Early Evaluation of the Cray XT5

Patrick Worley, Richard Barrett, Jeffrey Kuehn
Oak Ridge National Laboratory

CUG 2009
May 6, 2009
Omni Hotel at CNN Center
Atlanta, GA

Acknowledgements

- Research sponsored by the Climate Change Research Division of the Office of Biological and Environmental Research, by the Fusion Energy Sciences Program, and by the Office of Mathematical, Information, and Computational Sciences, all in the Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
- This research used resources (Cray XT4 and Cray XT5) of the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Prior CUG System Evaluation Papers

1. CUG 2008: *The Cray XT4 Quad-core : A First Look* (Alam, Barrett, Eisenbach, Fahey, Hartman-Baker, Kuehn, Poole, Sankaran, and Worley)
2. CUG 2007*: Comparison of Cray XT3 and XT4 Scalability* (Worley)
3. CUG 2006*: Evaluation of the Cray XT3 at ORNL: a Status Report* (Alam, Barrett, Fahey, Messer, Mills, Roth, Vetter, and Worley)
4. CUG 2005: *Early Evaluation of the Cray XD1* (Fahey, Alam, Dunigan, Vetter, and Worley)
5. CUG 2005: *Early Evaluation of the Cray XT3 at ORNL* (Vetter, Alam, Dunigan, Fahey, Roth, and Worley)
6. CUG 2004: *ORNL Cray X1 Evaluation Status Report* (Agarwal, et al)
7. CUG 2003: *Early Evaluation of the Cray X1 at ORNL* (Worley and Dunigan)

(and subsystem or application-specific views of system performance)

8. CUG 2006: *Performance of the Community Atmosphere Model on the Cray X1E and XT3* (Worley)
9. CUG 2005: *Comparative Analysis of Interprocess Communication on the X1, XD1, and XT3* (Worley, Alam, Dunigan, Fahey, Vetter)
10. CUG 2004: *The Performance Evolution of the Parallel Ocean Program on the Cray X1* (Worley and Levesque)

What is an Early Evaluation?

1. A complete evaluation:
   a. microkernel, kernel, application benchmarks, chosen to examine major subsystems and to be representative of anticipated workload
   b. optimized with respect to obvious compiler flags, system environment variables (esp. MPI) and configuration options
2. performed quickly (*in time for CUG* ☺)
   a. not exhaustive (can't answer all questions nor examine all options)
   b. minimal code modifications
3. with a goal of determining:
   a. performance promise (a lower bound)
   b. performance characteristics (good and bad)
   c. usage advice for users
4. in the context of an "evolving" system, subject to:
   a. HW instability
   b. system software upgrades

Target System

Cray XT5 at ORNL (*JaguarPF*)

- 18,722 compute nodes, 8 processor cores per node, 2 GB memory per core:
  - 149,776 processor cores and 299,552 TB memory
- Compute node contains two 2.3 GHz quad-core Opteron processors (AMD 2356 "Barcelona") linked with dual HyperTransport connections and DDR2-800 NUMA memory
- 3D Torus (25x32x24) with Cray SeaStar2+ NIC (9.6 GB/s peak bidirectional BW in each of 6 directions; 6 GB/s sustained)
- Version 2.1 of the Cray Linux Environment (CLE) operating system (as of February 2009)

Compared with Cray XT4 at ORNL (*Jaguar*)

- 7832 compute nodes, 4 processor cores per node: 31,328 processor cores
- Compute node contains one 2.1 GHz quad-core "Budapest" Opteron and DDR2-800 UMA memory

Initial Evaluation Questions

- Performance impacts of changes in
  - Processor architecture (2.3 GHz Barcelona vs. 2.1 GHz Budapest)
  - Node architecture (2 socket and NUMA vs. 1 socket and UMA):
    - Additional memory contention? Utility of large page support?
    - OpenMP performance (8-way maximum vs. 4-way maximum)
    - MPI communication performance
      - Intranode and Internode
      - Point-to-point
      - Collective
- Performance characteristics of running at increased scale
- Nature and impacts of performance variability
- Application performance

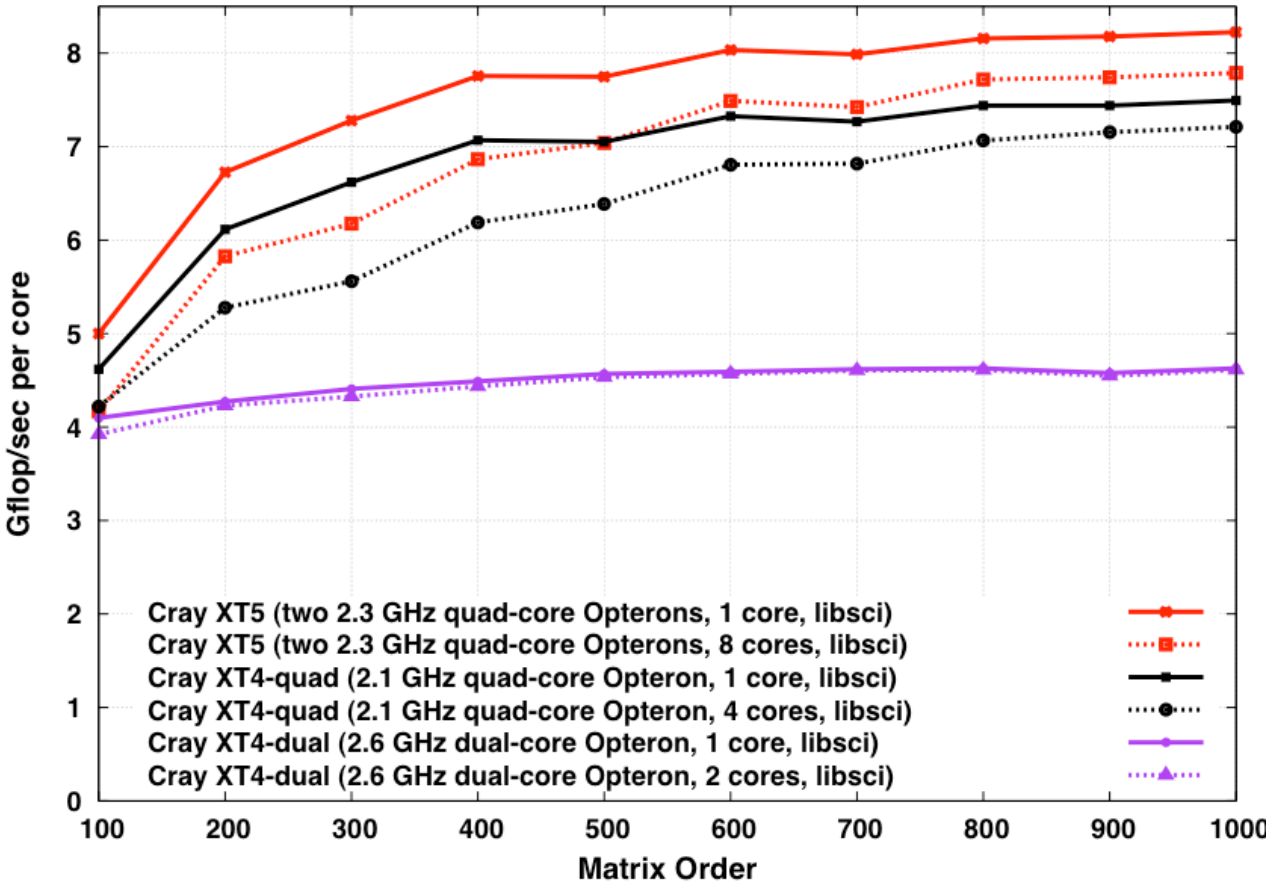
Status of Evaluation

- Evaluation far from complete:
  - System not open to general evaluation studies, only studies in support of early science application codes.
  - Scaling application codes to 150,000 cores is requiring re-examination of algorithms and implementations.
  - No full system scaling studies as of yet because of high cost and special requirements (e.g. interactive session).
  - Performance variability makes aspects of evaluation difficult:
    - There appear to be multiple sources of variability, some that may be eliminated easily, once diagnosed properly, and some that may be intrinsic to the system.
    - May be possible to mitigate impact of intrinsic variability once it has been diagnosed adequately.
- Too much data to present in 30 minute talk. Will describe highlights of preliminary results.

Talk Outline

1. Single node performance
   a. Kernels: DGEMM, FFT, RandomAccess, STREAM
   b. Application codes: POP, CAM
2. MPI communication performance
   a. Point-to-point: Intra- and Inter-node
   b. Collective: Barrier, Allreduce
   c. HALO
3. Application codes: approaches, performance, and progress
   a. AORSA
   b. XGC1
   c. CAM

Matrix Multiply Benchmark (DGEMM)



Evaluated performance of `libsci` routine for matrix multiply. Achieved 89% of peak, Some degradation observed from running benchmark on all cores simultaneously. Behavior similar to that on XT4-quad, scaled by difference in clocks.

Other HPCC Single Node Benchmarks (ratio of JaguarPF to Jaguar)

	Core Performance (1core active)	Core Performance (all active)	Socket Performance (all active)	Node Performance (all active)
FFT	1.074	1.134	1.134	2.267
RandomAccess	1.094	1.139	1.139	2.277
STREAM	0.998	0.937	0.937	1.874

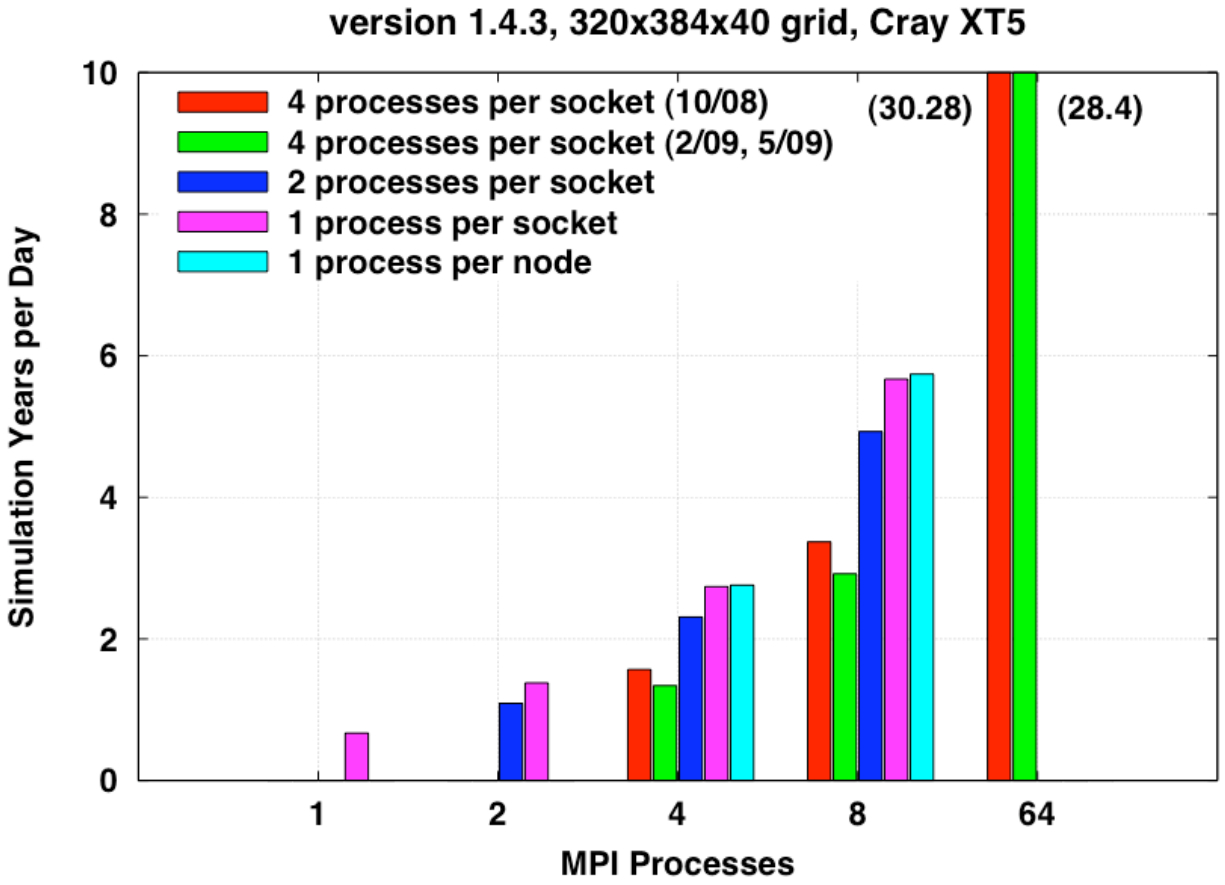
- Spatial locality Apps (like STREAM) see small penalty from increased contention – memory controller/channel limitation
- Temporal locality Apps (like FFT) see moderate improvement
- Even low locality apps (Like RandomAccess) see some benefit

