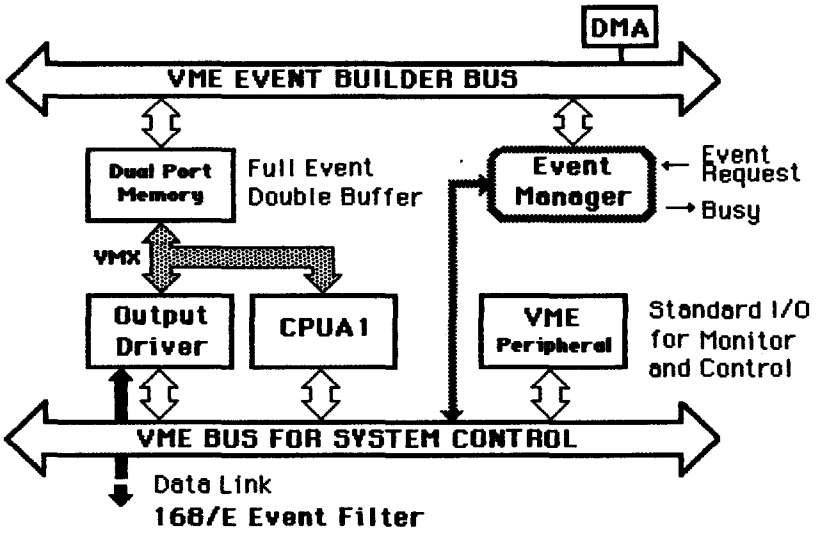


Figure 4. VME Event Builder Unit

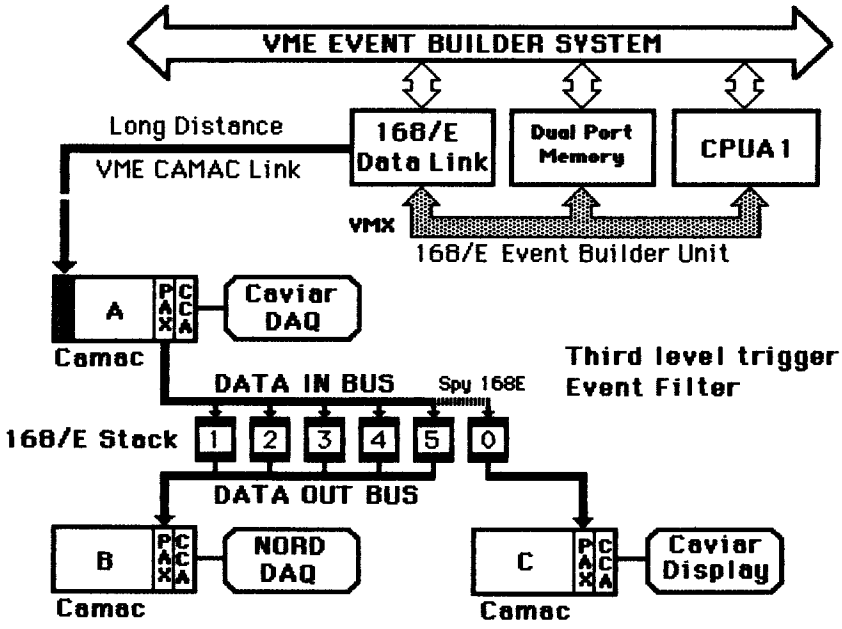


Figure 5. VME-VMX-CAMAC Link. 168/E Event Builder Unit

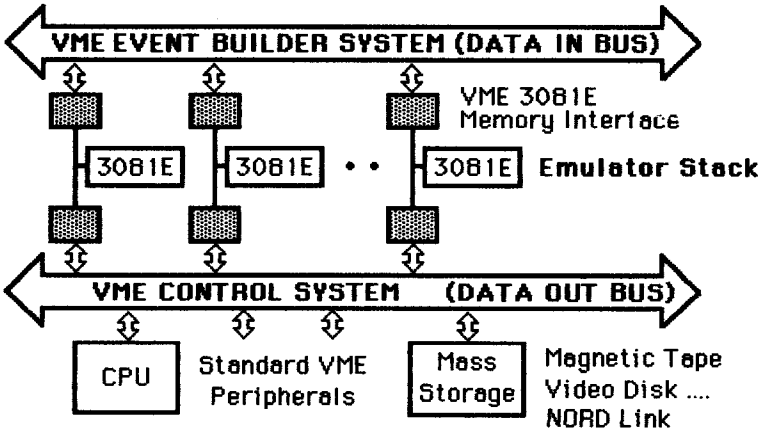


Figure 6. 3081/E VME Event Builder Stack

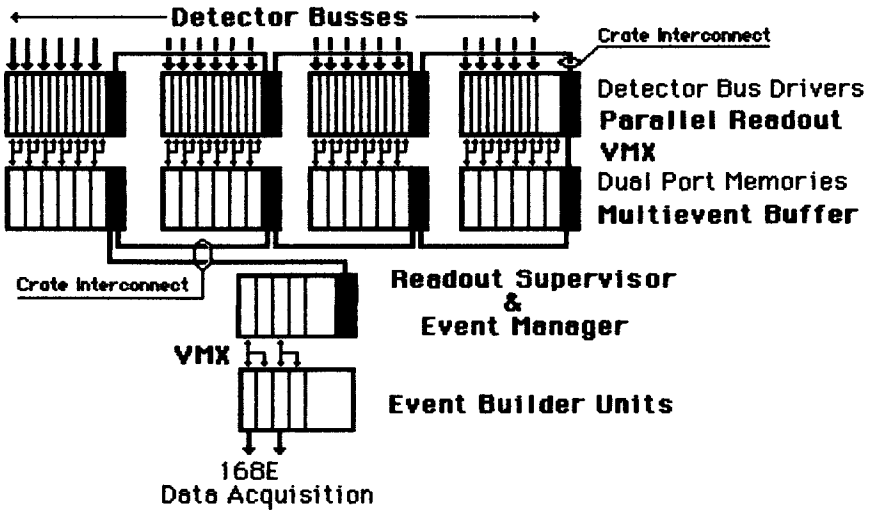


Figure 7. VME Crate Physical Layout

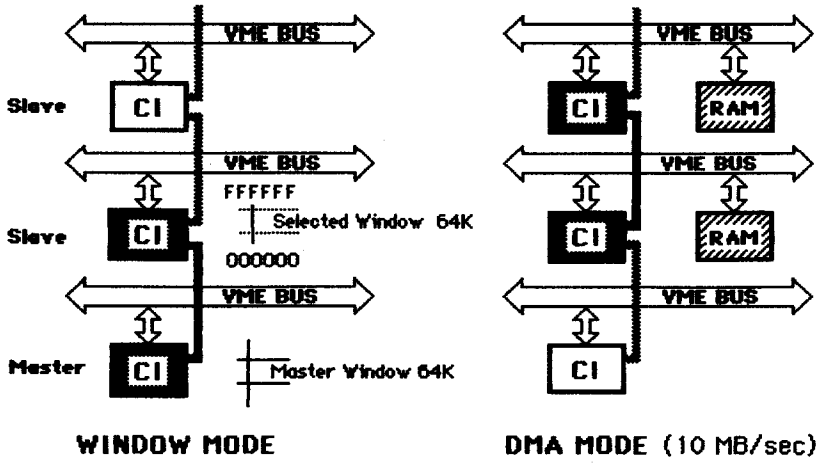


Figure 8. Crate Interconnect

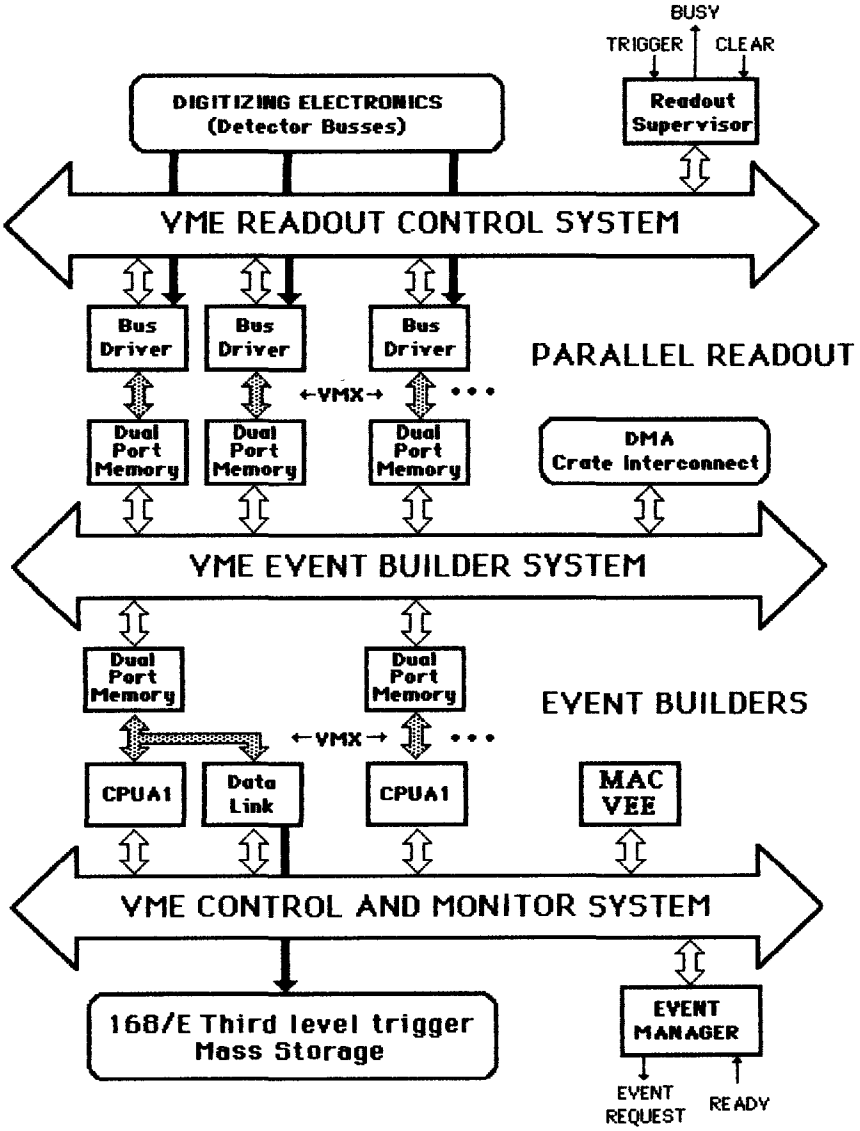


Figure 9. VME Readout Logical Layout


"Happy Trails"
Roy Rogers & Trigger

TRIGGER AND SPECIAL PROCESSORS IN CERN'S SPS AND ISR EXPERIMENTS

C. Verkerk, CERN
Geneva, Switzerland

1. Introduction

This report presents a brief review of the various uses of specialized processors in CERN's SPS and ISR experiments. It does not pretend to be exhaustive, nor does it contain exclusively new information. In fact, some applications of a number of devices were reported before; they are mentioned here again mainly to illustrate the wide variety of processors and of applications. Without counting the - often nameless - special trigger boxes or devices, a dozen or more types of specialized processors or systems are in use at CERN. The present report is arranged by type of processor. It illustrates each time the use of the device in an experiment; for detailed descriptions of the devices themselves the reader is referred to the litterature [1]. In this respect it is interesting to note that the original designs of some of the devices are as many as ten years old! Some have reached now the end of their useful life and their use is not planned for new experiments, others continue to be employed satisfactorily.

This report also contains some indications on future use of processors in experiments at present in the preparation state. On the other hand, the processors used in collider experiments or planned for LEP are excluded, as they are described in other papers presented at this conference.

2. MBNIM

MBNIM [2] is a modular system for building second level triggers, originally developed for the Omega facility. A number of experiments (UA2, EHS, SFM, WA78) now make use of the system and MBNIM has been adopted as a standard at GAMIL in Caen, France. Many of the modules are available from industry and in the course of time a pool of approximately 400 modules has been built up.

The modules are interconnected with a 16-bit flat cable front panel bus which runs at 100 MHz. Typical propagation times for a module are 50-100 ns. External timing can be applied to the modules ("fast mode"), or else a simple handshake can be used, which goes down the whole chain of modules before the acknowledge starts to travel back.

[Photograph opposite courtesy of David Carey, Fermilab, who is a life member of the Roy Rogers Fan Club.]

