The Effect of NUMA Tunings on CPU Performance

Christopher Hollowell <hollowec@bnl.gov>
RHIC/ATLAS Computing Facility
Brookhaven National Laboratory

Co-authors: Costin Caramarcu, William Strecker-Kellogg, Tony Wong, Alexandr Zaytsev

What is NUMA?

NUMA - Non-Uniform Memory Access

Memory architecture for multiprocessing computer systems where processors are directly attached to their own local RAM

Fast access to local memory

CPUs can access each other's (remote) memory, but there is an associated performance penalty

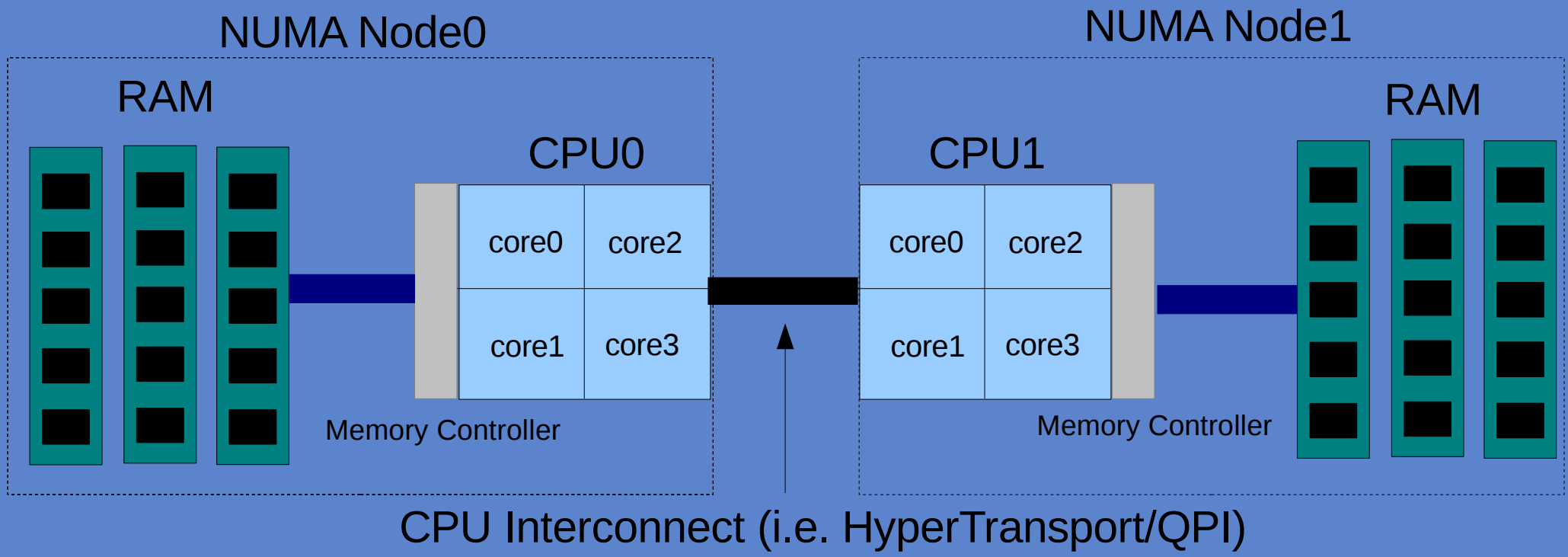
- Such requests are slower because they must flow through the remote memory's controlling CPU

AMD introduced NUMA support for x86 with the HyperTransport bus for Opteron in 2003

- Intel followed with NUMA support in Nehaelm via the QuickPath Interconnect (QPI) bus in 2007

Typical NUMA Architecture

NUMA Node – a grouping of CPU(s) and associated local memory



NUMA Topology – The layout of CPUs, memory and NUMA nodes in a system

How Can NUMA Improve Performance?

In uniform memory access (UMA) SMP systems, generally only one processor can access memory at a time

- CPUs block the memory bus when accessing RAM
- This can lead to significant performance degradation
  - The problem gets worse as the number of CPUs in a system increases

NUMA's advantage – each CPU has it's own local RAM that it can effectively access independently of other CPUs in the system

Requires a NUMA-aware operating system to optimize locality for potential performance improvements

- OS should preferentially:
  1. Attempt to allocate most/all of a task's memory to one CPU's local RAM
  2. Attempt to schedule a task to the CPU directly connected to the majority of that task's memory

Linux NUMA Support

Basic support with NUMA aware scheduler first appeared in kernel 2.5

numactl

- Display system NUMA topology
- Allows for the execution of a process with specific NUMA affinity tunings

```
# 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
```

Linux NUMA Support (Cont.)