SMP Performance Ratio (MultiCore to SingleCore)

	Jaguar	JaguarPF	% Improvement
FFT	0.704	0.743	5.6%
RandomAccess	0.645	0.671	4.0%
STREAM	0.408	0.383	-6.2%

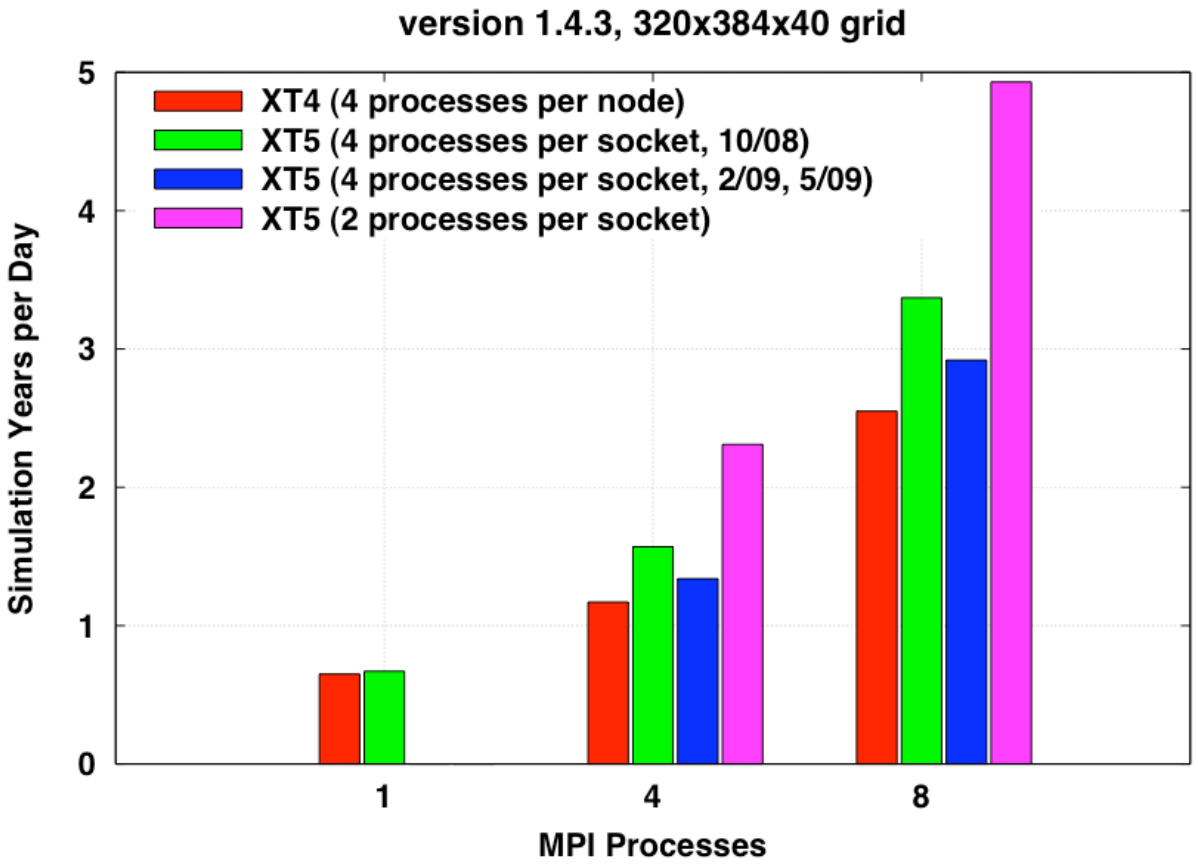
- SMP efficiency improved for apps that weren't bandwidth limited
- Memory bandwidth hungry apps suffer from increased contention
- At JaguarPF scale, 5% ~ 4000 cores
- Lessons:
  - Eliminate unnecessary memory traffic
  - Consider replacing MPI w/ OpenMP on node (see MPI results)

Parallel Ocean Program



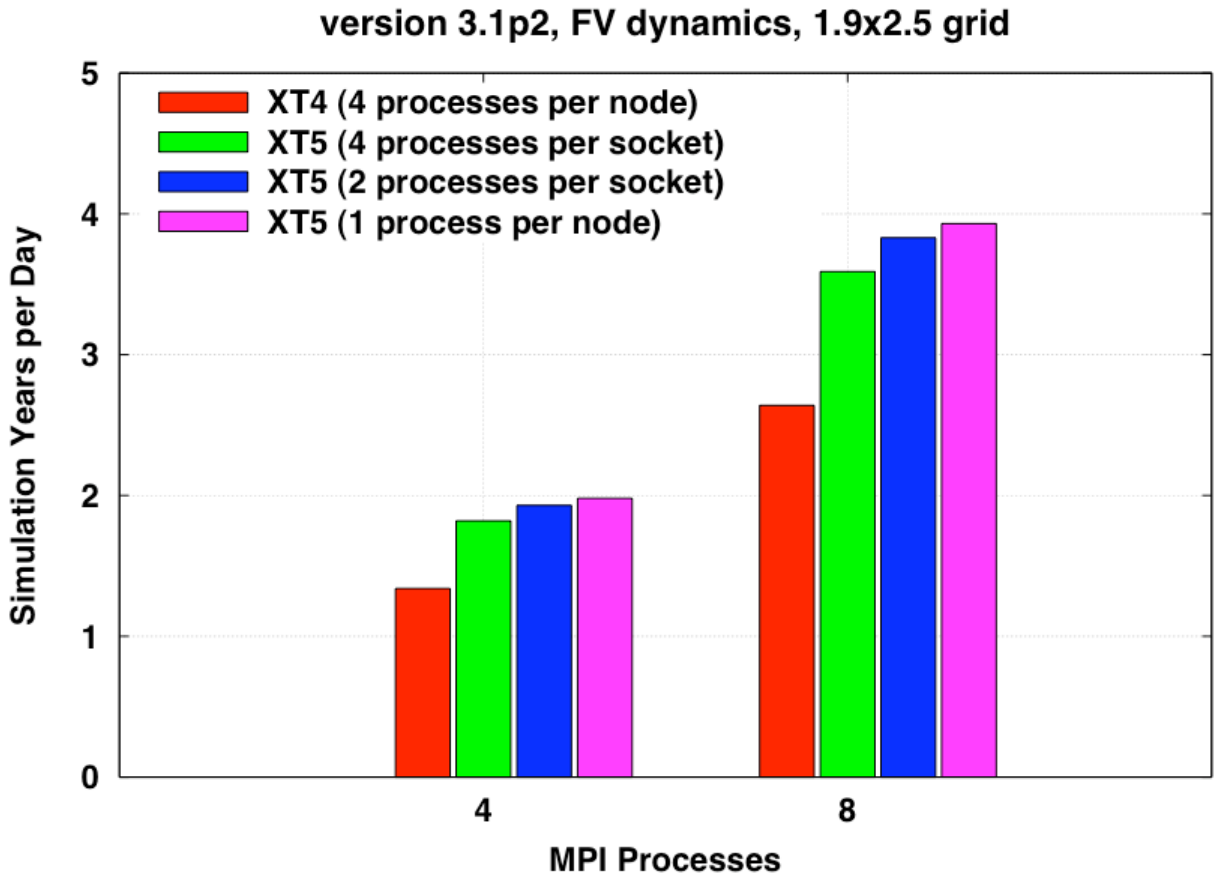
Using POP to investigate performance impact of memory contention. Using all cores in a node can degrade performance by as much as 45% compared to assigning one process per node. It is still much better to use all cores for a fixed number of nodes.

Parallel Ocean Program



For a single process, XT5 performance is nearly the same as the XT4. However, when using all cores in a socket XT5 performance was greater than 1.3X that of the XT4 in Oct. 08, and greater than 1.15X in Feb. 09 and May 09, both of which are greater than the difference in the clock speed.

Community Atmosphere Model

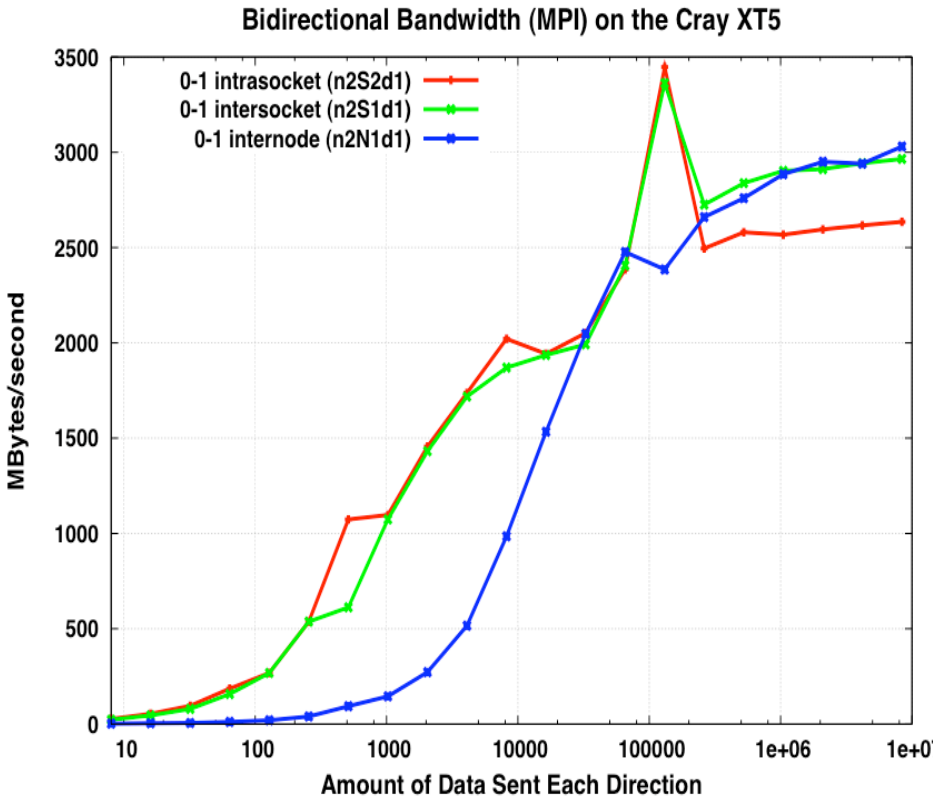


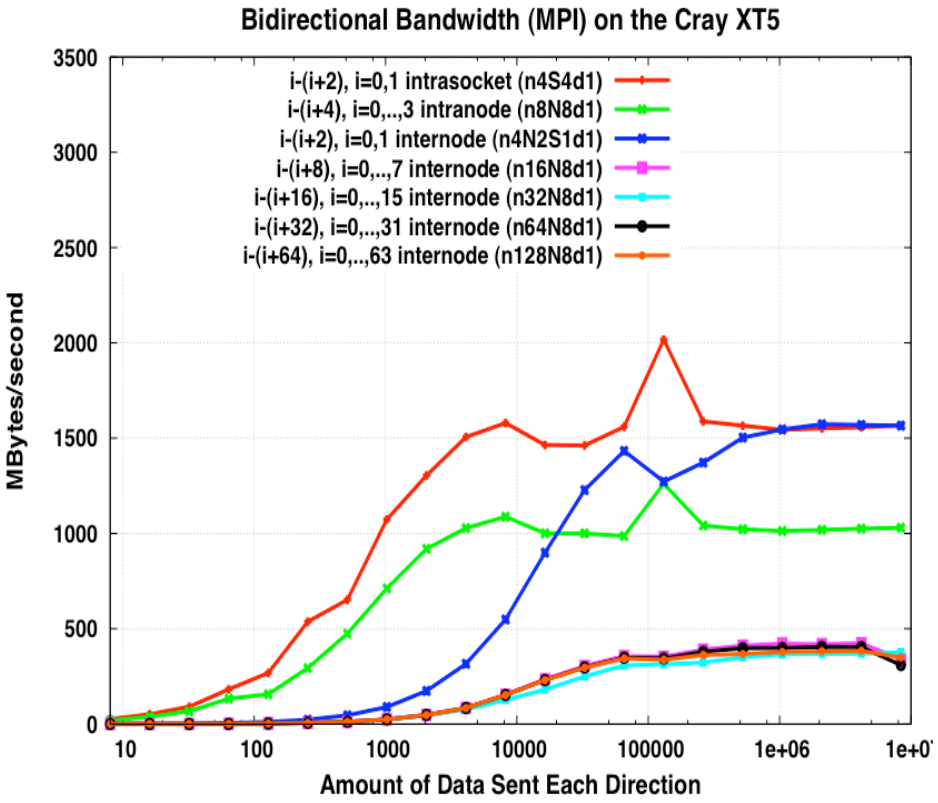
For the Finite Volume dynamics solver, XT5 is 1.38X faster than the XT4 on a single quad-core processor, and 1.36X faster on two quad-core processors. Using the same number of nodes (but only two cores per processor) increases the advantage to 1.44 and 1.45, respectively. Physics dominates runtime in these experiments.

