SciDAC
Scientific Discovery through Advanced Computing

Some Practical Issues in Evaluating and Comparing Methods on Parallel Computers

Patrick H. Worley
Oak Ridge National Laboratory

Frontiers of Geophysical Simulation
August 20, 2009
National Center for Atmospheric Research
Boulder, CO

Oak Ridge National Laboratory

Office of Science
U.S. Department of Energy

SciDAC
Scientific Discovery through Advanced Computing

Acknowledgements

- The work described in this presentation was sponsored by the Climate and Environmental Sciences Division of the Office of Biological and Environmental Research and by the Office of Advanced Scientific Computing Research, both in the Office of Science, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
- These slides have been authored by a contractor of the U.S. Government under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.
- This work used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

Oak Ridge National Laboratory

Office of Science U.S. Department of Energy

Paths to Peta/Exa/…-Scale

A spectrum of approaches have been proposed, ranging from

- Fully explicit methods, to minimize the need for nonscalable (in process count) nonlocal communications,

to

- Fully implicit methods, to minimize nonscalable (in problem size) decrease in timestep

with a many hybrids and technology "enhancements" for each of these, e.g.,

- Mixed explicit/implicit (over operator, over domain)
- Parallel-in-time, Waveform relaxation, asynchronous, and other time relaxation or blocking methods
- HW accelerators (minimizing the required process count)
- *many* different implicit system solvers

Scalability Digression

1. No algorithm scales on traditional parallel computers, given a reasonable definition of scalability, for most (all?) geophysical problems.
2. Some algorithms scale worse than others. Good choices will depend on specifics of problem and target platform.
3. For a range of problem sizes and processor counts, a particular algorithm may scale well enough on a particular parallel architecture that statement #1 is a theoretical rather than a practical issue.
4. For the current generation of massively parallel computers, statement #1 is often a practical issue.

“Blast from the Past”

Simple performance models (ala 1980s) with current (or anticipated future) system performance constants can be used to establish regimes of performance competitiveness and limits to scalability.

- While useful, this analysis is far from sufficient as it ignores numerical advantages/disadvantages of different methods.
- Rest of talk is a hand-waving discussion of example simple models, to demonstrate the process.
- My goal is to remind people that a lot can be done with simple models/arguments to understand promise (or problems) before developing a full implementation.

Basic Explicit Method Models I

For methods with local bases: finite difference/element, spectral element, …

(T/dt)(local computation + halo update)*

where *T* is final time and *dt* is the timestep, or

(T/dt)(max(local computation, halo update))*

if can overlap communication and computation, or

(T/dt2)(LocalComp(n)+ HaloComm(n))*

where *dt2 = n*dt* and communicate a large enough halo to allow local computation to proceed for *n* steps without further communication (trading off fewer communications with redundant computation and larger communication volume), or

Basic Explicit Method Models II

$$(T/dt1)*(LocalComp1 + HaloComm1) +$$
$$(T/dt2)*(LocalComp2 + HaloComm2)$$

for a simple 2 level subcycling algorithm, where usually *dt1 = n*dt2* for some integer *n*, or …

There is typically some global communication as well (e.g., CFL estimator). Even when using an asynchronous algorithm with locally determined timestep size, some global coordination is required. (Some) explicit methods are relatively simple to model (e.g., Chris Kerr's experience with cubed sphere FV on the XT4), but there can still be significant complexities at scale or when computation is inhomogeneous:

Basic Explicit Method Models III

sum_i *(max_p(LocalComp(i,p) + HaloComm(i,p))*

as an “upper” bound on cost when the local communication and computation costs are not uniform in time or space. Here *i* is the timestep index and *p* is the process id. As the Halo communication is a function of the assignment of processes to processors (and the underlying network topology and cluster architecture) and is affected by the load assigned to the neighbors that it is communicating with, this is not an easy function to estimate accurately. However, effective rates can be measured, and system specifications can be used to determine lower bounds on performance.

