21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015) IOP Publishing
Journal of Physics: Conference Series **664** (2015) 092010 doi:10.1088/1742-6596/664/9/092010

The Effect of NUMA Tunings on CPU Performance

Christopher Hollowell[1], Costin Caramarcu[1], William Strecker-Kellogg[1], Antonio Wong[1], Alexandr Zaytsev[1]

[1]Brookhaven National Laboratory, Upton, NY 11973, USA

E-mail: `hollowec@bnl.gov`, `caramarc@bnl.gov`, `willsk@bnl.gov`, `tony@bnl.gov`, `alezayt@bnl.gov`

Abstract. Non-Uniform Memory Access (NUMA) is a memory architecture for symmetric multiprocessing (SMP) systems where each processor is directly connected to separate memory. Indirect access to other CPU's (remote) RAM is still possible, but such requests are slower as they must also pass through that memory's controlling CPU. In concert with a NUMA-aware operating system, the NUMA hardware architecture can help eliminate the memory performance reductions generally seen in SMP systems when multiple processors simultaneously attempt to access memory.

The x86 CPU architecture has supported NUMA for a number of years. Modern operating systems such as Linux support NUMA-aware scheduling, where the OS attempts to schedule a process to the CPU directly attached to the majority of its RAM. In Linux, it is possible to further manually tune the NUMA subsystem using the *numactl* utility. With the release of Red Hat Enterprise Linux (RHEL) 6.3, the *numad* daemon became available in this distribution. This daemon monitors a system's NUMA topology and utilization, and automatically makes adjustments to optimize locality.

As the number of cores in x86 servers continues to grow, efficient NUMA mappings of processes to CPUs/memory will become increasingly important. This paper gives a brief overview of NUMA, and discusses the effects of manual tunings and numad on the performance of the HEPSPEC06 benchmark, and ATLAS software.

1. Introduction

NUMA, Non-Uniform Memory Access, is a memory architecture for multiprocessing computer systems where CPUs are directly attached to their own local RAM [1]. The architecture provides for fast processor access to local memory. Access to remote memory, which is generally local to other processors in the host, is possible, but slower. The x86 NUMA implementations are cache coherent (ccNUMA) and incorporate on-processor memory controllers. AMD introduced NUMA support for x86 with the HyperTransport bus for Opteron in 2003. Intel followed with their own NUMA implementation utilizing the QuickPath Interconnect (QPI) bus for the Nehalem processor in 2007.

In the NUMA architecture, a NUMA node is a grouping of CPU and associated local memory. When one speaks of the NUMA topology of a system, they're referring to the layout of CPUs, memory and NUMA nodes in a host. For x86, there has traditionally been one NUMA node per physical processor package. In this configuration, all cores on the same physical processor belong to a single NUMA node. This has started to change, however, with Intel's recent introduction of Cluster On Die (COD) technology in some Haswell CPUs [2], [3]. With COD enabled, the CPU splits its cores into multiple NUMA domains, which the OS can make use of accordingly.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

In Uniform Memory Access (UMA) systems, typically only one CPU can access system memory at a time. This can lead to significant performance reductions in SMP systems, with more severe degradation occurring as the number of processors in a system is increased. The primary advantage NUMA offers over UMA is that because CPUs have their own local memory, they can effectively access this memory independently of other CPUs in the system.

2. Linux NUMA Support

To optimize memory-processor locality, and thereby take advantage of potential NUMA performance improvements, a NUMA-aware operating should attempt to allocate most/all of a task's memory to one CPU's local RAM. Its process scheduler should also attempt to schedule a task to the CPU directly connected to the majority of that task's memory.

Basic support for a NUMA-aware scheduler first appeared for Linux in kernel 2.5, and evolved over time. The kernel's NUMA support has continued to be enhanced recently in the 3.x series. For the purpose of this study, however, our focus has been on Red Hat Enterprise Linux (RHEL) 6, Scientific Linux (SL) 6, and their kernel, based on 2.6.32, as this is what we're currently running on our processor farm at the RHIC/ATLAS Computing Facility (RACF). By default, the RHEL/SL6 kernel attempts to allocate RAM to the node that issued the allocation request. This, in combination with the kernel's attempts to schedule a process to the the same physical processor, and preferably the same core, in order to improve L1/L2/L3 cache hits [4], often results in process NUMA locality.