Computation Benchmarks: Summary

1. DGEMM “sanity check” looks good.
2. FFT, RandomAccess, and STREAM demonstrate steadily decreasing advantage of XT5 node over XT4 as contention for (main) memory increases.
3. XT5 per node performance is better than the XT4 per node performance no matter how measure it, for both POP and CAM. Memory contention (?) can degrade performance, especially for POP.
4. For this level of parallelism, OpenMP did not improve performance of CAM for same number of cores (not shown).
5. POP all-core performance has degraded by 15% since October. Performance when not using all cores in a socket is essentially unchanged. CAM performance has not changed significantly over this period, but CAM is more compute intensive for this problem granularity than is POP.

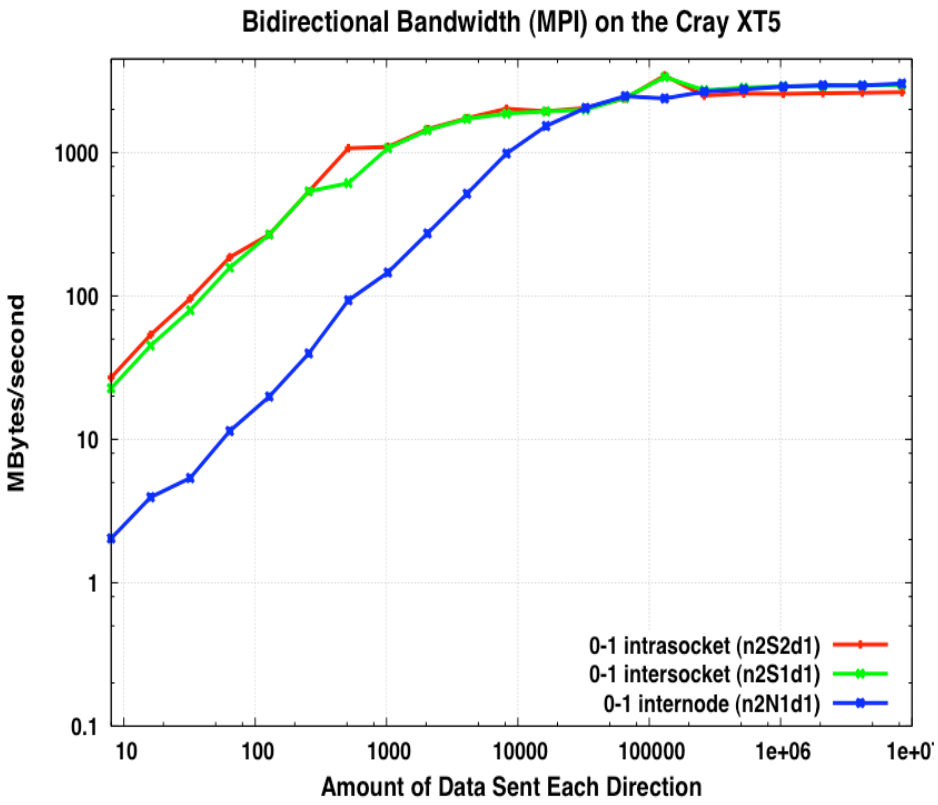
MPI Point-to-Point Performance

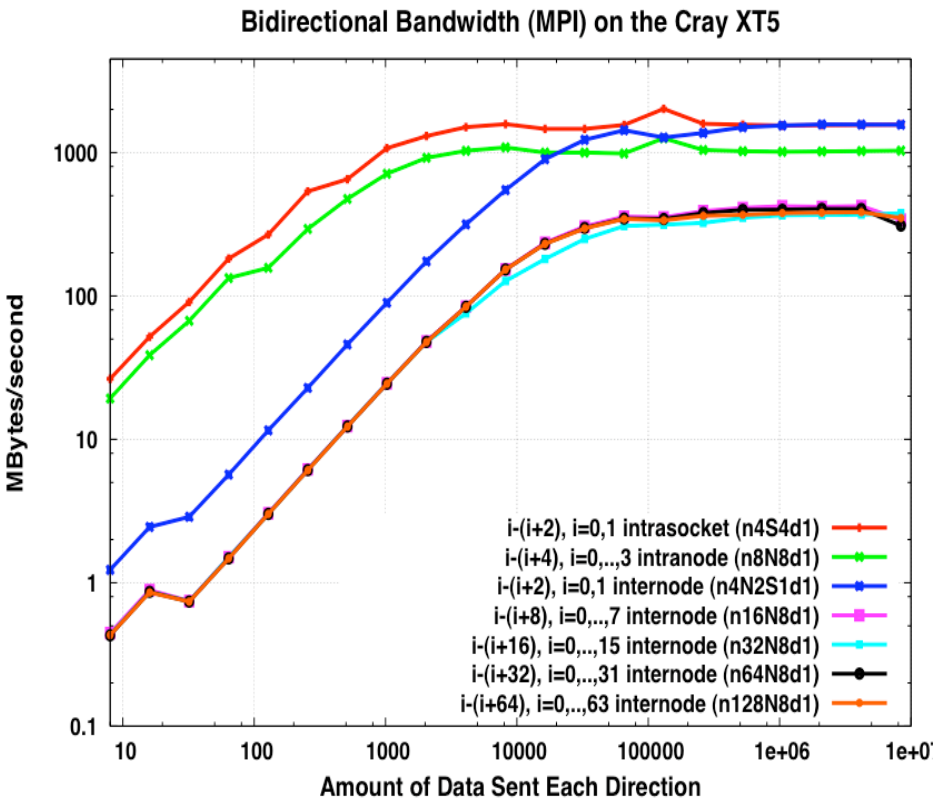




Bidirectional bandwidth for single process pair when single pair communicating and when multiple pairs communicating simultaneously. One pair, two pairs and eight pairs achieve same total internode bandwidth. These experiments were unable to saturate network bandwidth.

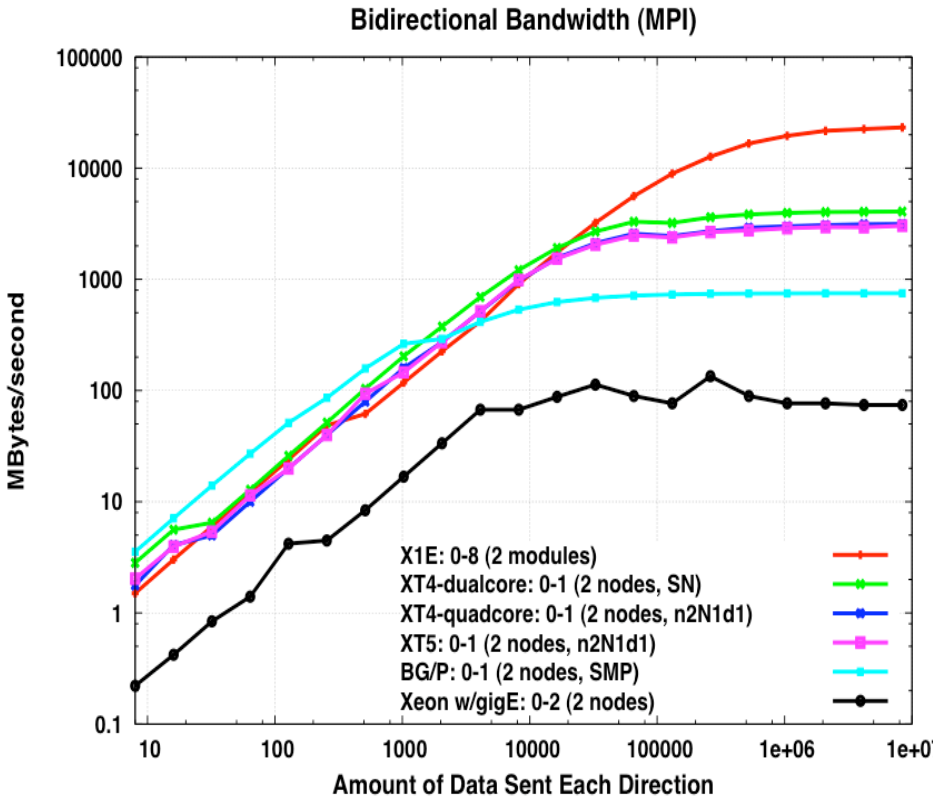
MPI Point-to-Point Performance

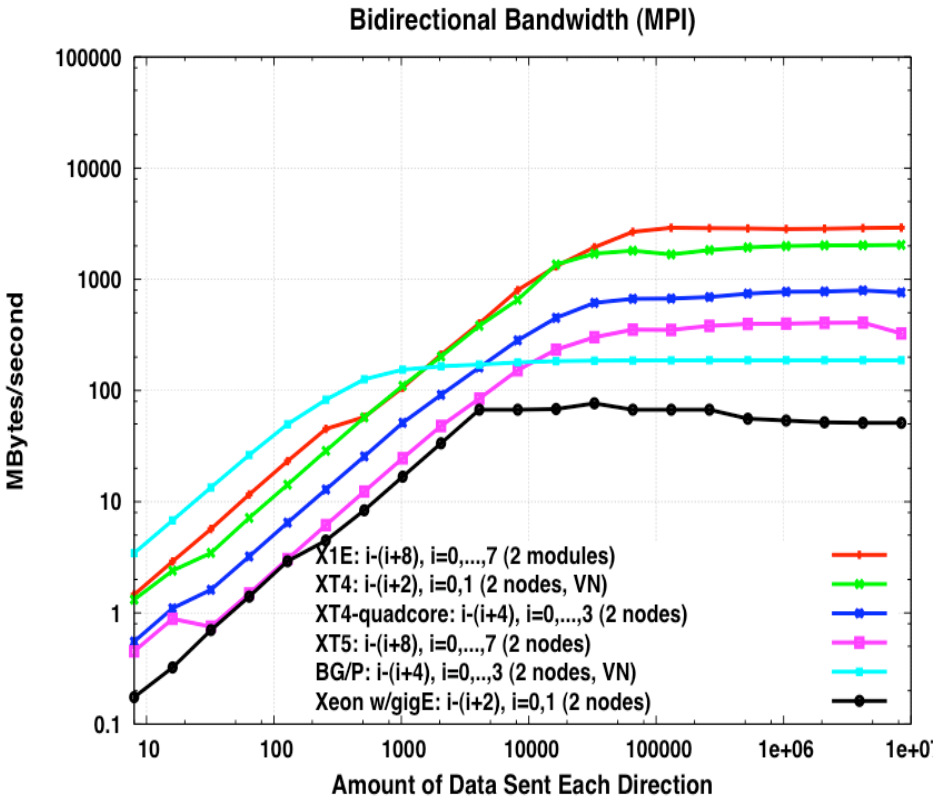




Same data, but in log-log plot. Intranode latency over 10 times lower than internode latency. Internode latency of single pair is half that of two pair, and 1/5 that of 8 pair. Latency is (also) not affected by number of nodes communicating, in these experiments.