Basic Implicit Method Models I

Solver (and solver implementation) specific:

– (simple) CG

(T/dt)(local computation +*

N(local solver computation + halo update + global sum))*

where *T* is final time, *dt* is the timestep, *N* is the number of iterations (estimated), halo update is for the residual calculation, and global sum is for the inner product calculations. Can also overlap (some) communication with computation.

Basic Implicit Method Models II

– Operator split with (simple) CG for fast modes?

(T/dt1)(LocalComp1 + HaloComm1) +*

(T/dt2)(LocalComp2 +*

N(SolverComp + HaloComm2 + global sum))*

where *dt1* is the timestep for the "slow" mode, and dt2 is the timestep for the "fast" modes.

Basic Implicit Method Models III

- CG with non-local preconditioner …
- Multigrid solver (V cycle: series of halo updates of different sizes and process separations? Equivalent to a specially constructed reduction + broadcast?)
- Domain decomposition methods: Schur complement, Schwarz, …

Check the literature, and/or roll your own. While much has changed in the past 20 years, models were generated for most of the basic algorithm classes.

Establishing Costs: Kernel Benchmarks

1. HALO

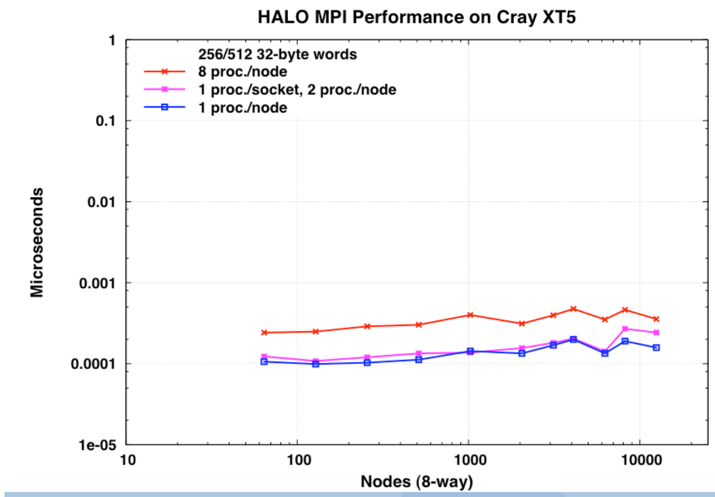
   Alan Wallcraft's benchmark for measuring halo communication performance, motivated by one of his ocean codes, and used to also evaluate a number of different implementations of the operator.