While the RHEL/SL6 kernel does have a default memory and CPU scheduling policy which is NUMA-aware, system administrators can fine tune the NUMA scheduling parameters for processes using a number of tools, including *numactl* and *numad*. Developers can also make changes to their own software to modify these parameters, using a number of provided system calls, such as *sched_setaffinity()* and *mbind()*.

numactl has been available in Linux for a number of years. It allows a user to view system NUMA topology, and start a process with specific NUMA node bindings for CPU and/or memory. Some typical examples of *numactl* usage are shown in Figure 1.

numad, on the other hand, became available fairly recently, making its first appearance for RHEL/SL in the 6.3 release. This software is a userspace daemon which monitors NUMA topology, and resource utilization, automatically making affinity adjustments to optimize process locality based on dynamically changing system conditions. *numad* uses the Linux *cpuset* cgroup to set NUMA affinity, and to instruct the kernel to migrate memory between nodes.

By default, x86_64 RHEL/SL6 enables Linux Transparent Huge Pages (THP). With THP enabled, the kernel attempts to allocate 2 MB pages for anonymous memory (i.e process heap and stack), rather than the x86 standard 4 KB, but will fall back to such allocations when necessary. The use of THP reduces process page table size, and thus increases Translation Lookaside Buffer (TLB) cache hits. This increases the performance of logical/virtual to physical address lookups performed by a CPU's memory management unit, and generally overall system performance. As part of the THP subsystem, the kernel introduced *khugepaged*. *khugepaged* is a kernel daemon which periodically scans through system memory, attempting to consolidate small pages into huge pages, and defragment memory.

Early versions of *numad* recommended changing the kernel's *khugepaged* sysfs *scan_sleep_millisecs* settings to 100 ms from the system default of 10,000 ms. This increases the frequency of *khugepaged* memory scans, and therefore small page coalescence and memory defragmentation. This can lead to a better ability for the system to make use of huge pages, and is beneficial when *numad* moves memory between nodes [5]. Newer versions of *numad* automatically set *scan_sleep_millisecs* to 1,000 ms, but this is changeable with the *-H* parameter.

```
Display NUMA Topology:
# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 20 21 22 23 24 25 26 27 28 29
node 0 size: 40920 MB
node 0 free: 23806 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19 30 31 32 33 34 35 36 37 38 39
node 1 size: 40960 MB
node 1 free: 24477 MB
node distances:
node   0   1
0:  10  20
1:  20  10

Force execution on node0 with memory allocation forced on node0
(out of memory condition when all node0 memory exhausted):
# numactl --cpunodebind=0 --membind=0 COMMAND

Force execution on node1 with memory allocation on node1 preferred:
# numactl --cpunodebind=1 --preferred=1 COMMAND
```

Figure 1. Using *numactl* to display NUMA topology, and run a process with forced and preferred specific NUMA node affinity

3. Benchmarks

We utilized two benchmarks to measure the impact of NUMA tunings on CPU/memory performance with a High Energy Physics and Nuclear Physics (HEP/NP) workload: HEPSPEC06 and timing of ATLAS KitValidation software execution. Tests were run under Scientific Linux 6, and were executed multiple times, with averages reported. Figure 2 lists the hardware benchmarked. All benchmarks were run on dual processor Intel Xeon-based systems. It would have been preferable to also benchmark some recent AMD Opteron-based hosts, as well as quad-CPU servers, but unfortunately, as of Spring 2015, we did not have such equipment available at our facility.

HEPSPEC06 is the standard HEP-wide CPU performance benchmark developed by the HEPIX Benchmarking Working Group [6]. It consists of the *all_cpp* benchmark subset of SPEC CPU2006 [7], run in parallel (one process per logical core in our case) to simulate a host fully loaded with processing jobs.