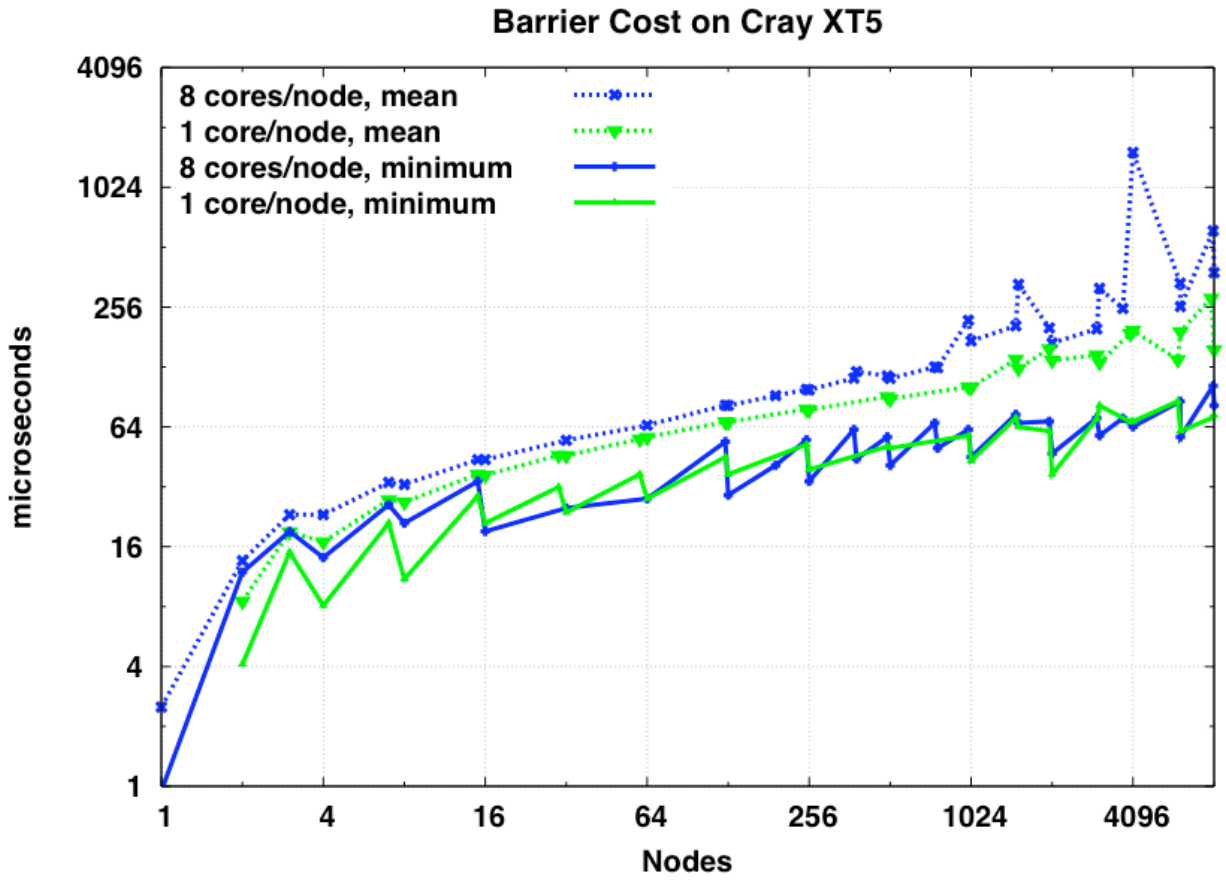
MPI Point-to-Point Performance





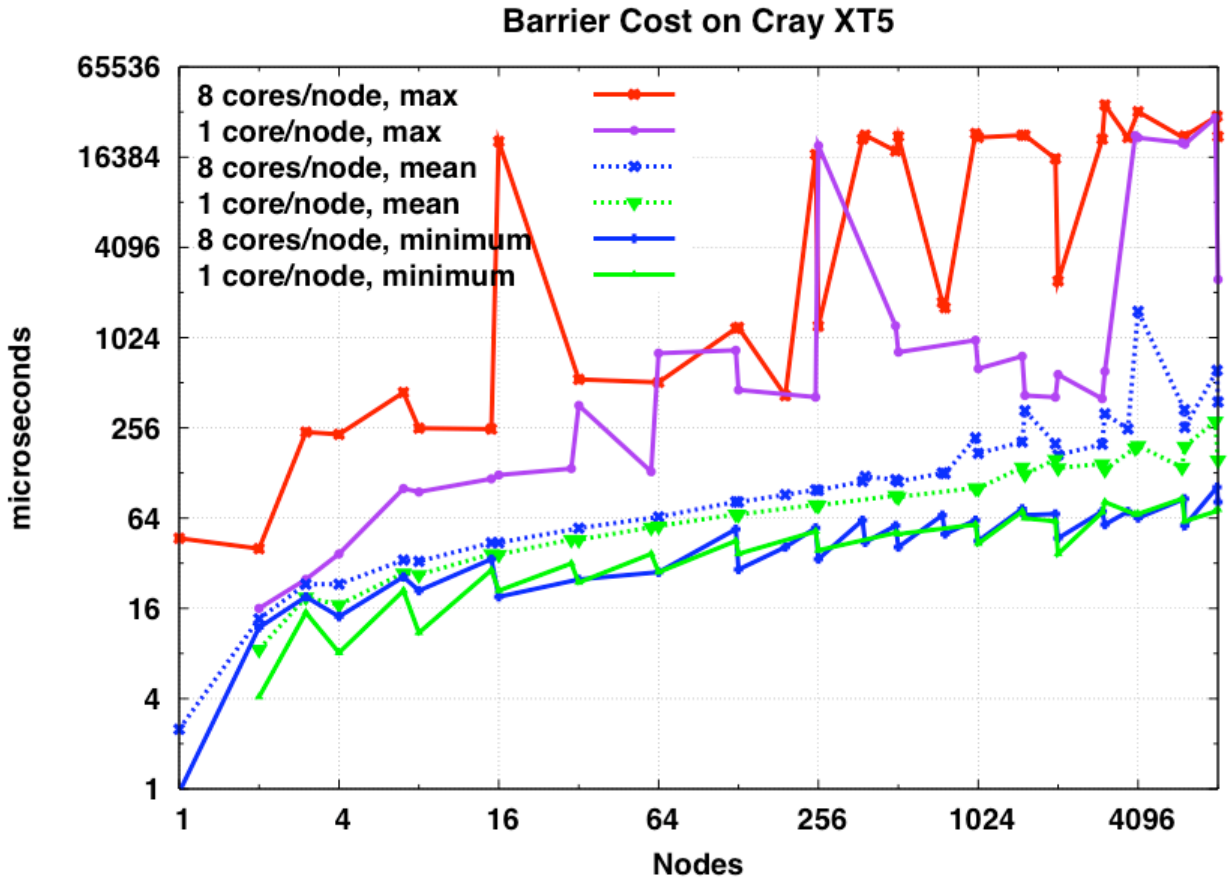
Log-log plots of bidirectional bandwidth between nodes for different platforms, both for a single pair and when all pairs exchange data simultaneously. XT5 and XT4-quadcore demonstrate same total internode performance, resulting in XT5 per pair performance to be half that of XT4 for simultaneous swaps.

MPI_Barrier Performance



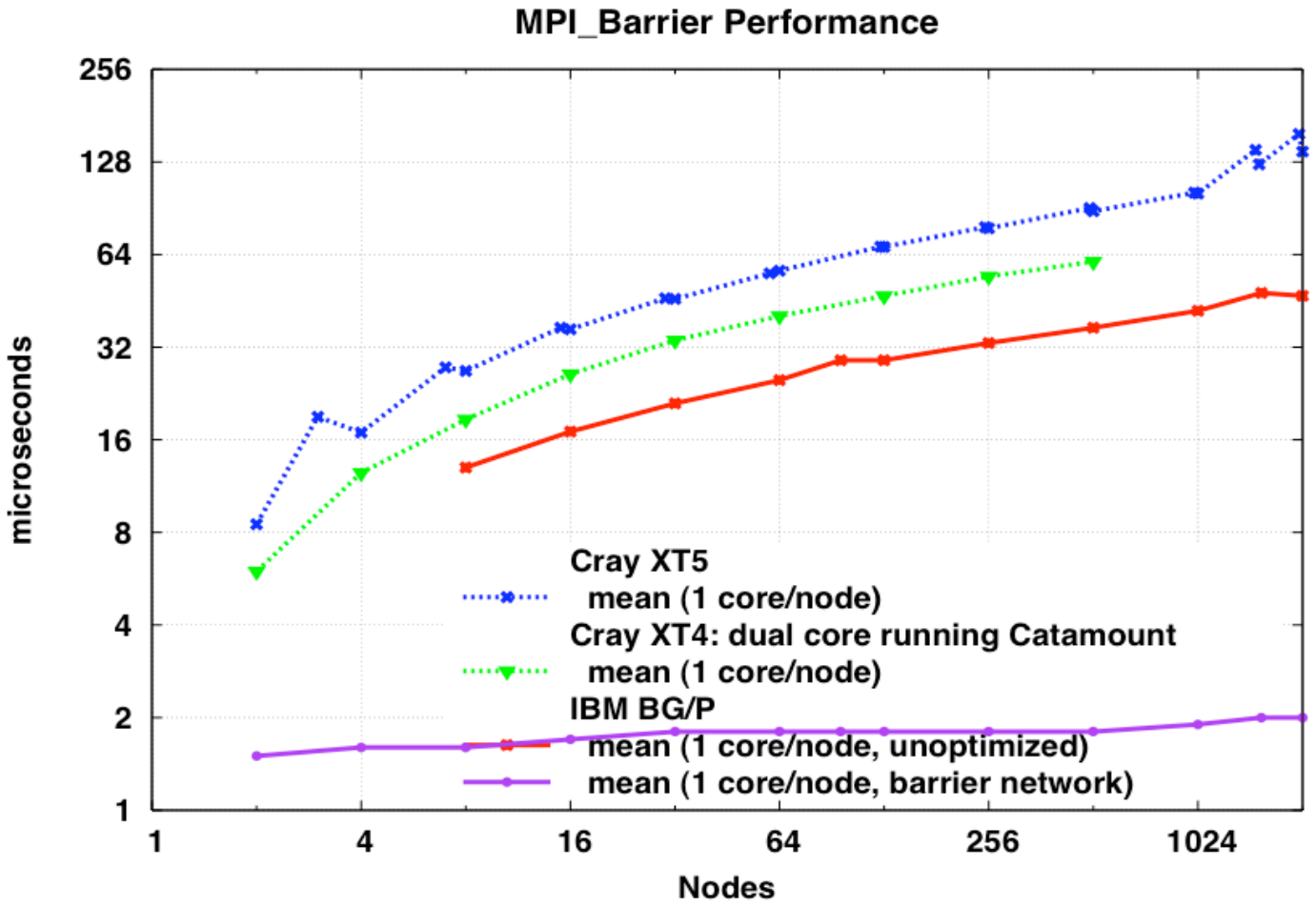
Observed minimum and average `MPI_Barrier` performance for one MPI process per node and for 8 MPI processes per node. Note preference for power-of-two number of nodes for minimum. Data *not* collected on a dedicated system.

MPI_Barrier Performance



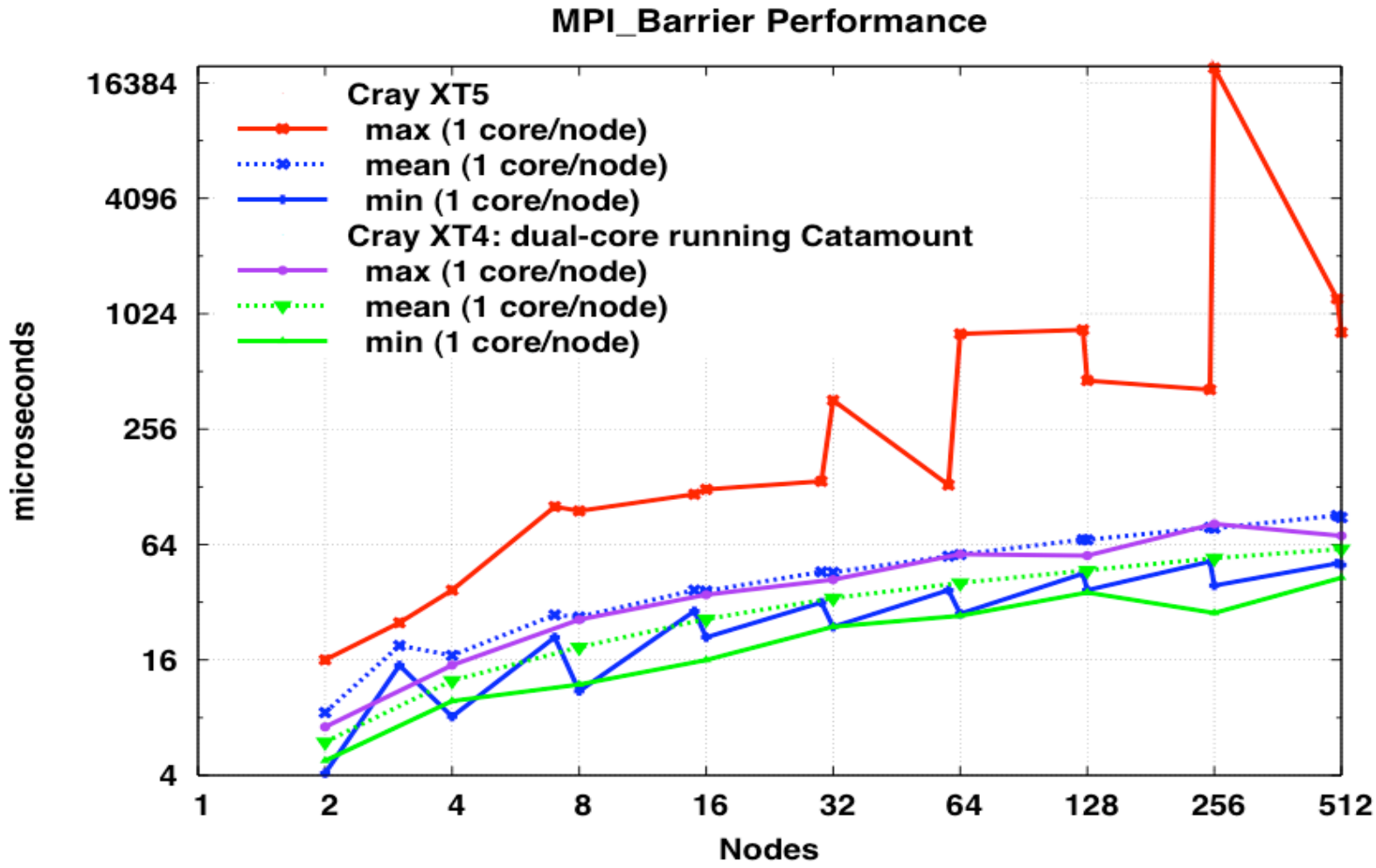
Observed minimum, average, and worst case `MPI_Barrier` performance for one MPI process per node and for 8 MPI processes per node. Worst case tends to increase with node count, but high costs can occur for relatively small node counts also.