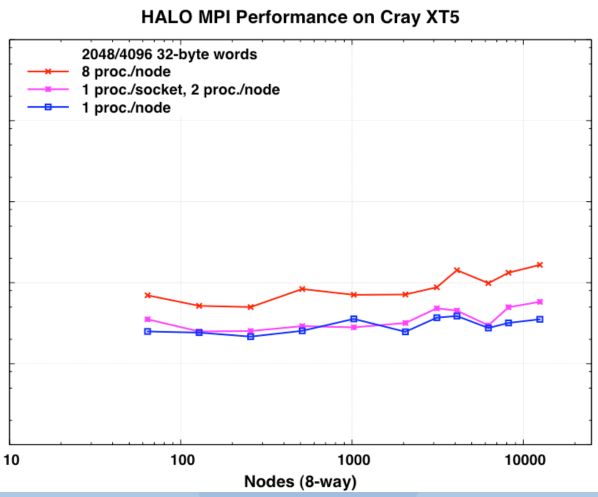
2. ALLREDUCE

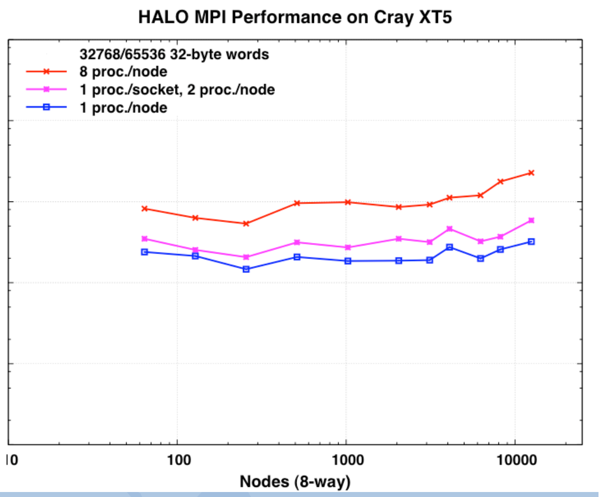
   My benchmark for evaluating both MPI_Allreduce and point-to-point implementation of the allreduce collective.

It is important to devise kernels that accurately reflect algorithm to be modeled. As such, these kernels have limited applicability, and are only suitable to identify issues.

