CURRENT STATUS OF THE GPU-ACCELERATED ELEGANT *

J.R. King, I.V. Pogorelov†, Tech-X Corporation, Boulder, CO 80303, USA
M. Borland, R. Soliday, Argonne National Laboratory, Argonne, IL 60439, USA
K. Amyx, Sierra Nevada Corporation, Centennial, CO 80112, USA

Abstract

Efficient implementation of general-purpose particle tracking on GPUs can result in significant performance benefits to large-scale tracking simulations and direct (tracking-based) accelerator optimization techniques. This paper is an update on the current status of our work on accelerating Argonne National Lab's particle accelerator simulation code ELEGANT using CUDA-enabled GPUs. We summarize the performance of beamline elements ported to GPU, and discuss optimization techniques for some important collective effects kernels. We also present the latest results of scaling studies with realistic lattices of the GPU-accelerated version of the code.

INTRODUCTION

ELEGANT is an open-source, multi-platform code used for design, simulation, and optimization of FEL driver linacs, ERLs, and storage rings [1, 2]. The parallel version, Pelegant [3, 4], uses MPI for parallelization. Several "direct" methods of simultaneously optimizing the dynamic and momentum aperture of storage ring lattices have recently been developed at Argonne [5]. These new methods typically require various forms of tracking the distribution for over a thousand turns, and so can benefit significantly from faster tracking capabilities.

Graphics processing units (GPUs) offer unparalleled general purpose computing performance, at low cost and at high performance per watt, for large problems with high levels of parallelism. Unlike general purpose processors, which devote significant on-chip resources to command and control, pre-fetching, caching, instruction-level parallelism, and instruction cache parallelism, GPUs devote a much larger amount of silicon to maximizing memory bandwidth and raw floating point computation power.

Our main goals for this project are (1) to port a wide variety of beamline elements to GPUs so that ELEGANT users can take advantage of the high performance that GPUs can provide, (2) support CUDA-MPI hybrid parallelism to leverage existing GPU clusters, and (3) maintain 'silent support' so that GPU-accelerated elements can be used without additional input from the user.

* Work supported by the DOE Office of Science, Office of Basic Energy Sciences grant No. DE-SC0004585, and in part by Tech-X Corporation. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.
† ilya@txcorp.com

BEAMLINE ELEMENT PERFORMANCE

In this section we present a list of the particle beamline elements fully ported to the GPU, and rough estimates of their acceleration compared to the reference CPU code, comparing an NVIDIA Tesla K20c GPU to an Intel Core i7-3770K CPU in simulations with a few million particles.

QUAD and DRIFT: Quadrupole and drift elements, implemented as a transport matrix, up to 3^{rd} and 2^{nd} order, respectively: ~ 100x acceleration, achieving particle data bandwidth of 80 gb/s and over 200 GFLOPS in double precision.

CSBEND: A canonical kick sector dipole magnet with exact Hamiltonian (computationally intensive): Nearly 30x acceleration due to its high arithmetic intensity.

KQUAD, KSEXT, MULT: A canonical kick quadrupole, sextupole, and multipole elements using 4^{th} order symplectic integration: 45x acceleration.

EDRIFT: An exact drift element: Roughly 20x acceleration (purely bandwidth bound).

RCOL: Rectangular collimator: 60x acceleration if particles are removed from simulation.

LSCDRIFT: Longitudinal space charge impedance: ~ 45x acceleration using optimized histogram calculation.

CSRCSBEND: A canonical kick sector dipole with coherent synchrotron radiation: Over 50x acceleration using optimized histogram calculation.

RFCW: RF cavity element, a combination of a first-order matrix RF cavity with exact phase dependence (RFCA), longitudinal wake (WAKE) and transverse wake (TRWAKE) specified as a function of time lag behind the particle, and LSCDRIFT: over 30x acceleration, convolution-based wake elements being the primary bottleneck.

SCRAPER: A one-side collimation element: a 14.5x acceleration with intensive random number generation.