MPI_Barrier: Platform Comparison



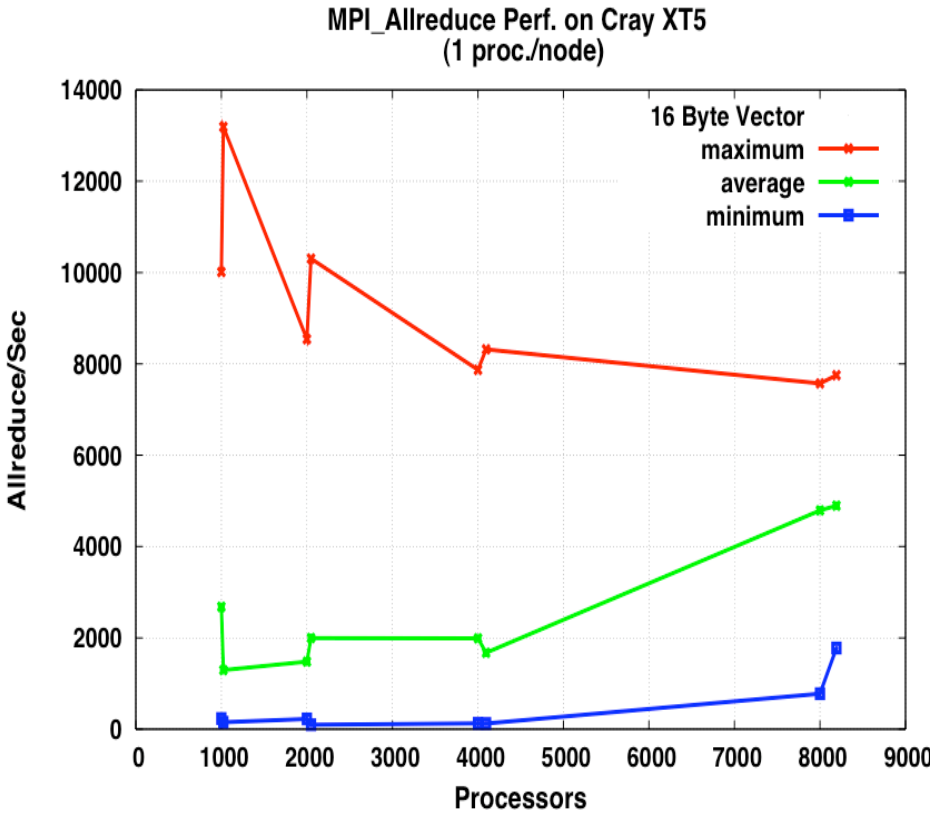
Lower latency? less frequent or smaller performance perturbations? give advantage to Catamount and to BG/P results.

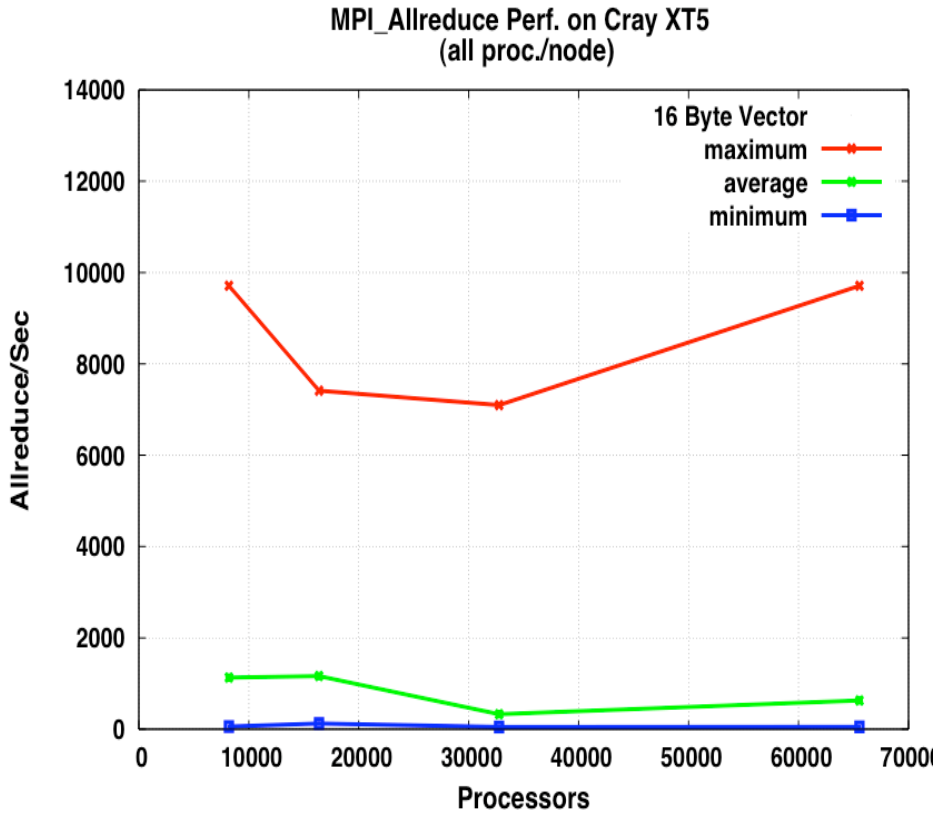
MPI_Barrier: Platform Comparison



Minimum times are comparable between Catamount and CLE results. Big difference is in maximum times.

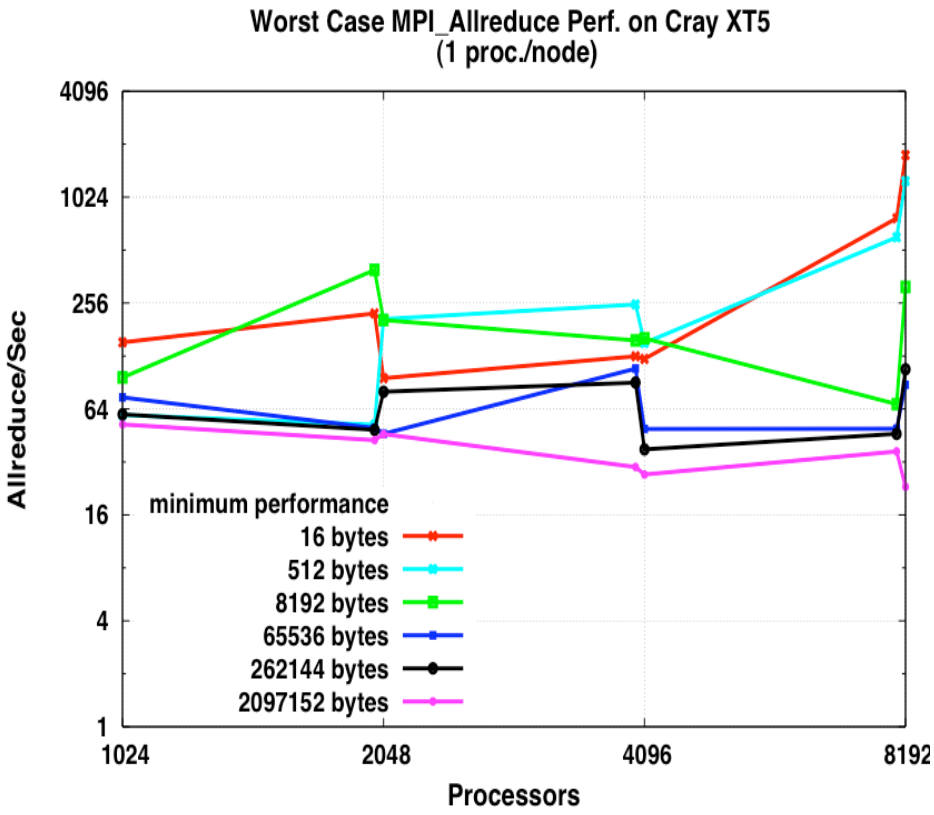
MPI_Allreduce (16B)

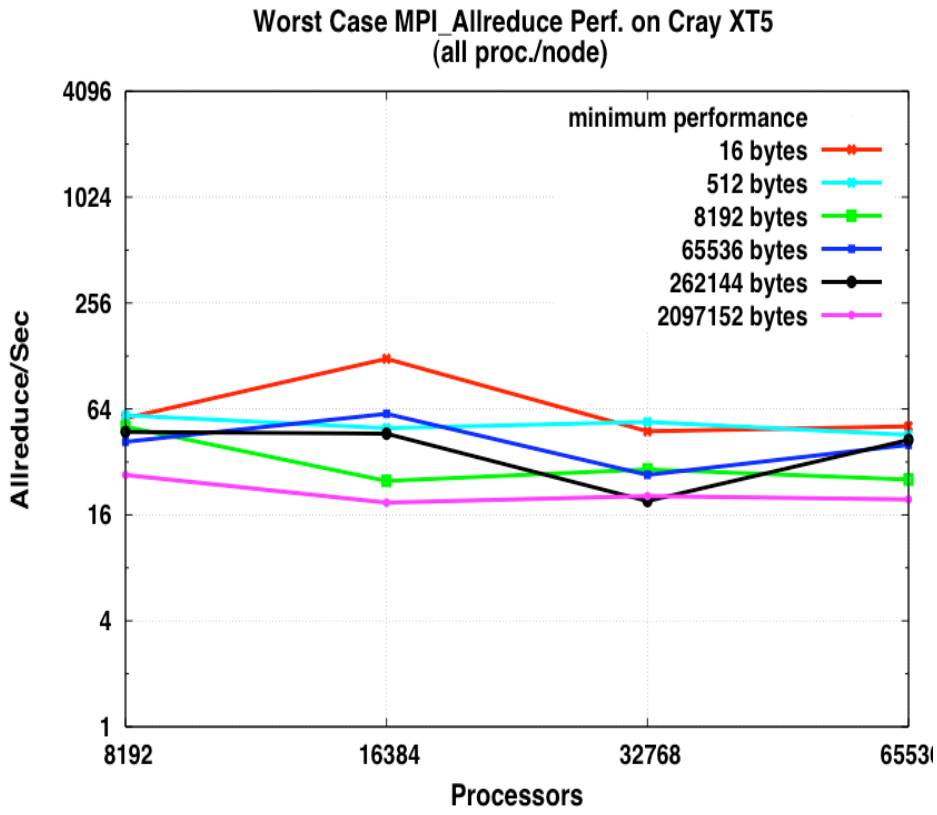




MPI_Allreduce rate (larger is faster) for summing REAL*8 vectors of length 2 when using one process per node and when using all processes per node. Optimal performance is not that much different in the two cases. Average and worst case performance are worse when using all processes per node. Data *not* collected on a dedicated system.