The range of modules comprises : a 16-bit ALU (10181); a 1K x 16-bit ECL look-up memory (RAHM); a multiplicity logic module (MUSIC) with a 5-bit output and an adder for combining with outputs from other modules; a pattern unit (PATRRO) which contains a MC 6801 microprocessor and a 10 Kbyte pattern table. The latter module will deliver a bit-pattern upon request or compare a pattern with the stored ones. A frequently used module is the bit-assigner, which assigns to every "1" in the input word a 16-bit output word. When more than one bit is set in the input, the outputs are OR-ed together. The module is used to predict from a pattern of hits in a given detector the expected hits in another detector. The comparison of the predicted and the observed pattern can then be done with the ALU. Two more module types (FASTRO and FASTREN) have been added recently for the fast read-out of MWPC of the Omega facility. FASTEN encodes the bits of a single plane and stores the cluster centres in a memory. Automatic scanning of the contents of the memory, in combination with other FASTREN modules, allows looping over the data from several planes, at a rate of 200 ns per combination.

In a recent Omega experiment, the 2nd level trigger required 3 tracks with p_T > 0.9 GeV/c in different quadrants of the downstream detectors. The number of bits in each quadrant was 3-4. The central regions of the chambers had been made insensitive, which resulted in a primary trigger rate of 500 per burst, for 10^6 interactions in the target. The second level trigger was entirely built from MBNIM modules and performed the selection of the transverse momentum :

$$p_T^2 = p^2(\sin^2\phi + \tan^2\lambda) > \text{Threshold}$$

p_T^2 was calculated using :

$$1/p = ay_1 + by_2, \qquad \sin\phi \sim \phi = \alpha y_1 + \beta y_2, \qquad \tan\lambda \sim \lambda = Z,$$

where a, b, α and β are constants, Z is the distance along the beam direction, y_1 and y_2 are coordinates in two different chambers, relative to y_0, the track coordinate at the target position.

This MBNIM system achieved a reduction of the trigger rate by a factor 12-14, in an average time of 30 μs.

3. CAB (Camac Booster)

The Camac Booster (CAB), is a microprogrammed machine, based on the Amd 2900. It was developed at the Ecole Polytechnique, Palaiseau, France, presents itself as a triple width Camac module and is commercially available. It has a 24-bit wide micro-instruction format, 4K memory for program. Data are 16-bit wide, 4K memory is included in the processor, but 3-port extensions of 32K, 200 ns

memory are available. CAB can be configured as a crate controller, or as a branch driver. By virtue of its GPIB interface it is widely used in France for small experiments and for test set-ups together with a hobby computer (Commodore in particular).