- Numad
  - Daemon which monitors NUMA topology and resource utilization, automatically making process affinity adjustments to optimize locality based on dynamically changing system conditions

  - Utilizes cgroups to set CPU/memory affinity, and to migrate memory between nodes. Adjustable scanning intervals (-i 15 default)

  - Added in Red Hat Enterprise (RHEL) and Scientific Linux (SL) 6.3

  - Early versions recommended changing khugepaged sysfs “scan_sleep_millisecs” setting to 100ms on systems with transparent huge (2M) pages enabled (default in RHEL/SL6)
    - Use of huge pages increases TLB cache hits, and therefore general performance
    - Default scan_sleep_millisecs parameter set to 10000ms
    - Change increases defragmentation of memory
    - Newer versions of numad automatically set scan_sleep_millisecs to 1000ms, changeable with -H parameter

Benchmarking NUMA Tunings

All tests run under Scientific Linux (SL) 6

This is the OS currently in use on our compute nodes at RACF

Benchmarks run multiple times on each system, with averages reported

HEPSPEC06 (HS06)

Standard HEP-wide CPU performance benchmark

Developed by the HEPiX Benchmarking Working Group

Subset of SPEC CPU2006 benchmark

Measures performance of a fully utilized system (all cores), simulating a host fully loaded with processing jobs

http://w3.hepix.org/benchmarks/doku.php/

Benchmarking NUMA Tunings (Cont.)

ATLAS Software Benchmarks

- Software from ATLAS KitValidation executed, and timed
  - Event Generation
  - Geant4 Simulation
  - Digitization
  - Reconstruction

- Currently based on an older ATLAS software release (16.6.5) for 32-bit SL5

- Software copied to local storage, out of CVMFS to reduce possible effects of CVMFS server and network load on performance. All I/O performed on local drives

- Run in parallel to exercise all logical cores

- Thanks to Shuwei Ye <yesw@bnl.gov> for developing this benchmark

Hardware Benchmarked

All tests run on dual processor Intel Xeon-based systems

Would like to benchmark recent AMD Opteron based hosts, and quad-CPU systems: unfortunately, we do not currently have such hardware at our facility

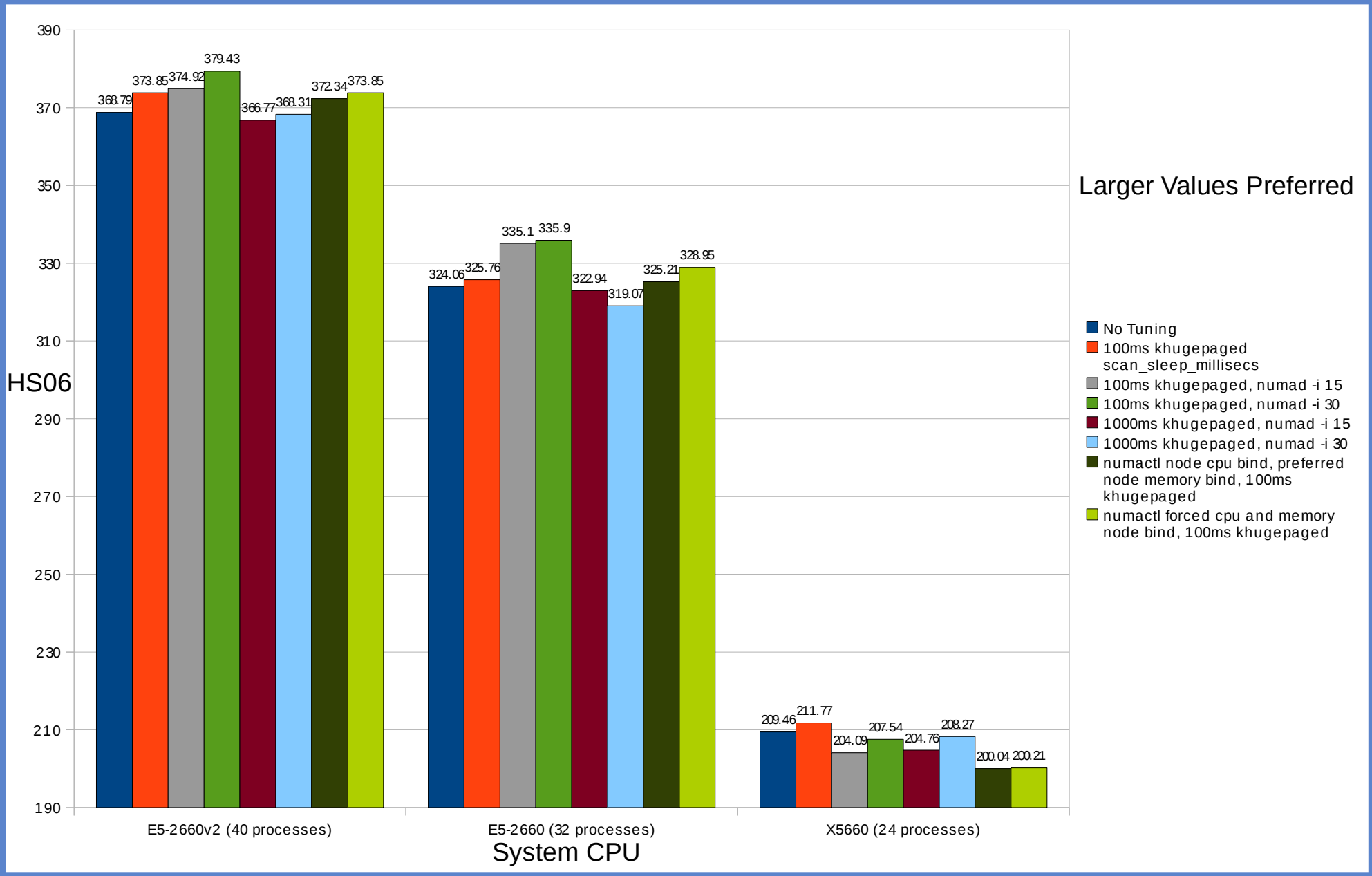
1. Penguin Computing Relion 1800i
   Dual Intel Xeon E5-2660v2 (Ivy Bridge) CPUs @2.20GHz
   40 logical cores total (HT on)
   10 x 8 GB DDR3 1600 MHz DIMMs
   80 GB total RAM
   4 3.5” 7200 RPM 2 TB SATA 6.0 Gbps Drives (Software RAID0)