MPI_Allreduce (Worst Case)



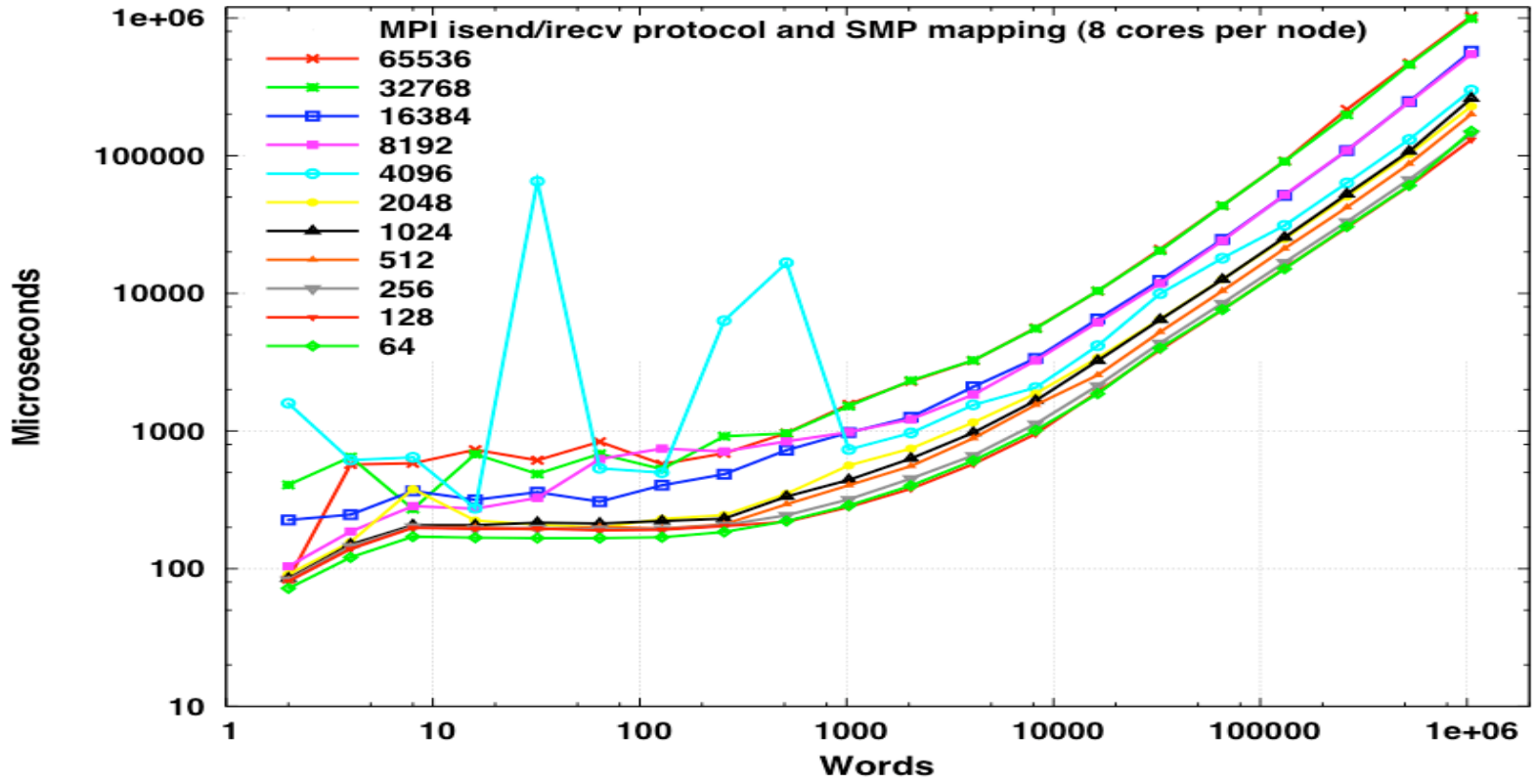


Worst case MPI_Allreduce rate for summing REAL*8 vectors of various lengths when using one process per node and when using all processes per node. Not as sensitive to process count?

HALO Benchmark

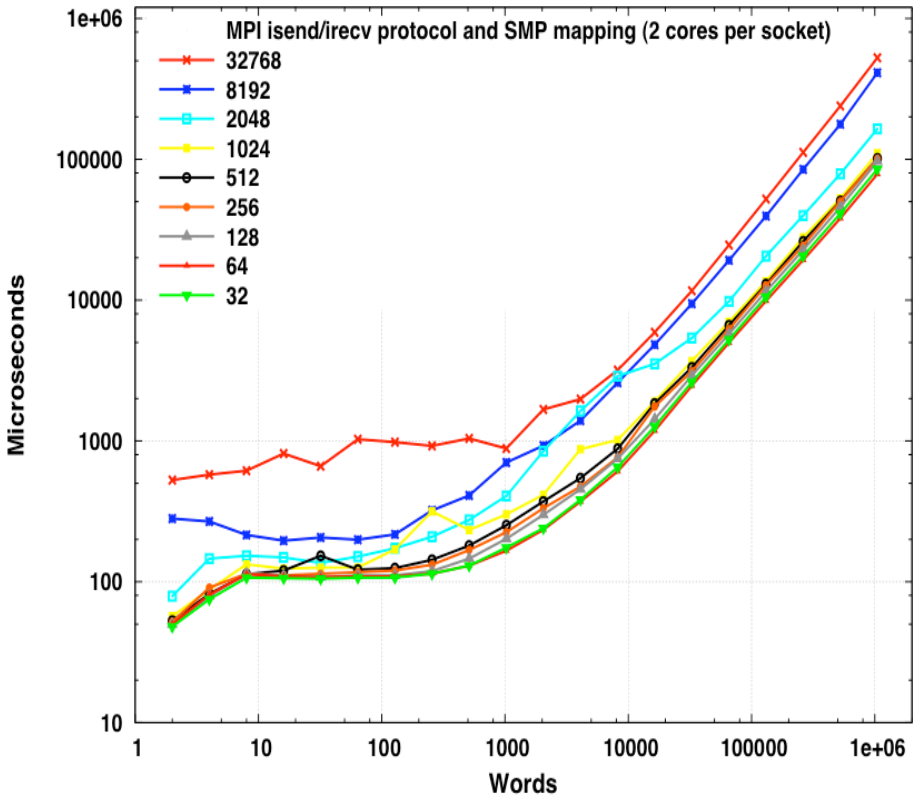
- Alan Wallcraft's HALO benchmark is a suite of codes that measures the performance of 2D halo updates as found in 2D domain decompositions of, for example, finite difference ocean models. Implementations exist for multiple message layers, and for multiple implementations for a given layer. The benchmark measures max time (over all processes) for a small number of repetitions, normalized by the number of repetitions.
- We used HALO to examine impact of process count on performance

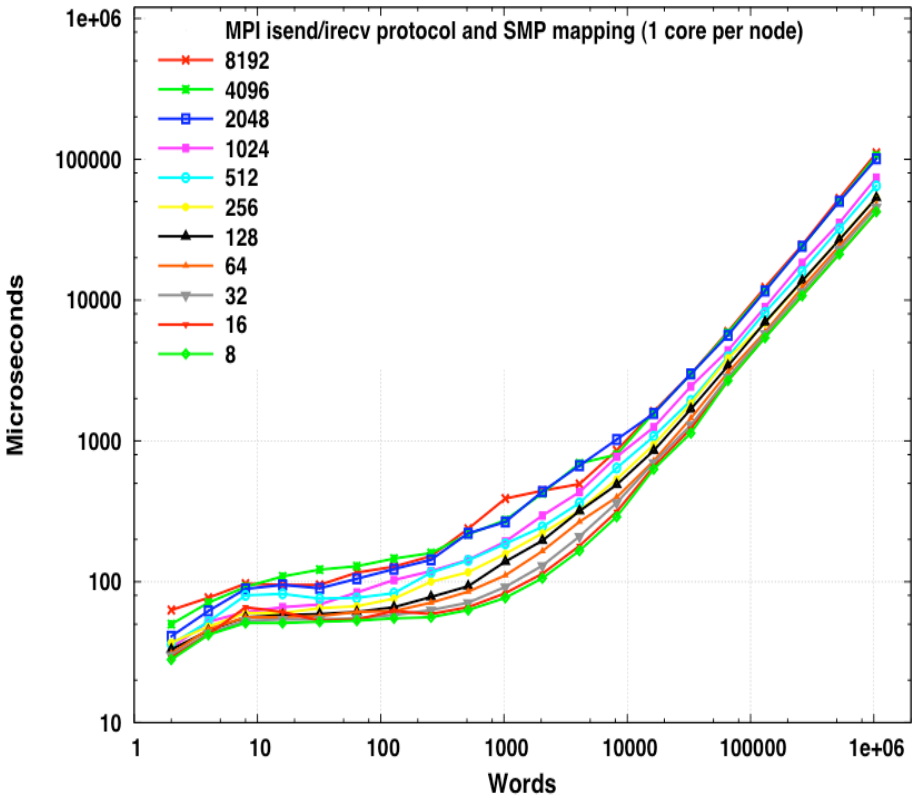
HALO Benchmark: Scaling Comparison



-N 8 experiments for SMP process mapping. The halo exchange is “logically” a local operator, so would like performance to not vary with process count. Process mapping determines actual locality. Ignoring “noisy” runs, cost appears to grow in a deterministic fashion as a function of process count, even for relatively small halos (because ???). Source of noise in noisy data?

HALO Benchmark: Scaling Comparison



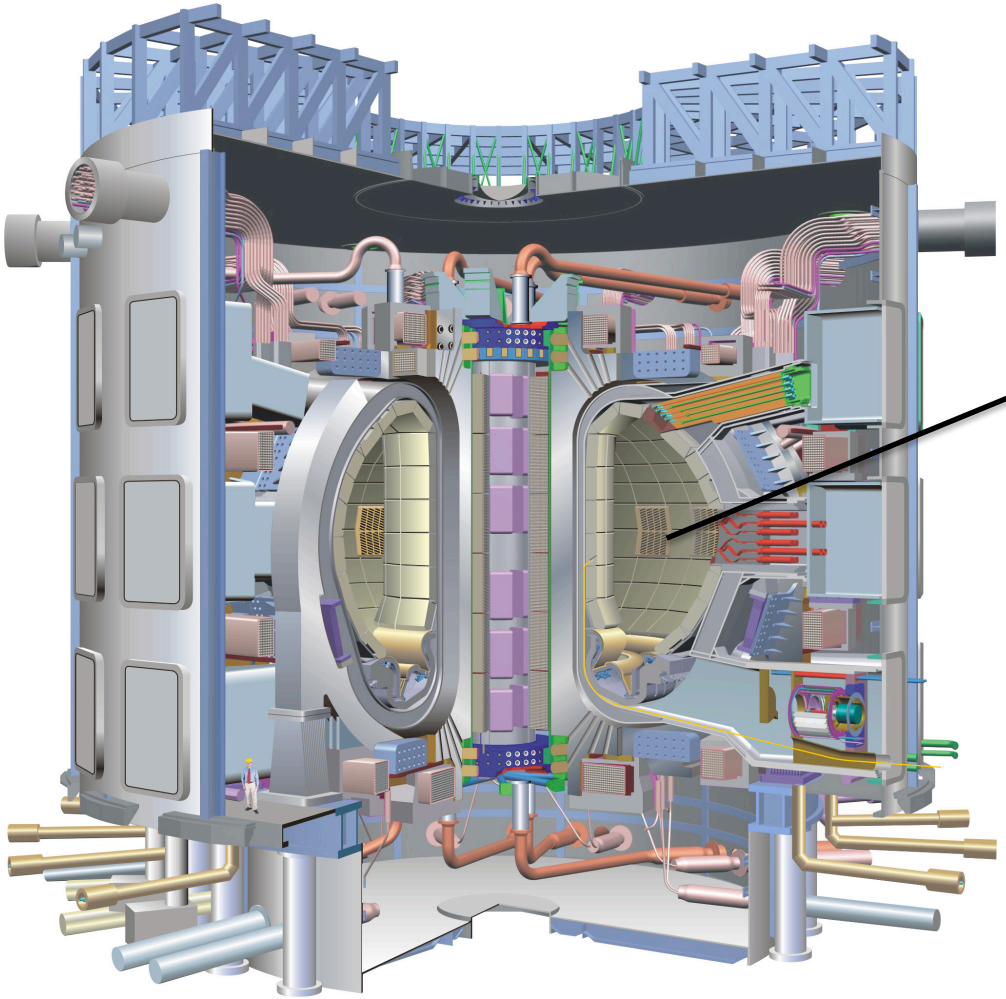


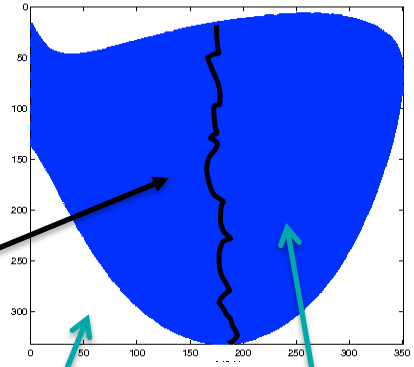
-N 4 and –N 1 experiments for SMP process mapping. Separation by process count holds in both of these cases as well, but “noise” is less evident except for >= 8192 processes for –N 4 .