To accurately simulate actual system utilization, Shuwei Ye from Brookhaven National Laboratory had previously developed a benchmarking suite which times software components extracted from ATLAS KitValidation [8]. This includes event generation, Geant4 simulation, digitization, and reconstruction. The suite is currently based on a somewhat older ATLAS software release (16.6.5) for 32-bit SL5. However, the benchmark results are still applicable to the use of newer releases since they share the same basic processing models. In order to reduce any possible effects of the CernVM Filesystem (CVMFS) server and network load on performance, the ATLAS software was copied to local storage, and all job I/O was also performed to local disk. Again, to simulate typical utilization, the benchmark was run in parallel, with one KitValidation process per core spawned and timed. The run-times for event generation and digitization in this benchmark were relatively short on our test hardware: on the order of 1 minute. This didn't give *numad* sufficient time to make affinity adjustments. Therefore for this study, our focus has been on the reconstruction and simulation test results.

Multiple NUMA tunings were tried with each benchmark. The configurations tested included execution with *numactl* forced node binding, execution with *numactl* forced CPU node binding with preferred memory node binding, and finally execution with *numad* enabled with both 15 second and 30 second *numad* scanning intervals. The results were compared to running on a

system with no tunings, and running with a 100 ms *scan_sleep_millisecs* setting alone.

Table 1. Evaluation hardware configuration.

System	Specifications
Penguin Computing Relion 1800i	Dual Intel Xeon E5-2660v2 (Ivy Bridge) CPUs @2.20 GHz 40 logical cores total (HT on) 10 x 8 GB DDR3 1600 MHz DIMMs 80 GB RAM total 4 x 3.5" 7200 RPM 2 TB SATA 6.0 Gbps Drives Configured as a software RAID0 device
Dell PowerEdge R620	Dual Intel Xeon E5-2660 (Sandy Bridge) CPUs @2.20 GHz 32 logical cores total (HT on) 8 x 8 GB DDR3 1600 MHz DIMMs 64 GB RAM total 8 x 2.5" 7200 RPM 2 TB SATA 6.0 Gbps Drives Configured as a software RAID0 device
Dell PowerEdge R410	Dual Intel Xeon X5660 (Westmere) CPUs @2.80 GHz 24 logical cores total (HT on) 6 x 8 GB DDR3 1333 MHz DIMMs 48 GB RAM total 4 x 3.5" 7200 RPM 2 TB SATA 3.0 Gbps Drives Configured as a software RAID0 device