At CERN, where some software support is provided, CABs are used in a number of experiments [3] : NA3 (3 processors), R704 and in the LEAR experiments PS177 and PS183. In addition, test set-ups for RICH (Ring Imaging Cherenkov detectors), ALEPH and the CHARM2 experiment are based on CABs.

For instance, in PS177 a CAB does the complete data acquisition and histogram filling. The rôle of the NORD 10 computer is practically restricted to control of magnetic tape recording and, exceptionally, the treatment of complicated events. The experiment PS177 is the first application of a CAB in a multiprocessor configuration. In the CHARM2 experiment two or three CABs will be used to acquire data from thousands of ADCs.

4. ESOP

ESOP is one of the oldest devices in use (its design was completed before bit-slices were available on the market). It is a microprogrammed processor, constructed from MSI (S-TTL) chips. It has a 1K x 48-bit microprogram memory, a 125 ns cycle and up to 64 K x 16-bit data memory.

The European Hybrid Spectrometer used two ESOP processors [4] : one for the read-out of ISIS, a large detector for particle identification, based on the relativistic rise of ionization; and one for the Optical Fiducial Volume Trigger (OFVT). In the latter application, a pre-flash (with a colour outside the sensitivity of the photographic film) is used to produce on an array of photodiodes an image of the growing bubbles in the Rapid Cycling Bubble Chamber (RCBC). This pre-flash takes place about 0.3 ms after the beam traversal. The photodiode array is read-out into an ESOP which detects interacting beam tracks and prepares a decision in about 0.6 ms. In case an interacting beam track is detected, a photograph is taken by triggering the main flash at about 1.5 ms after the beam traversal. The ESOP makes use of data provided by a downstream wire chamber. The optical background of the RCBC varies with time and ESOP builds up a "background memory", to be subtracted from the optical image. At each expansion this background memory is updated.

In the ISR experiment R807 two ESOPs were used [4]. The first detected high p_T particles, using a list of sector pairs in the

barrel shaped drift chambers, which was provided by the second level trigger. This second level trigger was generated with look-up tables, arranged into a rather general purpose device; the Very Fast Bus [1] (VFB). It contained 5 sets of 51 1K x 1-bit ECL memories, providing 50 inputs forming an address pattern. The outputs of 5 memories - one from each set - are ANDed and thus 51 different trigger outputs are available. A satisfied trigger selects one of three data paths : straight to tape, to ESOP1 or to ESOP2. The memories can be loaded via Camac and also the discriminator levels and the assignment of data paths to each trigger are under computer control. Signals at all stages are accessible for checking. Once the data path to ESOP1 is selected, ESOP reads 16 wires/sector and stores the data in a buffer memory (the DTR read-out is destructive). For a luminosity $L = 2 \times 10^{31} cm^{-2} s$, the following rates were obtained [1] :

	rate (s^{-1})	decision time
barrel counters	6.10^5	10 ns
pre-trigger	6.10^4	60 ns
VFB system	1.10^4	700 ns
ESOP	few	∿250 μs

ESOP achieved a rejection of 2000 with a reject time of 120 μs. To accept definitely an event with a particle with $p_T > 5.0$ GeV/c, 370 μs were needed.

The second ESOP in experiment R807 was used to reconstruct electron tracks.

Two ESOPs were used in the past in experiment NA11, and they were carried over into the successor experiment NA32 [4]. NA32 studies decays of $D\bar{D}$ pairs (10^{-13} - 10^{-12}s lifetime), using an active target consisting of silicon diode strips with a pitch of 20 μm (see figure 1). A short distance downstream two silicon strip chambers with a pitch of 400 μm are installed. Then follows a magnetic spectrometer and a calorimeter. ESOP1 performs the read-out of the active target and detects a change of multiplicity of charged particles, all within 0.5 ms. The exact test can be described as follows : with the vertex in plane K, the primary multiplicity is defined as $Mp = \min (C_{K+1}, C_{K+2})$, where C_j is the number of particles in plane j. Similarly the secondary multiplicity in the two downstream chambers is $M_s = \min (C_{17}, C_{18})$. An event is accepted if $M_s - M_p$ > threshold. The data flow for this process in shown in figure 2. ESOP2 in experiment NA32 is used for compressing data from the calorimeter.

As the experiment has started recently, performance figures are not yet available.

5. GESPRO

Another microprogrammed processor of early design and in use in experiment NA10 is GESPRO [5]. It was built at Strasbourg, is based on the Intel 3000 bit-slices, uses 2K x 48-bit microprogram memory with a 150 ns cycle and works on up to 32 Kwords of 24-bit data. It was previously used in experiments WA2 and WA42. Recently, to cover the needs of NA10, a floating point unit was added [5]. The experiment NA10 studies inclusive production of massive muon pairs, and uses 4 GESPROs with floating point units for the event selection. The detectors are arranged in sextants around the beam. A pre-trigger is derived from two hodoscopes and with the help of coincidence matrices the track with the highest p_T is selected from two further hodoscopes. This is done for each sextant and by combining different sextants the highest di-muon mass is selected. All data are read-out with RMH into 1 of 4 multiport memories (see figure 3); a special controller choses the first memory and GESPRO which are free. The memories can also be accessed by Camac. The selected GESPRO checks all di-muon candidates by calculating their invariant mass.

As GESPRO and RMH have simultaneous access to the memories, a reject can be made by GESPRO before the read-out is complete. Approximately 1/15 of the triggers are retained, 60% of those are definitely kept in the off-line analysis at low beam intensity, 40% at high intensity. With the 4 GESPROs working in tandem, triggers are accepted during 98% of the time (with a beam intensity of $10^6\pi$/burst) or 80% of the time (with a beam of $2.10^6\pi$/burst). Figure 4 gives an indication of the decision time per event.

6. MICE

MICE is an emulator of the PDP11, which is 3x faster than the fastest member of the family, the PDP11/70. It executes normal PDP11 instructions on 16-bit data, with an address space of 28K. It has a 1K x 128 bit writeable control store (WCS) with a cycle time of 105 ns. A little over one half of the WCS is coded to emulate the PDP11 instructions, the rest is available for user microprograms, if desired. Single instruction loops can be speeded up a factor two by the use of the REPEAT instruction, which is not part of the PDP11 repertoire. A JUMP to microcode instruction is provided to allow mixing of high-level languages and user-written microcode. MICE can communicate with the external world via a simplified UNIBUS or via DMA interfaces to RMH and Romulus read-out systems. MICE is loaded and controlled from a Camac module. It can be expanded with special hardware units. Seven machines are at present in use in experiments WA1, R704, Omega, WA78, UA6 and in Computer Aided Tomography [1]. In addition, on a number of occasions, MICE replaced a PDP11 in a SC experiment. The preparatory work was entirely done using a real PDP11 and a few hours before data taking started MICE was installed in its place. A factor 4 improvement in data taking rate was obtained.

The first application of MICE was in the neutrino experiment WA1, where cosmic rays are used for calibration of the calorimeters. The problem is to select cosmic ray muons which travel horizontally (λ < 25 mrad). The read-out of the MWPC does not deliver directly wire coordinates, and zeroes must be suppressed. MICE does this zero suppression on the fly, using 630 ns out of the 1.5 μs Romulus cycle. The total read-out time is 1.6 ms and plenty of time is available to transform the valid data into coordinates. The track finding and selection then takes 400-500 μs. The program consists of 350 lines of straightforward PDP11 assembly code and 250 lines of Fortran. After installation of MICE, the total number of calibration events decreased, but in critical regions of the detector 10x more calibration tracks than before became available.