Communication Benchmarks: Summary

1. Point-to-point performance makes sense, though these experiments do not stress the network. XT5 (8-way) node demonstrates approximately the same maximum internode bandwidth as quad-core XT4 node, and individual processes in an XT5 node may suffer from as much as twice the contention as processes in an XT4 node.
2. Best observed MPI_Barrier performance is reasonable – comparable to that on XT4-dual core when running Catamount operating system. Unfortunately, barrier performance is subject to significant variability, even at small process counts.
3. MPI_Allreduce has performance characteristics similar to that of MPI_Barrier, even for large vectors. Worst case performance appears to be relatively insensitive to processor count.
4. Cost of "logically local" halo update increases with total number of processes. This could be a sign of a poor logical to physical mapping, but regularity of growth may be better explained by an OS jitter-like performance degradation.
5. All performance variability-related issues under active investigation.

AORSA: All Orders Spectral Algorithm*
(Fusion energy modeling)





Solver options:

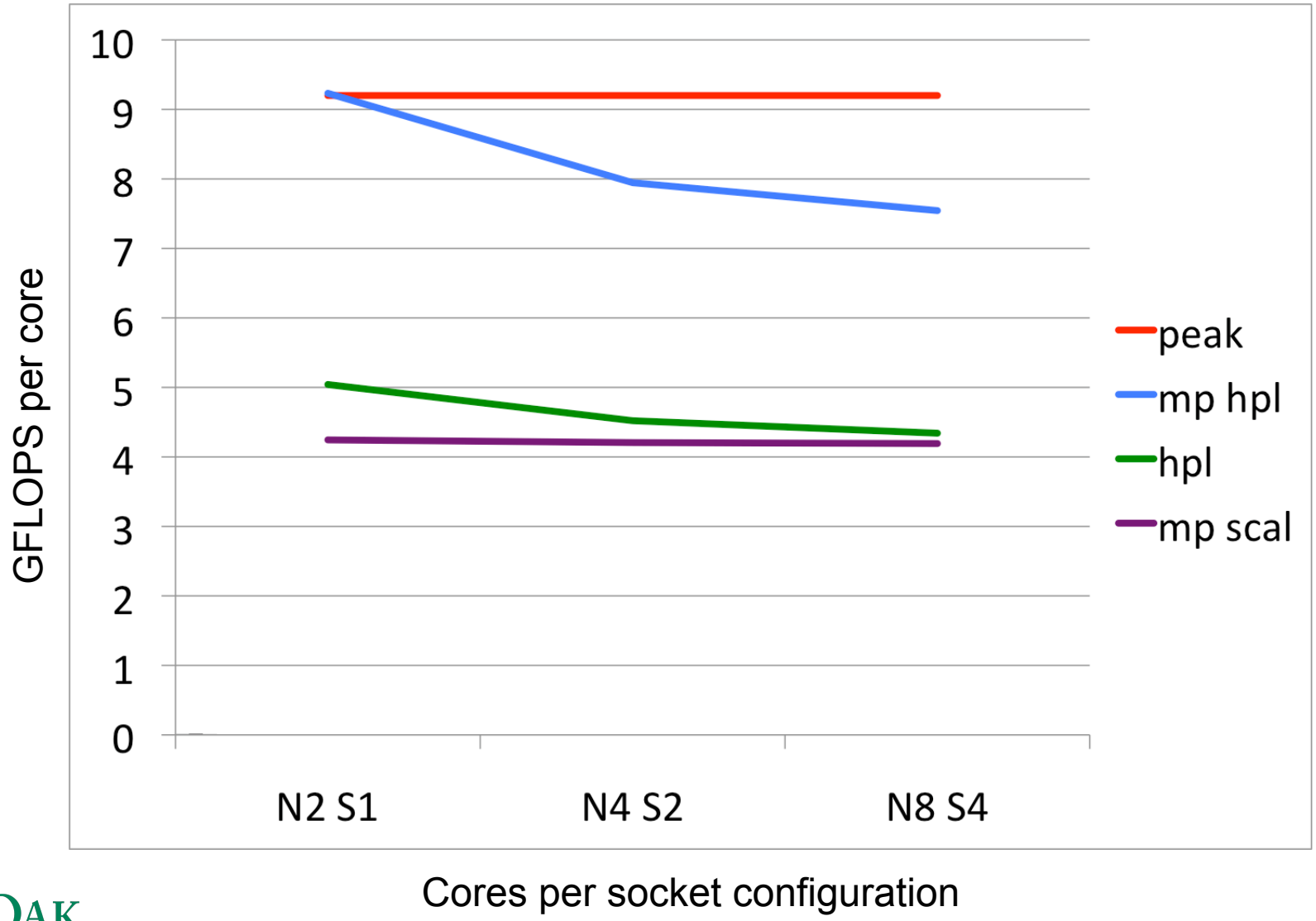
1. HPL
2. Mixed-precision HPL (mphpl)
3. Mixed-precision ScaLAPACK (mpscal)

*SciDAC SWIM project
(NSTD@ORNL, Fred Jaeger, Lee Berry)

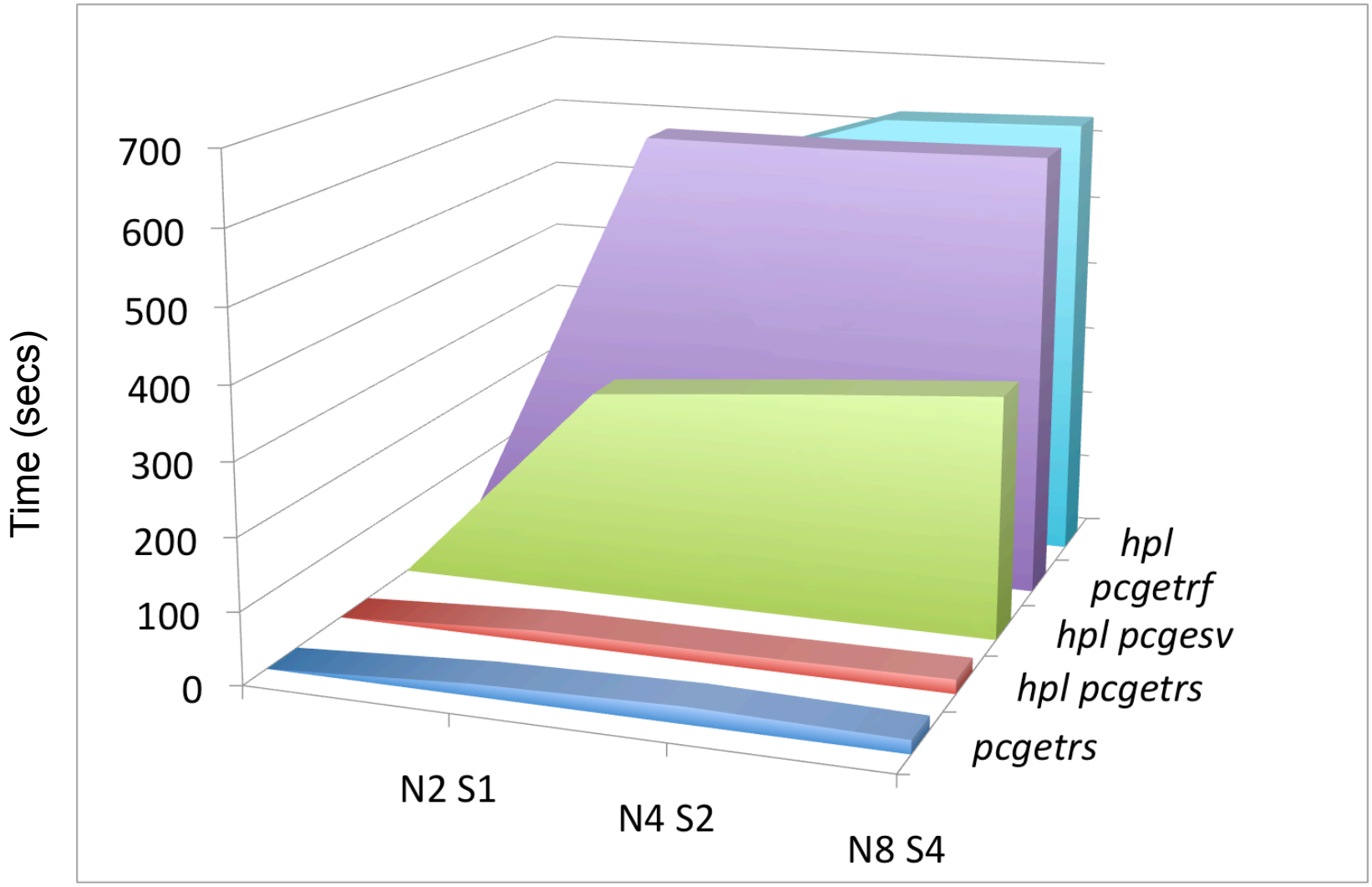
OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

AORSA : Solver performance, 16k cores

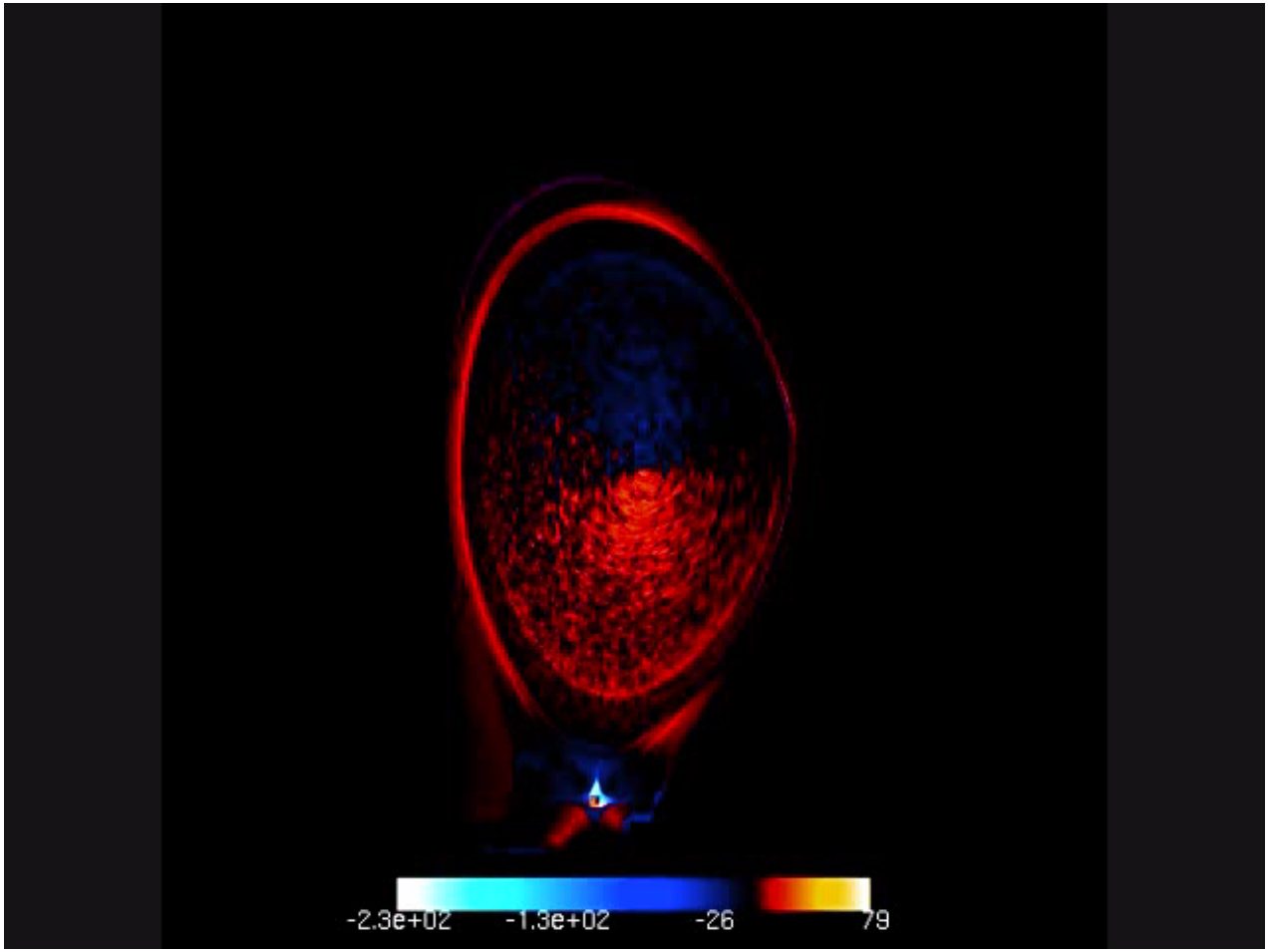


AORSA : Solver performance



XGC1: First full-f gyrokinetic simulation of whole device tokamak plasma

- XGC1 is a particle-in-cell code developed by S. Ku and C.S. Chang as part of the the Center for Plasma Edge Simulation project. It is used to study turbulent transport in magnetic confinement fusion plasmas, and is capable of treating the edge region in tokamak devices accurately.