MATTER: A Coulomb-scattering and energy-absorbing element simulating material in the beam path: an acceleration of 23x.

OPTIMIZATION OF COLLECTIVE-EFFECTS KERNELS

In this section we briefly summarize some of our results on optimization of the collective-effects elements. A more detailed discussion can be found in [6].

Histogram Computation

Most collective effects elements in ELEGANT require computing a histogram, which is difficult on a GPU because of the thread contention problem. Thread-safe atomic operations do not provide a practical solution to this problem

ISBN 978-3-95450-168-7

because they lead to an extreme performance degradation. Accordingly, we developed an optimized algorithm illustrated schematically in Fig. 1. The idea is to create as many subhistograms in shared memory as possible while maintaining a high block occupancy, to stride subhistogram access by *nWarpsPerBlock* so as to minimize the costly thread contentions that arise when threads in the same warp try to access the same memory location, and then to combine these subhistograms in shared memory (the latter step incurring but negligible cost). The final step in our histogram computation is a *__threadfence()*-based reduction that combines the results of multiple thread blocks.

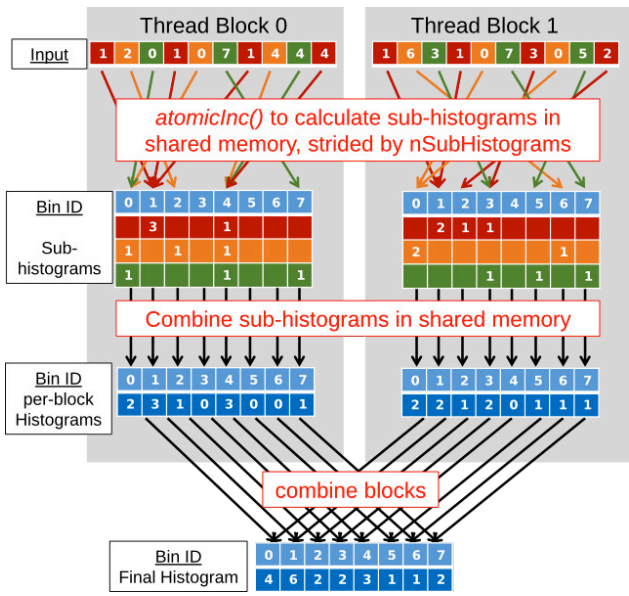

Figure 1: A schematic of the histogram computation algorithm. (See text for details.)

Particle Loss and Sorting

Many beamline elements allow for particle loss. When a particle is lost on the CPU, it is swapped with the particle at the end of the particle array and the particle count is decremented. This algorithm is not amenable to the GPU where concurrent particle-update operations are performed.

Our particle-loss algorithm is illustrated in Fig. 2. A computational kernel does two things to incorporate particle losses: 1) it returns an unsigned integer (zero if the particle is lost and unity otherwise); and 2) it fills a particle-sort index with the particle index plus the number of particles if the particle is lost, and the particle index otherwise. The particle-loss algorithm then performs a sum reduction over the return value. If the result is equal to the number of particles, no particles are lost and the remainder of the loss computation is skipped. If particles are lost, the end of particle array (size of the number of lost particles) is sorted with *Thrust::sort*, and then sort index is converted to unity if the particle is lost, and zero otherwise. An inclusive scan is performed which creates a particle linear index array. When two subsequent elements of this array are different, a particle