In the Omega facility MICE is use for data buffering and dead-time reduction [6]. The data acquisition computer is a VAX11/780. Events vary in size from experiment to experiment (1000-3000 words/event) and so do the rates (10-1D00 events/burst). MICE and its software handle interrupts forty times faster than the VAX operating system. This fact, together with combining several events in MICE into superevents before sending them to the VAX, results in a considerable increase in throughput (see figure 5). The program occupies 2 Kwords, leaving 26K free for event buffering. Hooks are provided for event filtering in MICE with user supplied Fortran routines. At the time of this symposium, WA77 had recorded $\sim 5.10^6$ events through MICE, at a rather low rate of 100-150 events/burst. From figure 5, it can be seen that the overall performance is 1200 events/burst at 400 words/event or 650 events/burst at 1000 words/event.

Another recent application of MICE is in experiment R704, with a hydrogen jet target in the ISR machine. This application is presented in more detail at this symposium (J-P. Guillaud, ref. 6).

MICE will be used in WA78 which studies hadronproduction of $B\bar{B}$ pairs. Here MICE will buffer the events and also do additional filtering using the software developed for Omega. The trigger will use ECL memory look-up and MBNIM.

7. FAMP

In the fixed target programme, the FAMP 68000-based multiprocessor system was used in experiment NA32. It is however not used there at present. Other uses of FAMP particularly in UA1 are presented at this symposium [7], [10].

8. 168/E Off-line pool

CERN runs an off-line pool of 7 168/Es, with 3 PDP11s and the usual mass-memories, Bermuda triangles, etc. The machines are used by R807, SFM and Asterix [9]. The facility is particularly appreciated because it can provide many CPU hours to a single experiment in a short period of time and at short notice. In the period December 1983 to March 1984 2600 IBM 168 equivalent CPU hours were delivered.

9. 168/E on-line

168/Es built at Saclay are used in experiments NA3 and NA4. In the experiment NA3 [10] events are stored in 4-port memories during the burst. After the pre-trigger (rate 0.5 - 1 x $10^5 s^{-1}$) a second level trigger using data from the special checker board chambers is employed. This special device selects tracks with a large vertical component of p_T and approximates the mass of the di-lepton. It uses FPLAs and look-up tables and brings the rate down to 20-30 s^{-1} with a 200 ns processing time. Together with a few other trigger sources the overall rate is $\sim$ 100 s^{-1}.

The accepted events are then read by 3 CABs into the buffer memory. The CABs do re-formatting, subtract pedestals and do a first reject on simple criteria. The events are sent to MORPION [1], a hard-wired straight-track finder. The results are returned to the memories. The 168/E then performs a complete spatial reconstruction of the events. At present 15 events per burst, of a certain trigger type, are treated in 2-3 seconds total.

The transfers Memory → MORPION → memory → 168/E → memory → PDP11 →tape are controlled by a PDP11/45 at present, but it is planned to run these tasks in a 68000 in future, liberating time on the PDP11 and improving throughput to 50 events/burst. In addition, in NA3, two 68000's are used; one makes histograms for monitoring purposes and the other re-formats MWPC data from $\sim$ 40 000 wires. These data travel again from the memory to the 68000 and back to the memory.

The use of 168/Es in UA1 will be presented in another contribution to this symposium [10]. In addition to these actual uses of 168/Es, UA2 plans the installation of 1 machine and NA31 of 2 processors (see below).

10 FASTBUS

NA 31 is the first fixed target experiment planning to make largely use of FASTBUS [11]. It proposes to measure within a few permille the ratio :

$$\frac{|\eta_{oo}|^2}{|\eta_{+-}|^2} = \frac{\Gamma(K_L \to 2\pi^o)/\Gamma(K_S \to 2\pi^o)}{\Gamma(K_L \to \pi^+\pi^-)/\Gamma(K_S \to \pi^+\pi^-)}$$

Several trigger levels can be distinguished :

trigger	rate/burst	selection
pre-trigger	5.10^4	
trigger	15000	analog thresholds in calorimeter
1st level	12000	analog peak counting
2nd level	1200	digital, AFBI
3rd level	300	$\pi\pi$ mass; vertex; 168/E
4th level	100	refinement by VAX; to tape.

Figure 6 gives an outline of the data-acquisition and trigger system. The AFBI is part of the ORSAY sequencer. The sequencer reads data from the ADC crates at 12 Mbyte/s. The AFBI then calculates, on the fly, partial sums (the x or y are provided by the number of the strip of the calorimeter) :

1. Energy : ΣE_i^J

   where i = 1, ..., 8 (4 times x and y)

   and J = 1, ..., 4 (e.m F and B and hadron F and B)

2. 1st moments : $\Sigma\ (x_i, y_i)\ E_i^J$

3. 2nd moments : $\Sigma\ (x_i, y_i)^2\ E_i^J$

The AFBI then applies a cut on the ratio e.m. calorimeter/rest.

At the second level the data may follow two routes : i) the back-up route, via FASTBUS and the CFI (Camac to Fastbus Interface) to a VAX11/750, at a rate of less than 1000 events/sec, or ii) via FBM (Fastbus Block Mover) to a 168/E at 70 ns per 32-bit word. The 168/E will perform a mass calculation in $\sim$ 10 ms and the accepted events are transferred to a 2 Mbyte memory at 100 events /sec. The transfer is controlled by a 68000. It is expected that 25% of the events will pass the 168/E. The data are then sent in blocks of 64 K to the VAX where further filtering should reduce the number of events by another factor 3.

Beside the AFBI (designed by Drulot, Orsay) a number of other interesting modules - all with internal buffering and self-test facility - are foreseen :

- THU, the Time History Unit, which contains a stack 24-bit wide 256 words deep. Every 20 ns a 24-bit pattern from various detectors is written into it; the trigger marks t = 0. In this way the history from 2.56 μs before the trigger till 2.56 μs after is recorded. At read-out only those patterns which are different from the preceding pattern are read, together with their time-mark. The THU is designed by Passuelo and Galliotto (Pisa).

- Block Mover (designed by Pregernig, CERN).

- CFI (designed by R. McLaren, L. Gustavsson and E.M. Rimmer, CERN) is 68000 controlled; it can be triggered.

- ADCs. 16 channels are routed to 1 ADC, 6 ADCs will be placed on a board, 19 boards in a crate. Designed in Saclay.

- TDCs. Every 4th wire connected to TDC, resolution 5 ns. Designed at CERN (Leferrier).

After the initial system will be completed, NA31 will evolve towards a two-crate FASTBUS system. A FIFO will be added to de-randomize the events for the 16B/Es. It will be 64 Kwords in 4 interleaved banks. Built from standard chips, it should provide overlapped reception and transmission of data at better than 100 ns per 32-bit word.

11 XOP

When speaking of the future, XOP [B] should be mentioned as a possible processor for a number of SPS and LEP experiments. It is an ultrafast and modular successor to ESOP, microprogrammed as its predecessor. The cycle time is 50 ns, the machine consists of several units and the horizontal microinstruction allows to perform different operations simultaneously. A number of shortcomings of ESOP have been remedied. Tests of the first complete XOP should be made this summer; a number of modules have been tested separately.

12 Conclusion and Acknowledgements

At the end of this long list of devices and applications, one is struck by the wide variety of both, and one is led to admire the inventiveness of so many people. In addition to being incomplete in substance, this report is also bound to be imperfect in acknowledging all the contributions to the use of specialized trigger and processing devices at CERN. For the preparation of this talk I received a wealth of information from my colleagues mentioned in the reference section. My sincere thanks go to all of them.