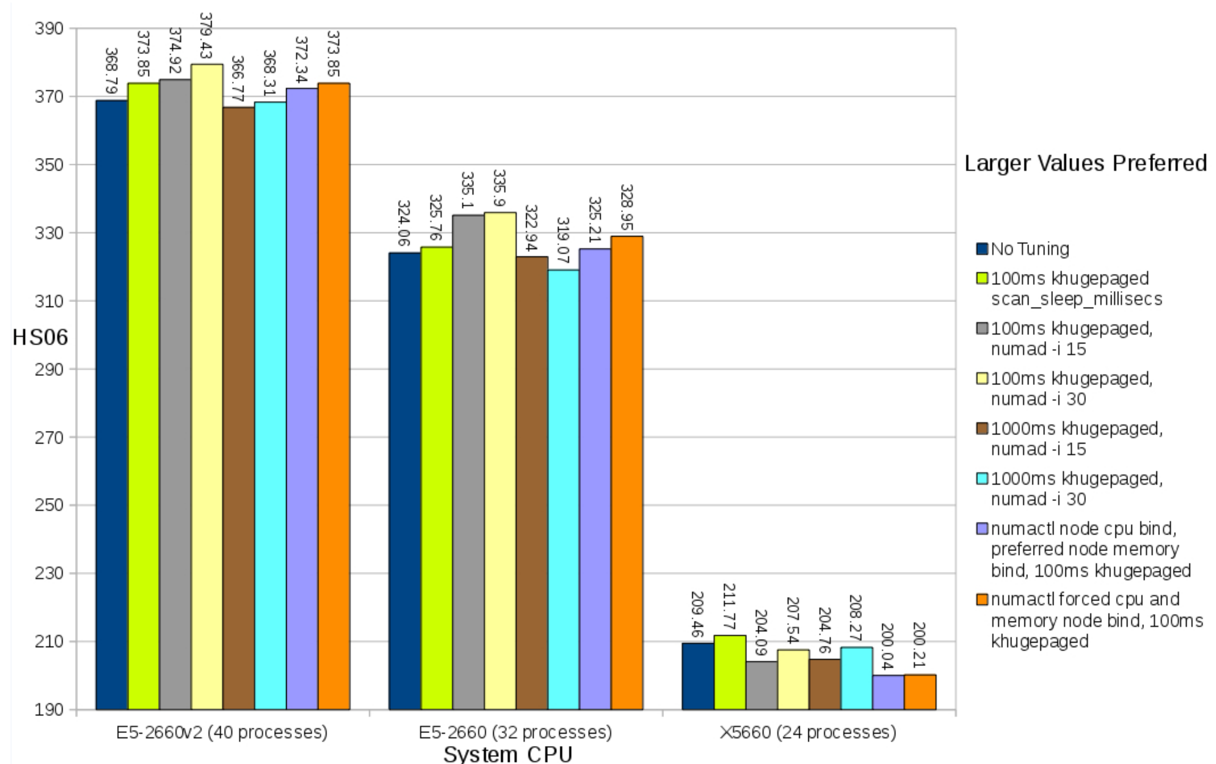

Figure 2. Average HEPSPEC06 results with various *numactl* and *numad* configurations.

4. Results

4.1. HEPSPEC06

Simply decreasing the *khugepaged* scanning period to 100 ms improved HESPEC06 performance by up to 1.4% (standard error 0.3%). Manual NUMA bindings for each HESPEC06 process had little effect on systems with newer Intel CPUs (Sandy Bridge and Ivy Bridge), and lead to a performance reduction on the Dell R410 host with Westmere CPUs and fewer logical cores. Utilizing *numad* with a maximum scanning interval of 30 seconds (*-i 30*) appears to be best, and lead to a performance improvement of 1.5% (standard error 0.4%) for the Ivy Bridge based system, and 3.1% (standard error 1.4%) for the Sandy Bridge host. However, the use of *numad* slightly reduced performance for the Westmere system.

4.2. ATLAS KitValidation Reconstruction

Unlike with HEPSPEC06, the 100 ms *scan_sleep_millisecs* change slightly reduced reconstruction performance, in general. However, as seen with HESPEC06 on newer CPUs, running the benchmark with *numad* management improved performance. With this benchmark, a 15 second *numad* scanning interval was best, and lead to an execution wall clock time reduction of 1.9-4.2% per process (standard error 1.7% or less). In general, slight improvements were seen with manual *numactl* bindings, however, they were not as significant as with *numad*.