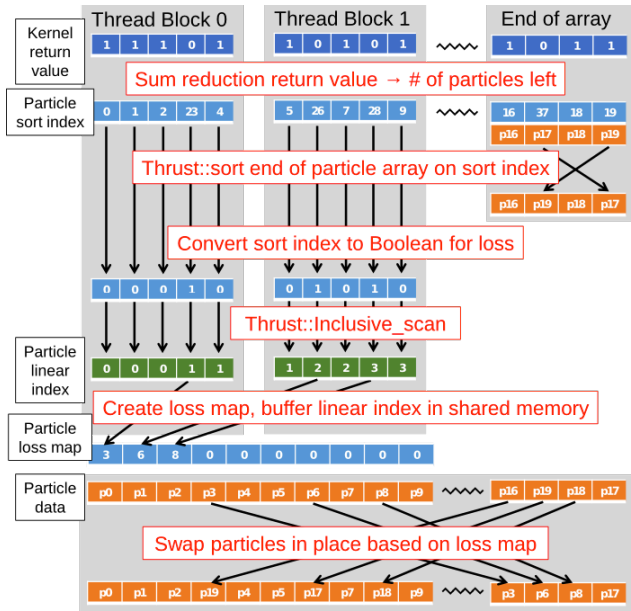

Figure 2: A schematic of the particle-loss sorting algorithm in GPU-accelerated ELEGANT for a simplified case with 20 total and 4 lost particles. (See text for details.)

is lost and the value of the second element in the pair, i, indicates that it is the i^{th} article that is lost. This information is used to produce a contiguous particle loss map which contains indexing information on the lost particles. A final step uses the particle loss map to swap particles to the end of the particle array, and the particle count is decremented by number of lost particles. Although the final two steps of this algorithm contain uncoalesced reads and writes, it is still more efficient than a straightforward sort-by-key algorithm due to the sparsity of operations.

Relative to the sort-by-key algorithm, this optimized algorithm is 4× faster with 0.5% losses, 3× faster with 5% losses, 2.5× faster with 10% losses and roughly equivalent with 50% losses (benchmarking on an NVIDIA Tesla K20c).

DISTRIBUTED-MEMORY SCALING

In this section we present the latest results from performance and scaling studies of the GPU-accelerated ELEGANT relative to the CPU-only version of the code. These studies were performed on the 18,688-node, hybrid-architecture, Titan Cray XK-7 supercomputer at the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory There are two 8-core AMD Opteron CPUs and a single NVIDIA Tesla K20x GPU per Titan node. We use the LCLS beam delivery linac as our test lattice, so that these studies represent end-to-end application performance in a realistic setting, as opposed to the kernel- and function-specific results presented in other sections of this paper.

Results of the weak scaling studies (where the number of cores is increased in proportion to the problem size) are shown in Fig. 3. One can see that most beamline elements exhibit nearly perfect scaling over the explored range of the

ISBN 978-3-95450-168-7

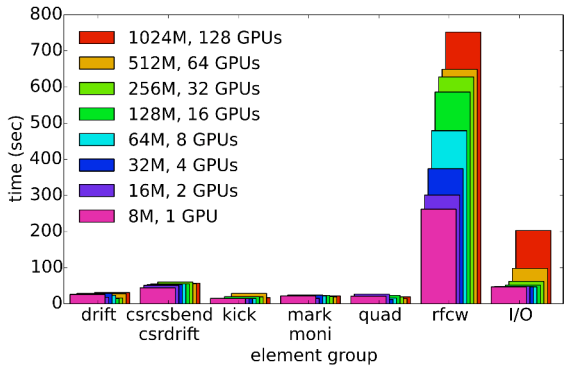

Figure 3: Weak scaling results for the GPU-accelerated version of ELEGANT, arranged by beamline element. LCLS driver linac lattice was used as the test case for this study.

problem sizes (up to about 1 billion macroparticles, thus covering an important-in-practice range of problem sizes). One exception is the RFCW element (a combination of the simple RF cavity element RFCA, longitudinal and transverse wakes, and LSC), which shows worse scaling than other collective-effects elements. We are still working on fully understanding this issue. It should be noted that the weak scaling performance of RFCW is quite good nonetheless: compared to an 8M-particle run on a single Titan node (one GPU), a 128 times larger simulation (1024M particles on 128 nodes) only takes about 2.5 times as long. The same is true for the LCLS driver linac simulation as a whole, which is dominated by the computation of passage through the RF cavities once the parallel I/O capabilities of Pelegant are used to reduce the time spent in reading-in the distribution data. The full LCLS beamline simulation was done within 18 minutes and 20 seconds when using 1×10^9 particles, which is not far below the number of physical electrons in the beam. (The number of macroparticles in simulations with the GPU version of the code can of course be increased beyond the 1B in our scaling studies.)