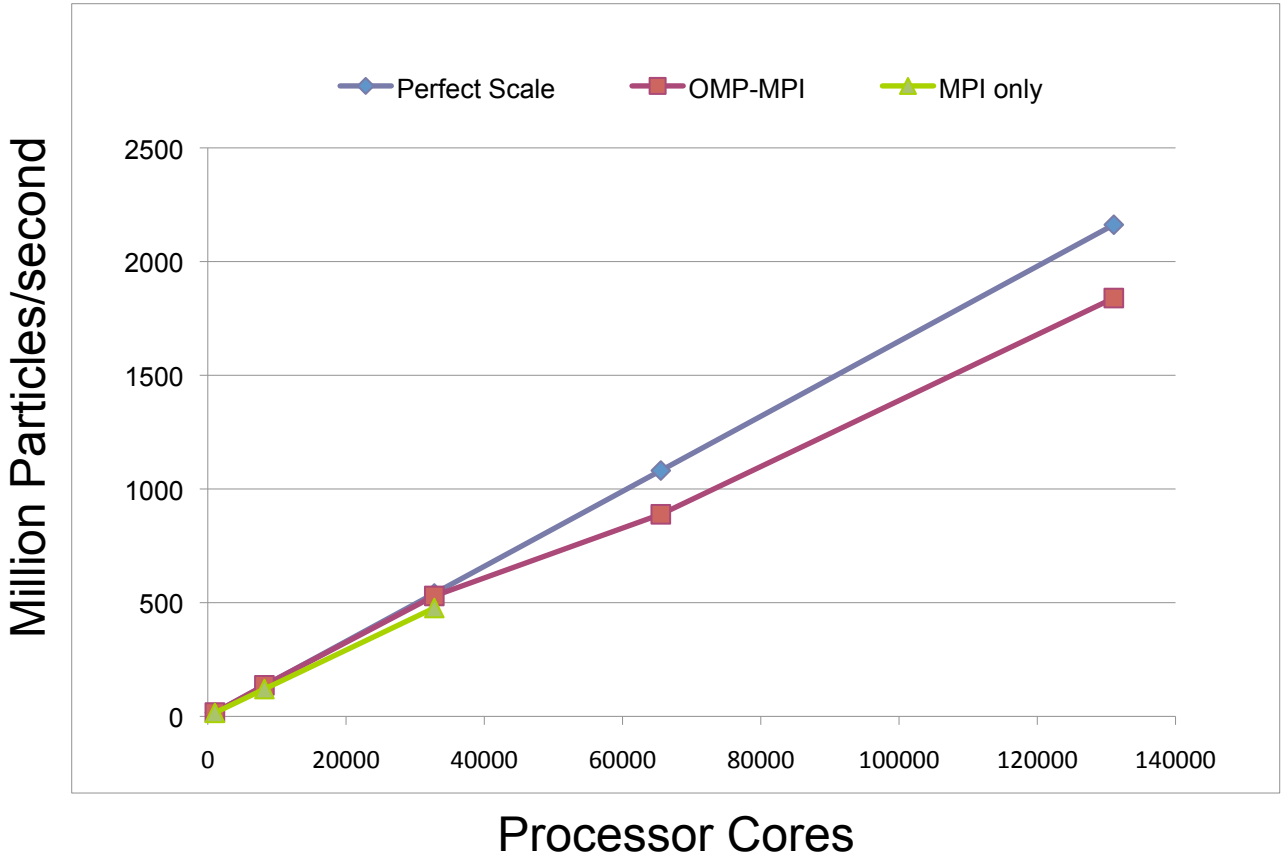


Turbulent electrostatic potential

Scaling XGC1 to 150,000 cores

- XGC1 uses PETSc to solve an elliptic problem at each timestep. Performance experiments are typically weak scaling in total particle count and strong scaling in grid size.
- High performance communication algorithms failed when using more than 8000 processes. Developed equally high performing alternatives (utilizing flow control) that scale to over 32768 processes (largest tried). See R. Mills talk.
- Solution of Poisson problem on fixed size grid scaling poorly at large process counts, and is primary “site” of performance variability. Introducing OpenMP parallelism to minimize number of MPI processes. Also considering alternative solvers.
- OpenMP parallelism of loops over particles dominated by search for particle location in underlying grid are less efficient when using 8 threads than when using 4 threads, possibly due to NUMA effects when threads not local to a single socket.

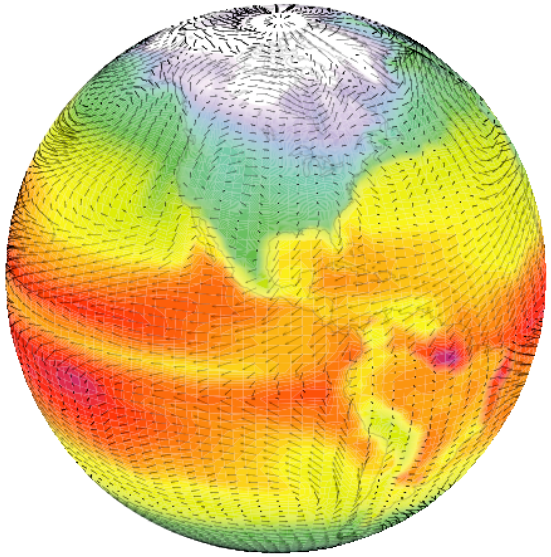
XGC1: Current performance scaling



- Reasonable scalability out to 131,072 cores when using OpenMP (4 or 8 threads). Still working to improve both scalability and performance.

Community Atmosphere Model (CAM)

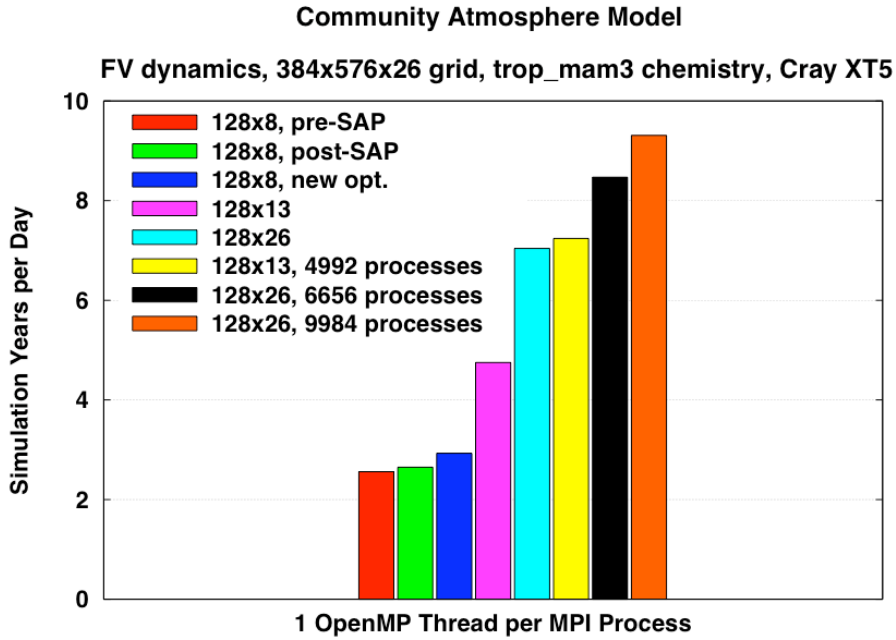
- Global atmosphere circulation model developed at National Center for Atmospheric Research, with contributions from DOE and NASA funded researchers. Used as atmospheric component of CCSM.
- Tensor product *longitude* x *latitude* x *vertical level* grid over the sphere
- Hybrid MPI/OpenMP parallelism and 1D or 2D domain decomposition.
- Timestepping code with two primary phases:
  - *Dynamics*: advances evolution equations for atmospheric flow; 2D lat-lon decomposition in one phase and 2D lat-vert in another phase, requiring two remaps per timestep when using 2D decomp.
  - *Physics*: approximates subgrid phenomena, such as precipitation, clouds, radiation, turbulent mixing, …; support for arbitrary decomposition on horizontal grid.
  - "coupling" between the dynamics and the physics requires remap when physics load balancing is enabled.

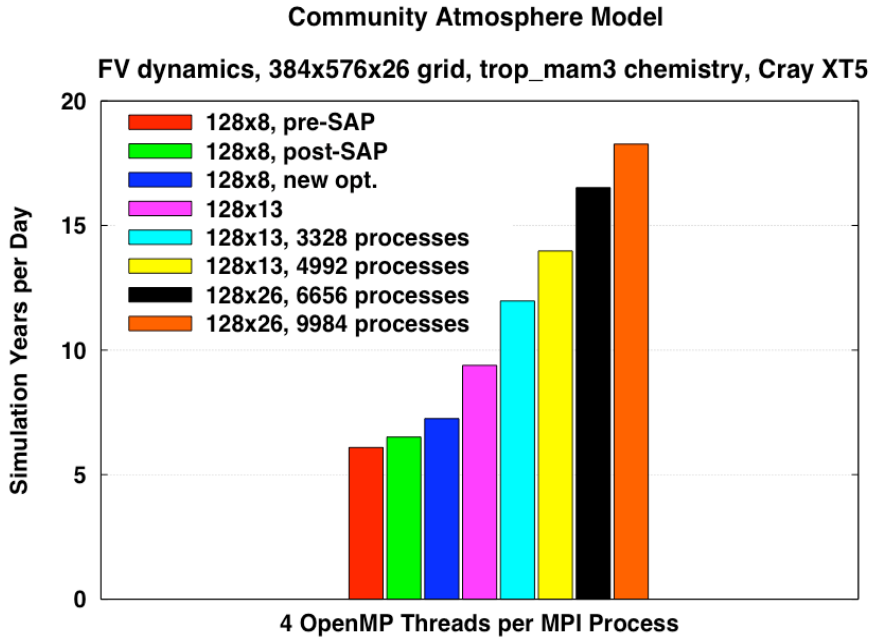


Scaling CAM to 40,000 cores

- Fixed “0.5 degree” problem can use a maximum of 3328 MPI processes in phase of code with most limited parallelism. Utilizing more processes in other phases and 4-way OpenMP parallelism permits approximately 40,000 cores.
- Communication between phases and gathers/scatters associated with I/O required introduction of flow control, for reliability and for performance. See R. Mills talk.
- In original implementation, OpenMP and MPI parallelism were applied to the same loops. Have been moving OpenMP to other loops, for cases when MPI exhausts parallelism in original loops.

CAM Performance Scaling





Contributions to performance from exploitation of increasing amounts of parallelism.

Application Codes: Summary

- AORSA
  - Periodically revisiting code is important, including evaluating and exploiting external libraries and technologies.
  - Both memory contention within the node and contention within the network are evident. A coordinated approach is needed: OpenMP?
- XGC1
  - Flow control is important in communication algorithms for large process counts.
  - OpenMP important to control cost of Poisson solve and other MPI communication overheads.
  - NUMA affects performance when using OpenMP with only one MPI process per node.
- CAM
  - Flow control is important in communication algorithms for large process counts, especially gathers and other mismatched numbers of senders and receivers.
  - At scale, need to revisit placement of OpenMP parallelism in code.