References

1. The majority of the processors or devices mentioned in this report have been already described at the "Topical Conference on the Application of Microprocessors to High-Energy Physics Experiments", Geneva, May 1981. Technical specifications can be found in the Proceedings of this Conference: CERN Report 81-07.

2. T. Armstrong et. al, Using MBNIM electronics for the trigger in a high-energy physics experiment, Nucl. Instr. Methods 175 (1980) 543.

   A. Corre, H. Pflumm and J.P. Wuthrick, A calculated trigger with MBNIM electronics.

   F. Bourgeois: private communication

   W. Beusch: private communication

3. Y. Perrin: private communication

4. H. Anders et al, Implementation and performance of the optical fiducial volume trigger used with the rapid cycling bubble chamber, Nucl. Instr. Methods 215 (1983) 377.

   H. Anders et al, The hardware design of the optical fiducial volume trigger for the rapid cycling bubble chamber, Nucl. Instr. Methods 219 (1983) 66.

   H. de Rijk et al, Application of an ESOP processor in a second generation charm experiment (presented at poster session of "Three Day In-depth Review on the Impact of Specialized Processors in Elementary Particle Physics", Padua, March 23-25, 1983.

   G. Lütjens: private communication

   B. Heck: private communication

5. J. Lecoq et al, A fast arithmetical unit used with microprogrammable processors, Nucl. Instr. Methods, 213 (1983) 329.

   J. Lecoq, Thèse de doctorat d'état, Université de Haute Alsace, 1982.

   J. Lecoq et al, Results on the programmation of the software second level trigger in NA10 experiment at CERN, submitted to Nucl. Instr. Methods.

   J.J. Blaising: private communication

6. J.-P. Dufey: private communication

   M.J. Esten: private communication

   J.-P. Guillaud: these proceedings

7. J. Vermeulen: these proceedings

8. T. Lingjaerde, XOP, a second generation fast processor for on-line use in high energy physics experiments, CERN, Data Handing Division, DD/81/11.

9. A.W. Edwards, N.A. McCubbin and J.-P. Porte, Preparation of off-line programs for the present 168/E and recommendations for the future, CERN, Data Handling Division, DD/83/21.

10. Callot: private communication

    S. Cittolin: these proceedings

11. K. Peach: private communication

    H. Verweij, Fastbus in Europe, status and plans, Proceedings 1983 Nucl. Sci. Symposium.

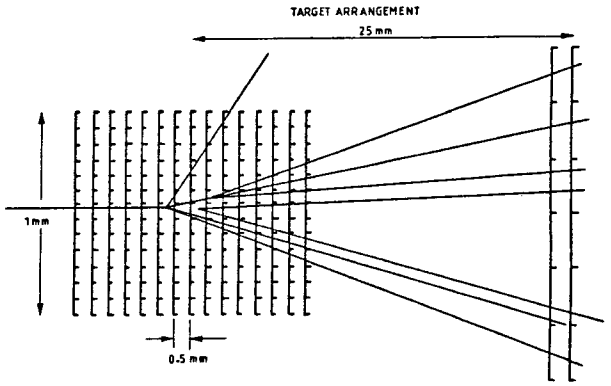


Fig. 1: Arrangement of the active target for experiment NA32

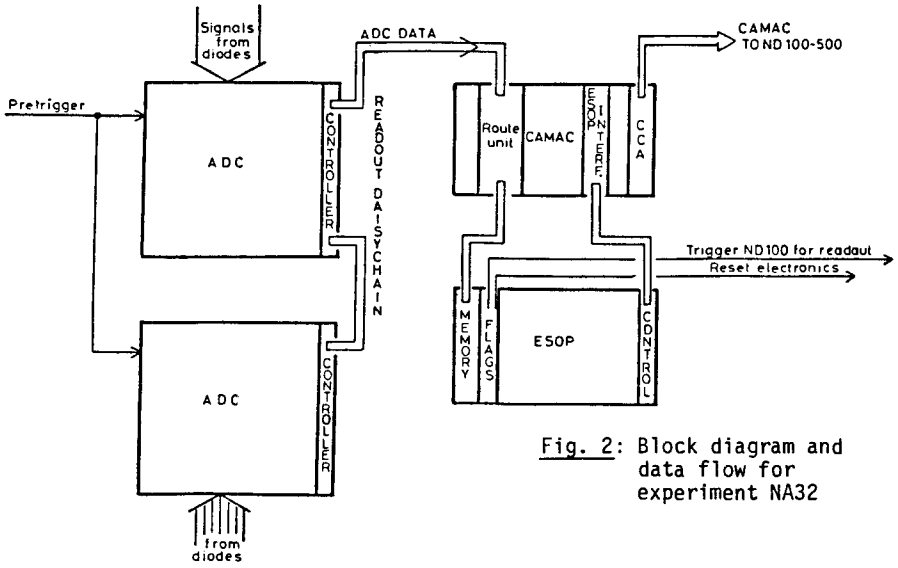


Fig. 2: Block diagram and data flow for experiment NA32

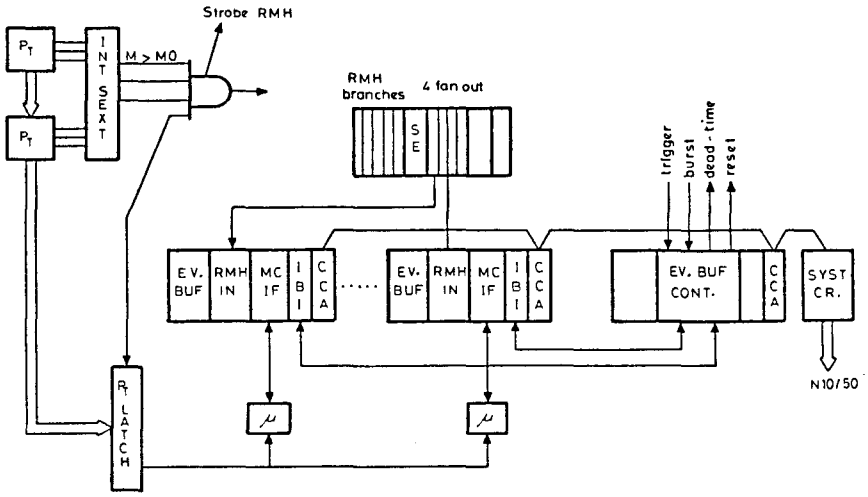


Fig. 3: Block diagram of event selection system for experiment NA10. The blocks marked μ represent GESPROs, of which four are used in total.

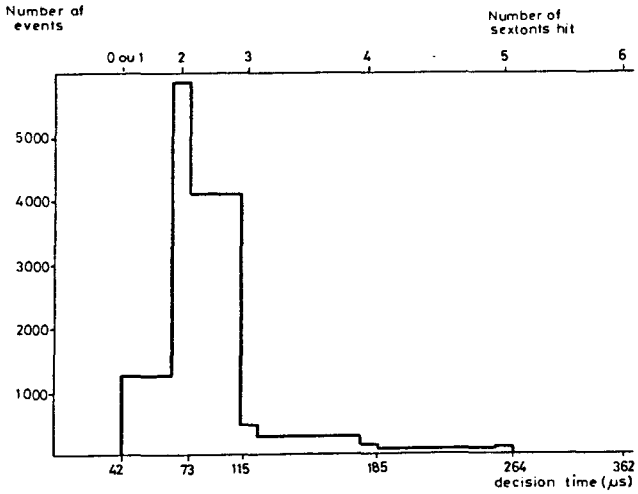


Fig. 4: Histogram of decision times achieved by a GESPRO in experiment NA10. The decision time is directly related to the event topology