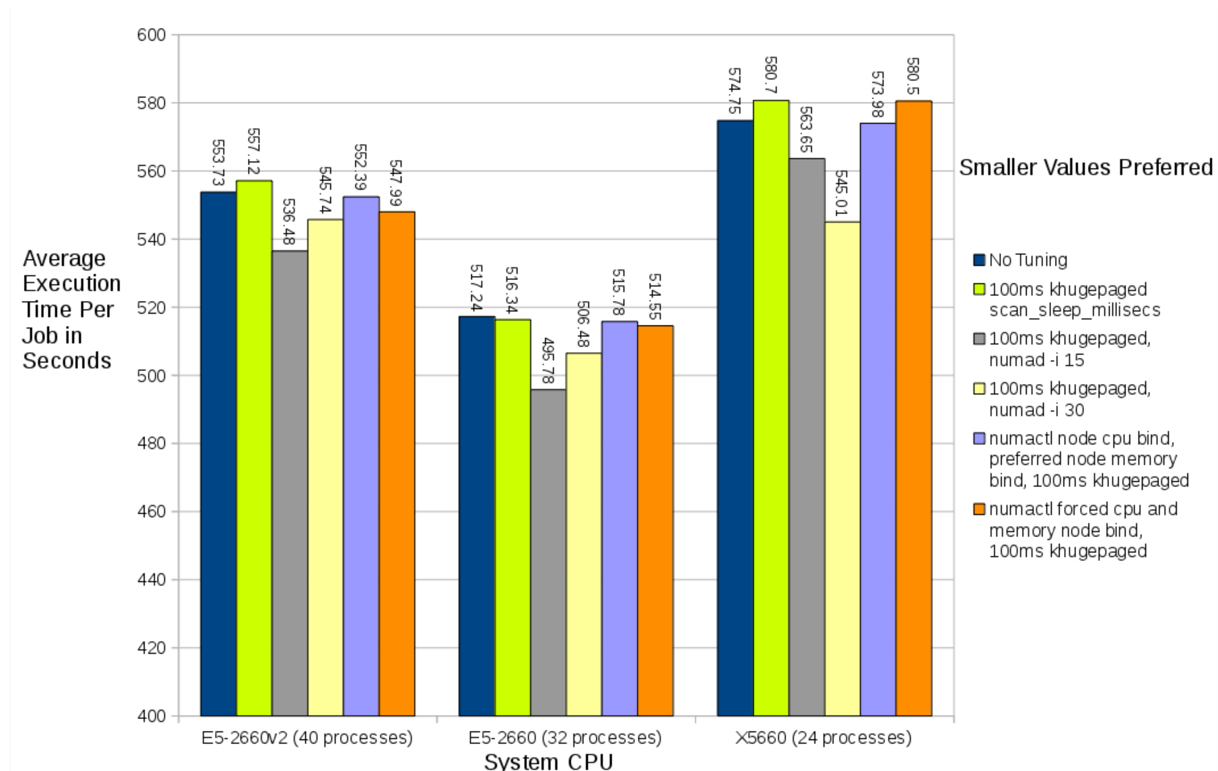

Figure 3. Average ATLAS KitValidation reconstruction software run-times with various *numactl* and *numad* configurations.

4.3. ATLAS KitValidation Simulation

The *khugepaged* scanning period change to 100 ms improved simulation performance by 3.5-4.5% (standard error less than 0.4%). Interestingly, *numad* management reduced simulation

performance by 2.6-4.4% (standard error less than 0.5%) for the 30 second max interval setting, during our tests. There was also no benefit seen with manual *numactl* bindings. Therefore, NUMA affinity tunings were not beneficial for simulation. A possible explanation could be that the majority of the simulation software's memory manipulations fit in processor cache. The mechanism behind *numad*'s performance reduction for simulation is also not completely understood, but we speculate it's related to *numad*'s movement of memory between nodes.