Figure 4 represents results from a variant of strong scaling, where the number of cores is kept constant but the problem size is increased. For these tests we used a single Titan node (housing 16 CPU cores and a single GPU) and a comparatively small number of particles ranging from 500k to 32M, so as to get a sense of the code performance in small-scale simulations on a workstation as opposed to a large parallel cluster. At 1 million particles, the full application run time is 1 minute on the GPU and 8 minutes 14 seconds on 16 CPU cores (a greater than 8x speedup). At 8 million particles, the full application run time is 7 minutes, 19 seconds on the GPU and one hour and 4 minutes (approximately a 9x speedup). CPU cases with greater than 8 million particles have run times too long for the Titan queueing system which is limited to two hours.

As regards comparing the performance of the GPU-enabled ELEGANT to that of the parallel CPU-only version, one useful metric may be to compare the scaling of the number of cores needed for the two versions of the code

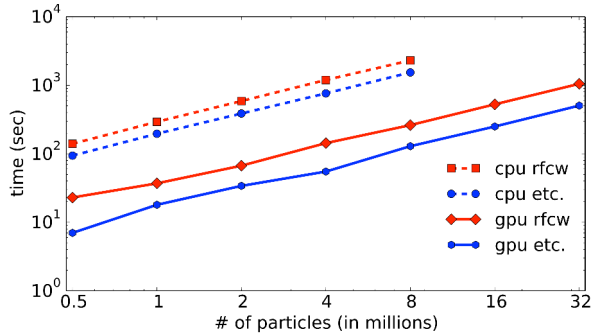

Figure 4: Strong scaling results for the CPU and GPU versions of ELEGANT on a single node of Titan. “etc." refers to the total simulation time minus time spent in RFCW elements.

to achieve approximately the same time to solution, as the problem size varies. (As always, for good performance by the accelerated code, a sufficiently large number of particles is required on the GPU.) We found that, for an 8M-particle simulation of the LCLS lattice, a simulation with the GPU version on 1 Titan node (1 GPU) takes approximately the same time (7 min 19 sec) as a CPU version run utilizing 128 CPU cores (8 nodes), which finished in 8 min 19 sec. Timing on Titan with 128M particles resulted in 16 K20 Kepler GPUs (12:57) falling in between 1024 and 2048 cores (64 and 128 nodes) for the CPU version in terms of time to solution (17:01 and 9:54, respectively). By this metric, a GPU cluster would be a more cost-efficient hardware choice for this type of simulations.

REFERENCES

[1] M. Borland, “elegant: A Flexible SDDS-compliant Code for Accelerator Simulation”, APS LS-287, September 2000

[2] M. Borland, V. Sajaev, H. Shang, R. Soliday, Y. Wang, A. Xiao, W. Guo, “Recent Progress and Plans for the Code ELEGANT,” in Proceedings of ICAP’09, WE3IOpk02 (2009)

[3] Y. Wang, M. Borland. “Implementation and Performance of Parallelized ELEGANT”, in Proceedings of PAC07, THPAN095 (2007)

[4] H. Shang, M. Borland, R. Soliday, Y. Wang, “Parallel SDDS: A Scientific High-Performance I/O Interface,” in Proceedings of ICAP’09, THPsc050 (2009)

[5] M. Borland, V. Sajaev, L. Emery, and A. Xiao, “Direct Methods of Optimization of Storage Ring Dynamic and Momentum Aperture”, in Proceedings of PAC09, TH6PFP062 (2009)

[6] K. Amyx, J.R. King, I.V. Pogorelov, M. Borland, and R. Soliday, “Current Status of the GPU-Accelerated ELEGANT,", in Proc. of IPAC’14, p. 454, MOPME035 (2014)

ISBN 978-3-95450-168-7