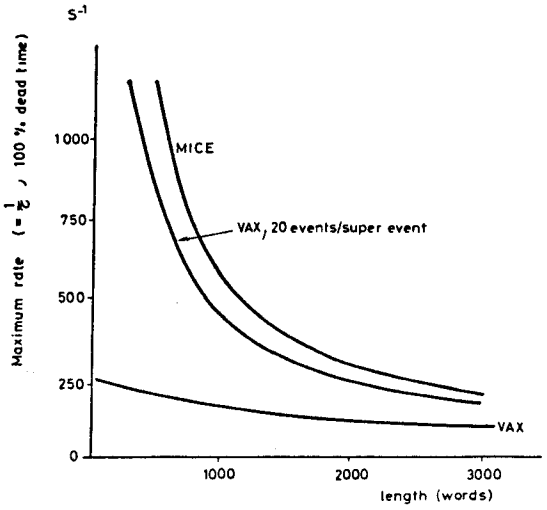


Fig. 5: Performance of Omega data acquisition system, using a MICE emulator

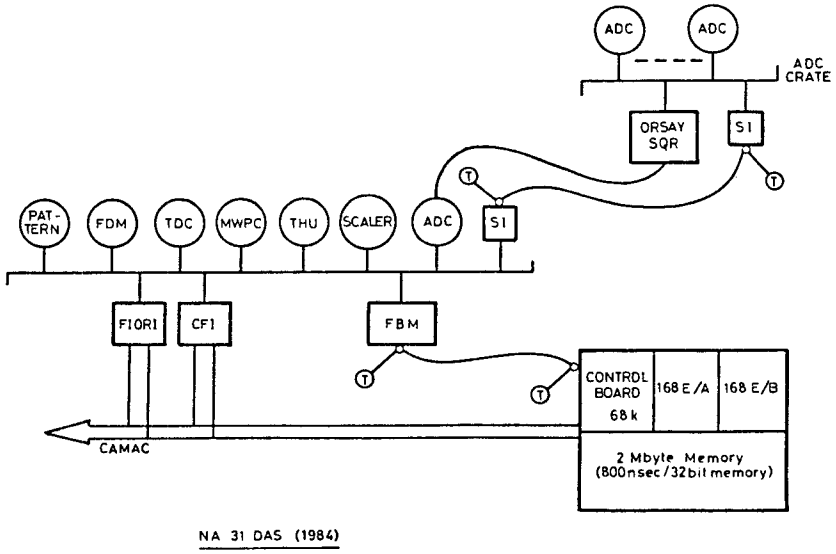


Fig. 6: Schematic of the proposed data acquisition system for experiment NA31, using FASTBUS

QUESTIONS AND ANSWERS

Q: How fast is XOP?

D. Notz

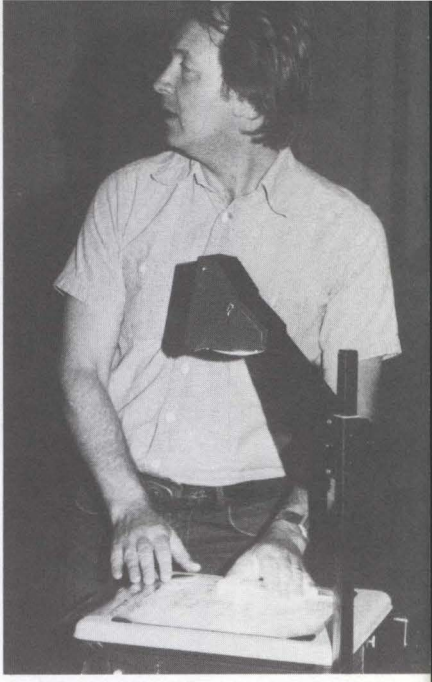
A: XOP should be 4 times faster than ESOP. Introduce nested loops.

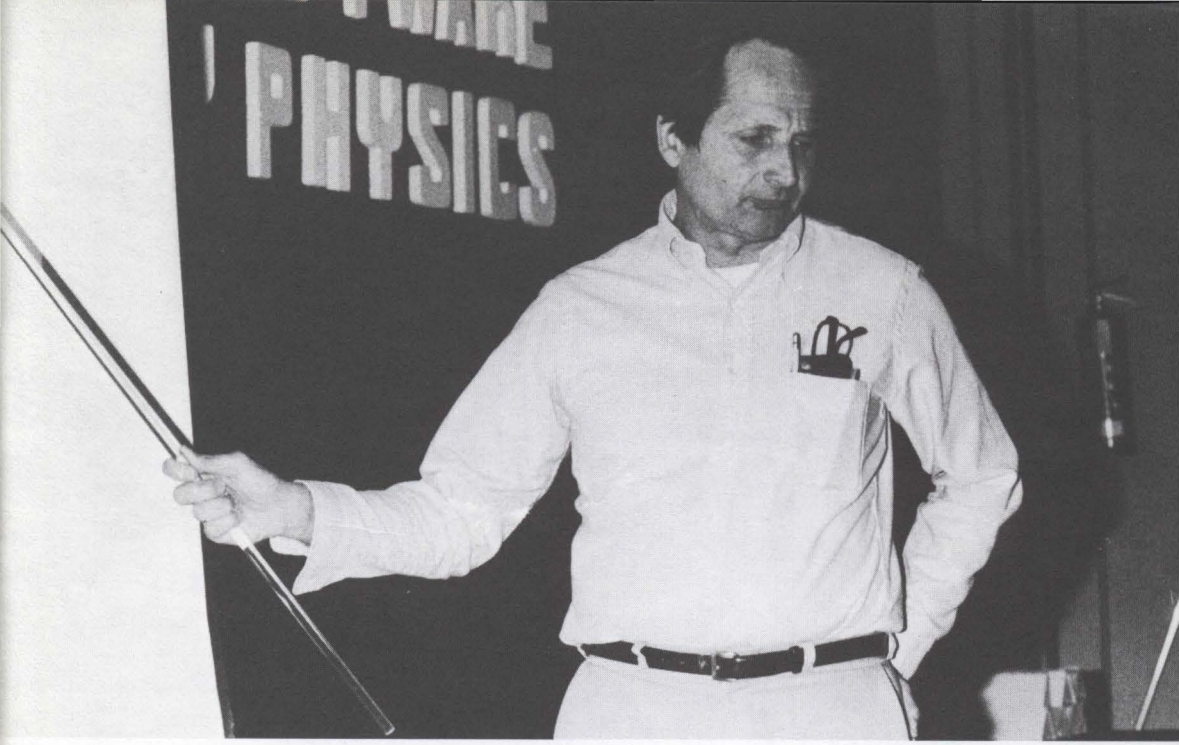
Q: Who is working on the 68000 FORTRAN Compiler and why is a commercial product not used?