2. 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 total RAM
   8 2.5” 7200 RPM 500 TB SATA 6.0 Gbps Drives (Software RAID0)

Hardware Benchmarked (Cont.)

3. 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 total
   4 3.5” 7200 RPM 1 TB SATA 3.0 Gbps Drives (Software RAID0)

HEPSPEC06


390
370
350
330
310
290
270
250
230
210
190
HS06
368.79
373.85
374.92
379.43
366.77
368.31
372.34
373.85
324.06
325.76
335.1
335.9
322.94
319.07
325.21
328.95
209.46
211.77
204.09
207.54
204.76
208.27
200.04
200.21
E5-2660v2 (40 processes)
E5-2660 (32 processes)
X5660 (24 processes)
System CPU
Larger Values Preferred
No Tuning
100ms khugepaged scan_sleep_millisecs
100ms khugepaged, numad -i 15
100ms khugepaged, numad -i 30
1000ms khugepaged, numad -i 15
1000ms khugepaged, numad -i 30
numactl node cpu bind, preferred node memory bind, 100ms khugepaged
numactl forced cpu and memory node bind, 100ms khugepaged

HEPSPEC06 (Cont.)

Performance improvement by simply decreasing khugepaged scanning period to 100ms from default 10000ms

Running numad improved HS06 performance for systems with newer Intel CPUs with more cores