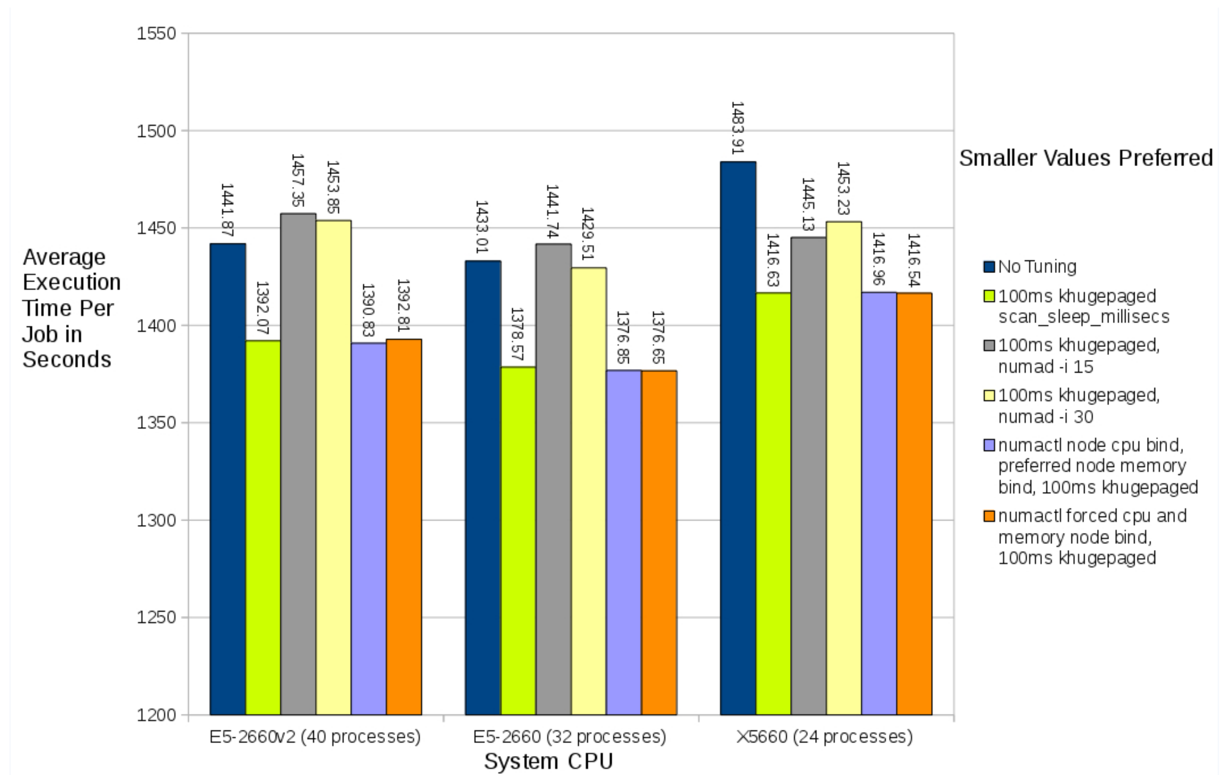

Figure 4. Average ATLAS KitValidation simulation software run-times with various *numactl* and *numad* configurations.

5. Conclusions

Linux NUMA tunings had a positive impact on performance of up to 4.2% for some HEP/NP benchmarks. However, specific tunings were best for different workloads and hardware. Unfortunately, there doesn't appear to be a "one size fits all" optimal configuration.

Overall, simply changing the sysfs *khugepaged* scanning period (*scan_sleep_millisecs*) to 100 ms was beneficial to the majority of HEP/NP workloads tested on dual-CPU Intel Xeon-based systems. Manual static NUMA bindings with *numactl* lead to slight performance gains, or had little/no effect, for all benchmarks on systems with newer CPUs.

Performance gains on the order 2-4% were seen on systems with newer Intel CPUs (with more cores) running *numad*, but only for HEPSPEC06 and reconstruction. Simulation performance was reduced by *numad*, however. Therefore, this daemon should not be run where simulation jobs are executed. However, the use of *numad* would be possible for sites with segregated processor farms where particular queues/job-types are restricted to run on distinct sets of hosts. For sites with mixed job-type processor farms, one can instruct *numad* to ignore simulation jobs by passing their PIDs to the daemon via the *-x* option. This doesn't require a *numad* restart to implement, but would require batch system modification or a job wrapper.

While large performance gains from various NUMA tunings were not seen in this study, it's possible *numad* and *numactl* tunings may lead to more significant performance increases on quad-socket servers, or dual-socket servers with COD Haswell processors. Additional NUMA nodes and cores increase OS scheduling and memory allocation complexity. Therefore, it would be interesting to incorporate such configurations in a future study.

References

[1] Non-uniform memory access: http://en.wikipedia.org/wiki/Non-uniform_memory_access
[2] Intel Xeon E5 Version 3: Up to 18 Haswell EP Cores: http://www.anandtech.com/show/8423/intel-xeon-e5-version-3-up-to-18-haswell-ep-cores-/4
[3] Szostek P and Innocente V 2015 Evaluating the power efficiency and performance of multi-core platforms using HEP workloads *To be published in the CHEP 2015 proceedings, J. Phys.: Conf. Ser.*
[4] Lameter, C 2013 NUMA (Non-Uniform Memory Access): An Overview *ACM Queue: vol. 11, no. 7*
[5] numad: http://www.unix.com/man-page/centos/8/numad/
[6] HEP-SPEC06 Benchmark: http://w3.hepix.org/benchmarks/doku.php/
[7] SPEC CPU 2006: https://www.spec.org/cpu2006/
[8] Ye, Shuwei (BNL): 2015 private conversation on KitValidation benchmarking