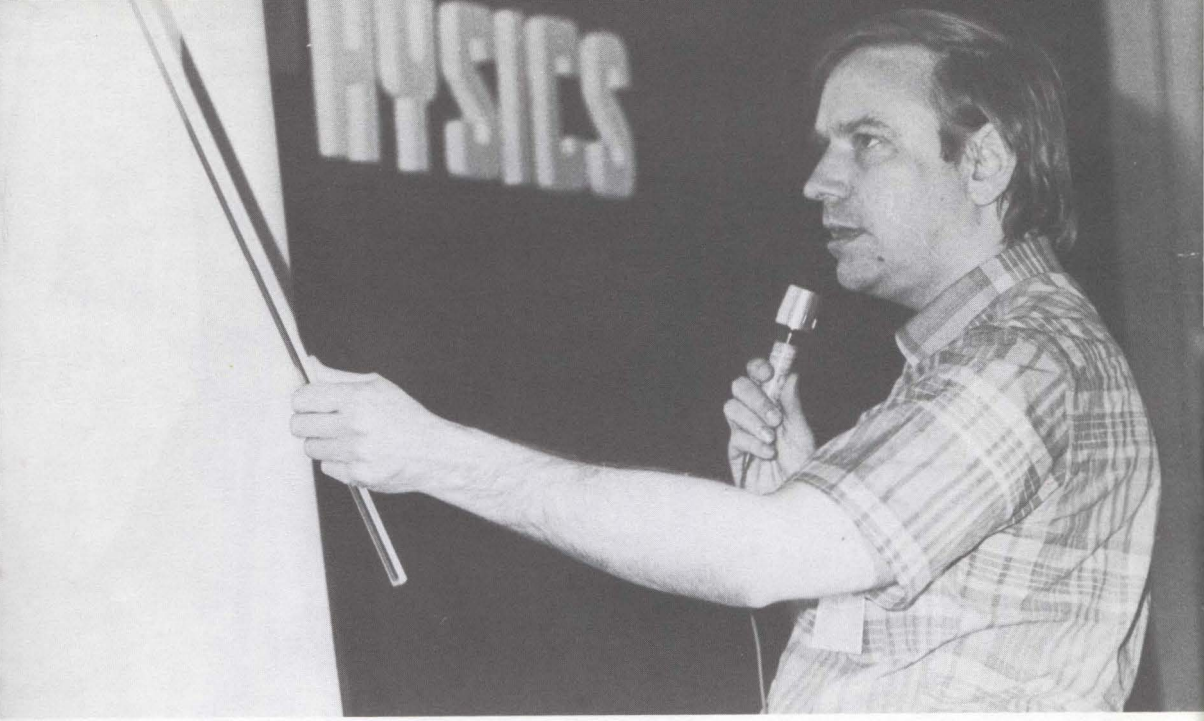
T. Nash

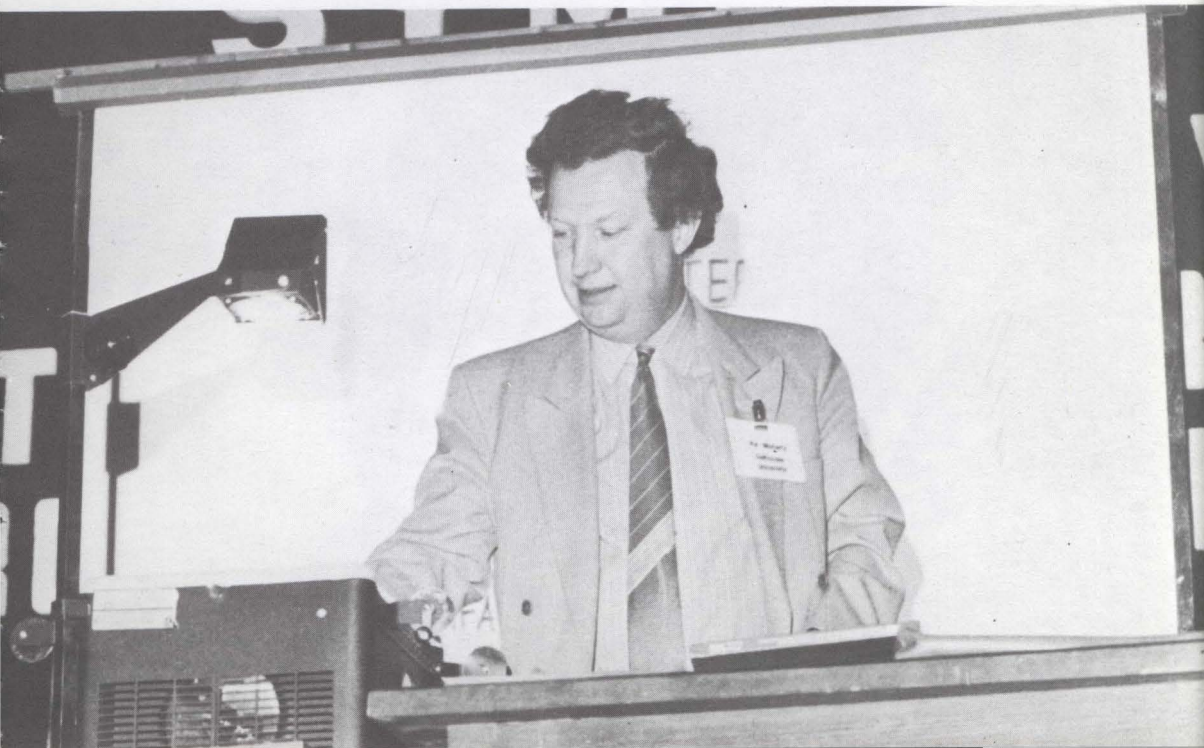
A: Two FORTRAN Compilers are at present evaluated. One has been written by Hans von der Schmitt, using a compiler-compiler. The other is a commercial product from ACE (Amsterdam).


SYMPOSIUM
ON RECENT DEVELOPMENTS IN
COMPUTING PROCESSOR AND SOFTWARE
RESEARCH FOR HIGH-ENERGY PHYSICS
UNIVERSIDAD DE GUANAJUATO