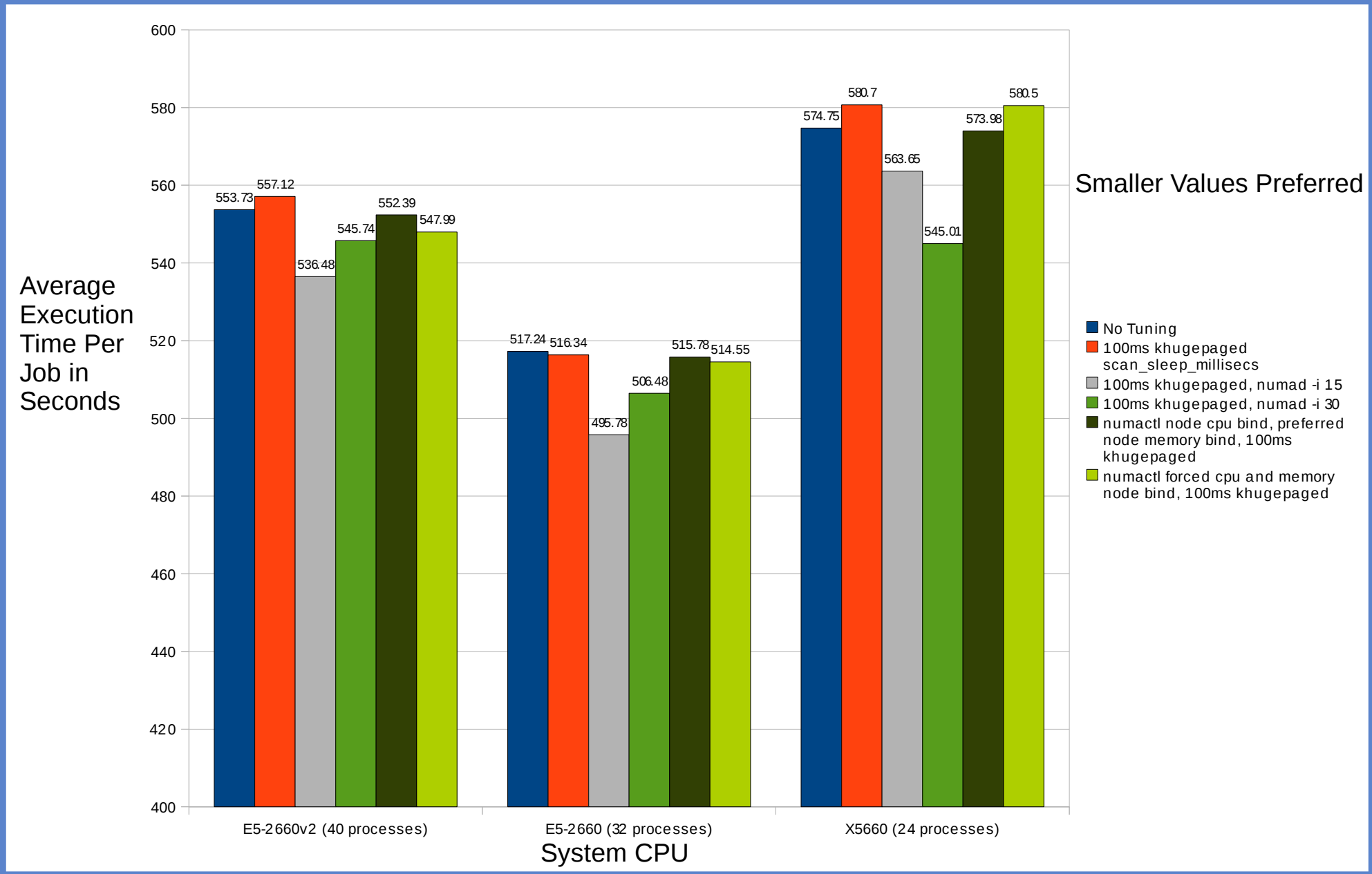
- Max scanning interval of 30 seconds best for these systems
- 1.5% effect for Ivy Bridge and 3.1% for Sandy Bridge

Running numad on Westmere (lower core count) reduced performance

Manual NUMA bindings for each HS06 process had little effect on systems with newer Intel CPUs

- Detrimental on the Westmere host

ATLAS Software in Parallel – Reconstruction



ATLAS Software in Parallel – Reconstruction (Cont.)