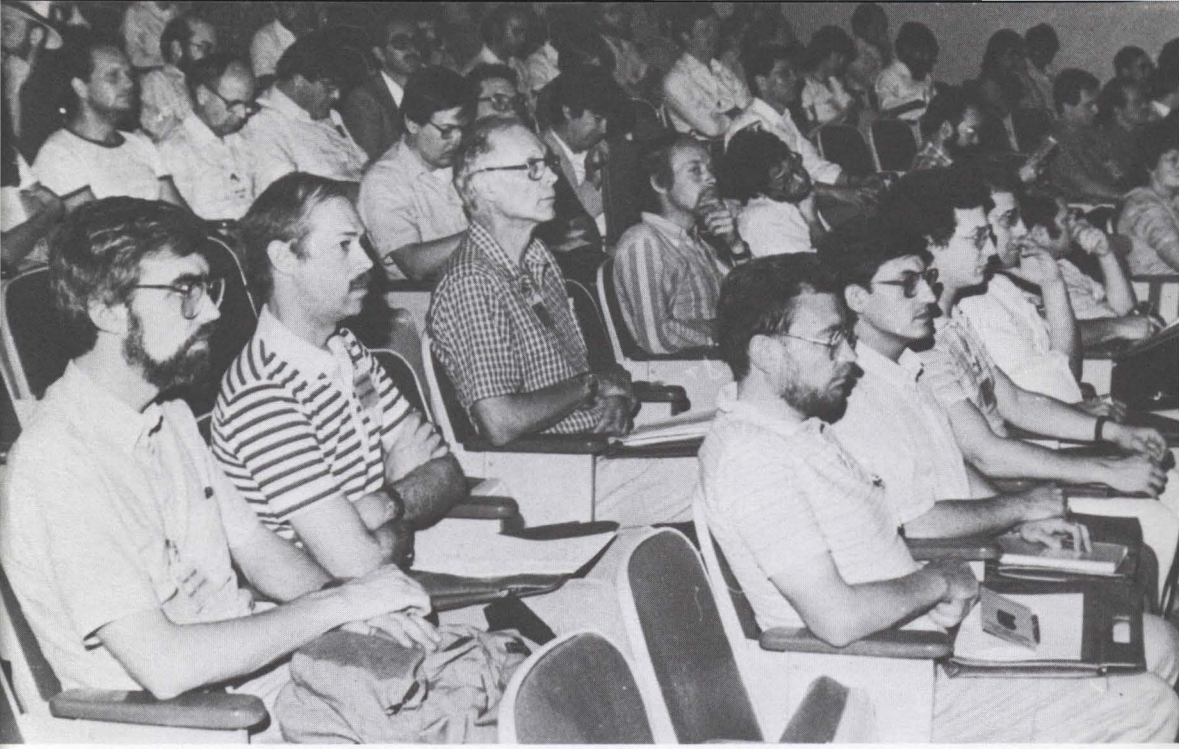

PHYSICS

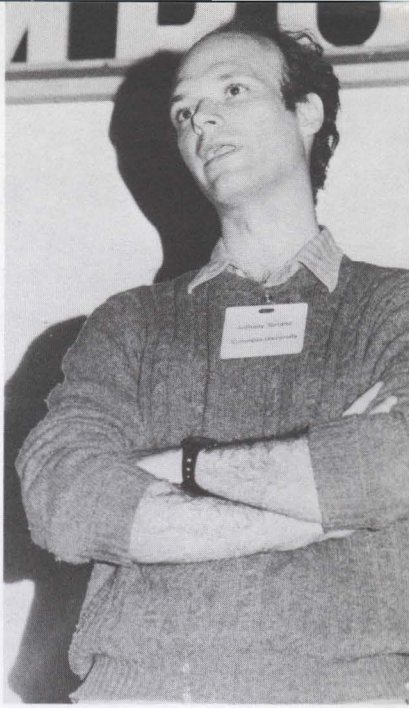

PHYSICS




SOR A
ENE


Coca-Cola






TEATRO JUAREZ

PARTICIPANTS AT THE INTERNATIONAL SYMPOSIUM ON RECENT DEVELOPMENTS IN COMPUTING, PROCESSOR, AND SOFTWARE RESEARCH FOR HIGH-ENERGY PHYSICS

Name	Institution
Alvarez, Antonio	University of Guanajuato, Salamanca
Alvarez, Mario	Univ. Veracruzana
Alverson, G.	Northeastern University
Amann, James	Los Alamos National Laboratory
Amendolia, Salavator	INFN - Pisa, Italy
Austrich, Jordi	UNAM
Avilez, Clicerio	UNAM and Fermilab
Bebek, Christopher	Cornell University
Beck, Frank	Fermilab
Becks, K.-H.	University of Wuppertal
Belmont, Ernesto	ININ
Bensinger, James	Brandeis University
Bintinger, David	UC, Dan Diego
Bongiorno, Vito	Control Data Corporation
Borghini, Michel	CERN
Brandenburg, George	Harvard
Brenner, Al	Fermilab
Brody, Tomas	UNAM
Brower, Richard	UC, Santa Cruz
Bunge, Carlos	UNAM
Campbell, Myron	Enrico Fermi Institute
Cantoral, Eduardo	UAP
Carroll, Terry	Fermilab
Charlesworth, Allan	Floating Point Systems
Cittolin, Sergio	CERN
Cocho, Germinal	UNAM
Conetti, Sergio	McGill University
Correa, Walter	UNAM
Cutts, David	Brown University
Delfino, Manuel	University of Wisconsin
Donaldson, Rene	Fermilab
Escalona, Joaquin	UNAM
Fine, Richard	Columbia University
Fischler, Mark	Fermilab
Franzini, Paolo	Columbia University
Franzini, Juliet-Lee	SUNY, Stony Brook
Gaines, Irwin	Fermilab
Gajski, Daniel	University of Illinois, Urbana
Garcia, Alvaro	University of Guanajuato
Gentit, Francois-Xavier	CEN Saclay
Giannini, Gianrossano	Pisa University
Greenhalgh, John	Princeton University
Guillaud, Jean-Paul	LAPP Annecy le Vieux
Hajjar, Victor	CEN Saclay
Johnson, Marvin	Fermilab
Kaplan, Daniel	Fermilab
Kasha, Henry	Yale University
Kozlowski, Thomas	Los Alamos National Laboratory

Kreisler, Mike	University of Massachusetts, Amherst
Kunz, Paul	SLAC
Lankford, Andrew	SLAC
Lebrun, Paul	Fermilab
Leipuner, Lawrence	Brookhaven National Laboratory
Lepage, Peter	Cornell University
Leu, Peter	University of Hamburg
Levine, Michael	Carnegie Mellon University
Lopez, Armando	University of Guanajuato
Lopez, Pedro Luis	University of Guanajuato
Maples, Creve	Lawrence Berkeley Laboratory
Martinez, Victor	UAM
Maya, Mario	UAP
Medina, Luis Humberto	UNISON
Menocal, Isabel	UNAM
Mertens, Volker	DESY
Miura, Kenichi	Fujitsu Ltd.
Monroy, Jorge	UNAM
Monzon, Eduardo	University of Guanajuato
Morales, Hugo	UNAM
Moreno, Matias	UNAM/CINVESTAV
Moriarty, K. J. M.	Dalhousie University
Morris, Thomas W.	Brookhaven National Laboratory
Mungia, Horacio	
Nash, Tom	Fermilab
Niederer, James	Brookhaven National Laboratory
Notz, Dieter	DESY
Otto, Steve	California Institute of Technology
Perez, Raul	UAP
Peterson, Mike	Digital Equipment Corp.
Polvado, Robert	Northeastern University
Poutissou, Renee	TRIUMF
Quintana, Jaqueline	UAM Iztapalapa
Rincon, Eduardo	UAEM
Rivera, Jose Manuel	UNAM/IPN
Rochester, Leon	SLAC
Rutherfoord, John	University of Washington
Saavedra, Sergio	UNAM
Sala, Silvano	INFN, Milan
Sanchez, Cesar	UAEM
Sanchez, Miguel	UNAM
Sanders, Harold	Enrico Fermi Institute
Schalk, Terry	University of California, Santa Cruz
Schulz, Hans	DESY
Scratchley, Glen	Digital Equipment Corporation
Shambroom, David	Northeastern University
Sheaff, Marleigh	University of Wisconsin
Siskind, Eric	NYCB Real-Time Computing, Inc.
Slaughter, Jean	Yale University
Soriano, Miguel Angel	UAP
Steiner, Kent	ETA Systems, Inc.
Strachman, Zaharia	LPHNE Paris
Terrano, Anthony	Columbia University
Urraca, Manuel	UNAM

Verkerk, Rinus	CERN
Vermeulen, Jos	NIKHEF-H, Amsterdam
von der Schmitt, Hans	Bonn University
von Rueden, Wolfgang	CERN
Votta, Lawrence	AT&T Bell Laboratories
Whyley, Robert	College of William & Mary
Willmann, Robert	Purdue University
Yacaman, Miguel	UNAM
Youngs, William	Digital Equipment Corp.
Zeller, Raymond	Brown University

HONORED GUESTS

Lic. Enrique Velazco Ibarra	Gobernador del Estado de Guanajuato
Dr. Jorge Flores	Subsecretario de Educacion Publica
Dr. Jaime Tacher	CoNaCyT
Lic. Nestor Raul Luna Hernandez	Rector de la Universidad de Guanajuato
Dr. Jaime Martuscelli	Coordinador de Ciencias, UNAM
Dr. Salvador Malo	Director de Investigacion Cientifica y Superacion Academica de la Subsecretaria
Dr. Miguel Jose Yacaman	Director, Instituto de Fisica, UNAM
Dr. Ariel Valladares	Director de Vinculacion de la Subsecretaria de Educacion Publica