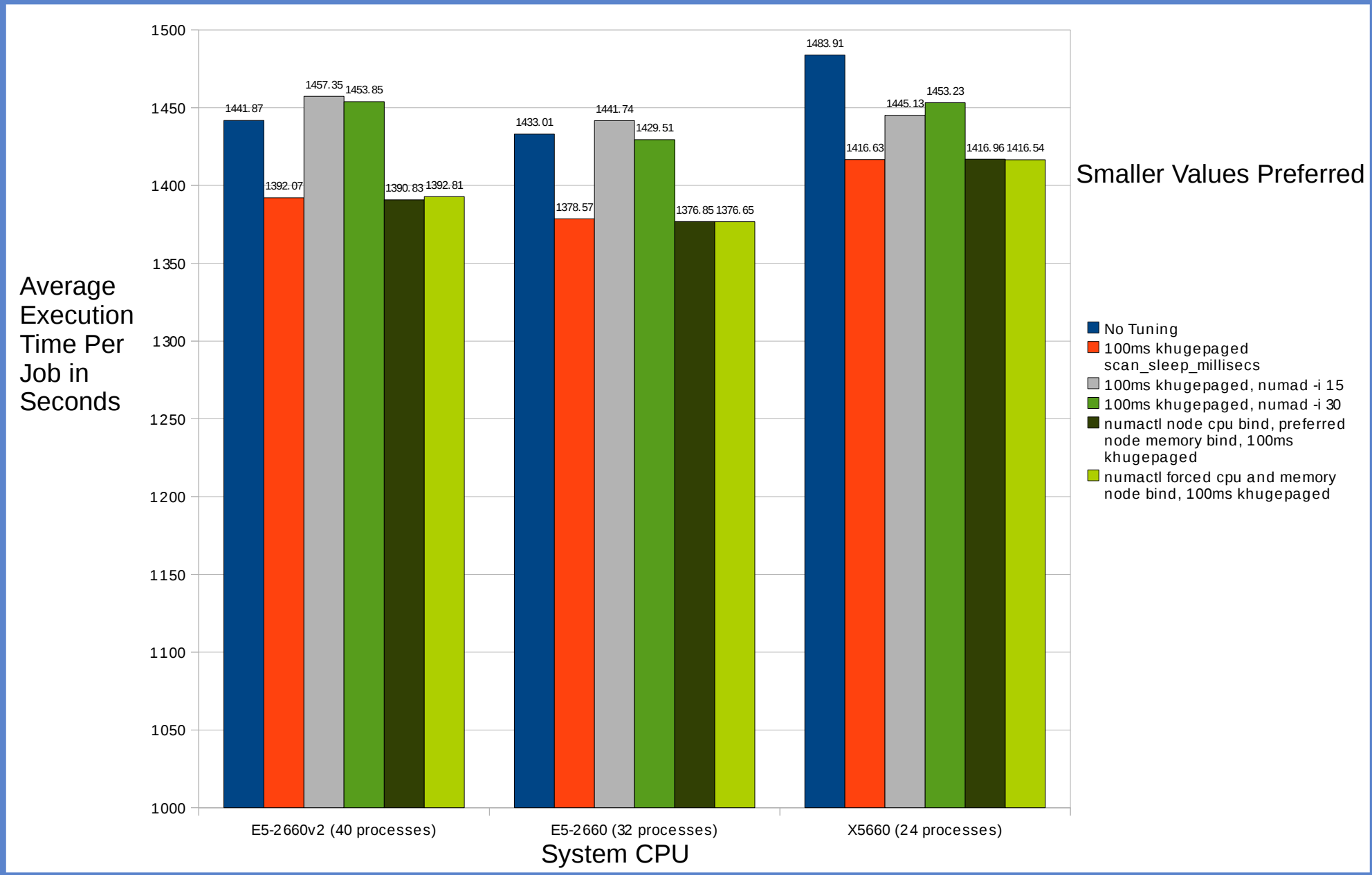
khugepaged scanning period decrease lead to a slight performance increase for Sandy Bridge, and a small performance decrease for Ivy Bridge and Westmere

Numad improved performance on all systems

- Max scanning interval of 15 seconds appears best on systems with newer CPUs
- Reduced execution wall time by 1.9-4.2% per process

Smaller benefit seen from manual bindings with numactl

ATLAS Software in Parallel – Simulation



ATLAS Software in Parallel – Simulation (Cont.)

khugepaged scanning period change to 100ms improved performance by 3.5-4.5%

Numad reduced simulation performance by 2.6-4.4% (30 second max interval setting)

No benefit seen from manual bindings with numactl

Do the majority of simulation data manipulations fit in processor cache?

Conclusions

- Specific NUMA tunings best for different workloads and hardware
  - Unfortunately there doesn't appear to be a “one size fits all” option

- Overall, changing the sysfs khugepaged scanning period (scan_sleep_millisecs) to 100ms was beneficial to typical HEP/NP workloads on dual-CPU Intel Xeon based systems

- Performance gains (~2-4%) were seen on systems with newer Intel CPUs (more cores) running numad, but only for HEPSPEC06 and reconstruction
  - Simulation performance was reduced by numad
  - Recommend starting this daemon on dual-CPU Sandy Bridge or Ivy Bridge systems where simulation jobs are not run
    - Suitable for a processor farm where particular queues/jobs are restricted to run on distinct sets of hosts
    - One can also instruct numad to ignore certain PIDs via the '-x' option: doesn't require daemon restart
      - Would require batch system modification or a job wrapper

Conclusions (Cont.)

Manual static NUMA bindings with numactl lead to slight performance gains, or no effect for all benchmarks on systems with newer CPUs

Numad and numactl tunings may lead to more significant performance increases on quad-socket servers

- Additional NUMA nodes and cores increase OS scheduling and memory allocation complexity