HALO on Cray XT5



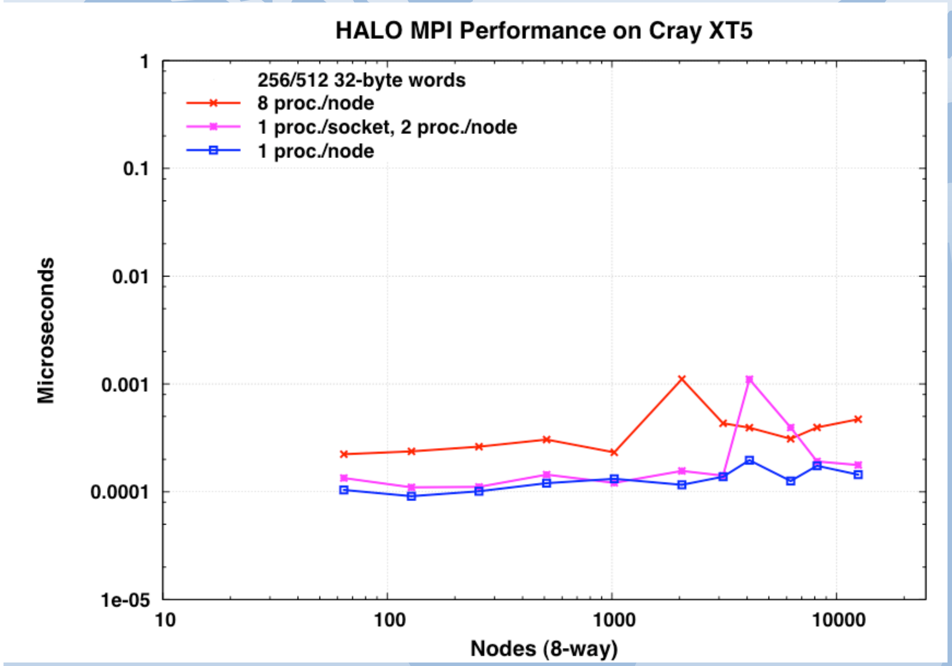


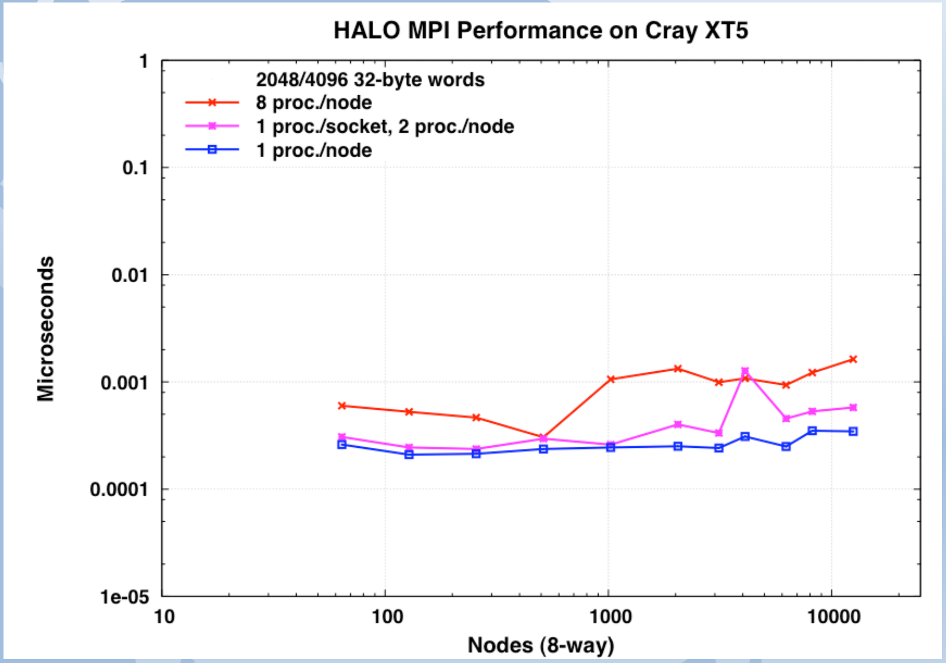


Average cost of halo update measured over approximately 3 seconds worth of iterations. Would need to run additional experiments to determine sensitivity to different node assignments. Should also examine impact of different process-to-processor assignments. Factor of 3X advantage from not using all processes in node. Some mild growth in cost with process count for larger haloes.

SciDAC Scientific Discovery through Advanced Computing

HALO on Cray XT5





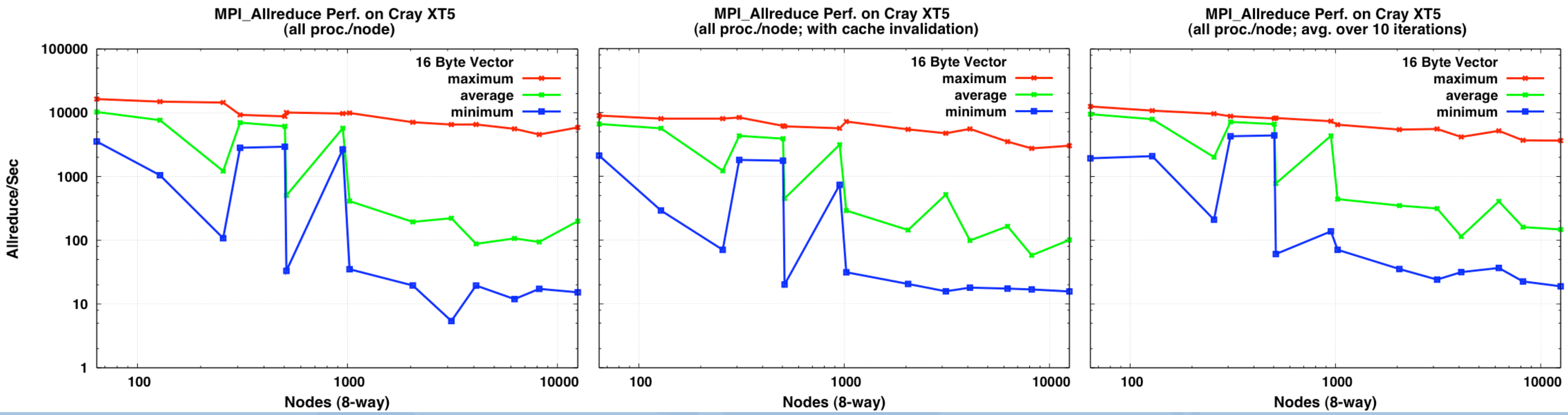
Some experiments with smaller haloes show signs of performance variability or perturbation. These may also impact large halo updates, but are relatively less important. Difficult to quantify nature of variability as MPI_Barrier is also subject to performance variability.

Oak Ridge National Laboratory

Office of Science U.S. Department of Energy

SciDAC
Scientific Discovery through Advanced Computing

ALLREDUCE on Cray XT5

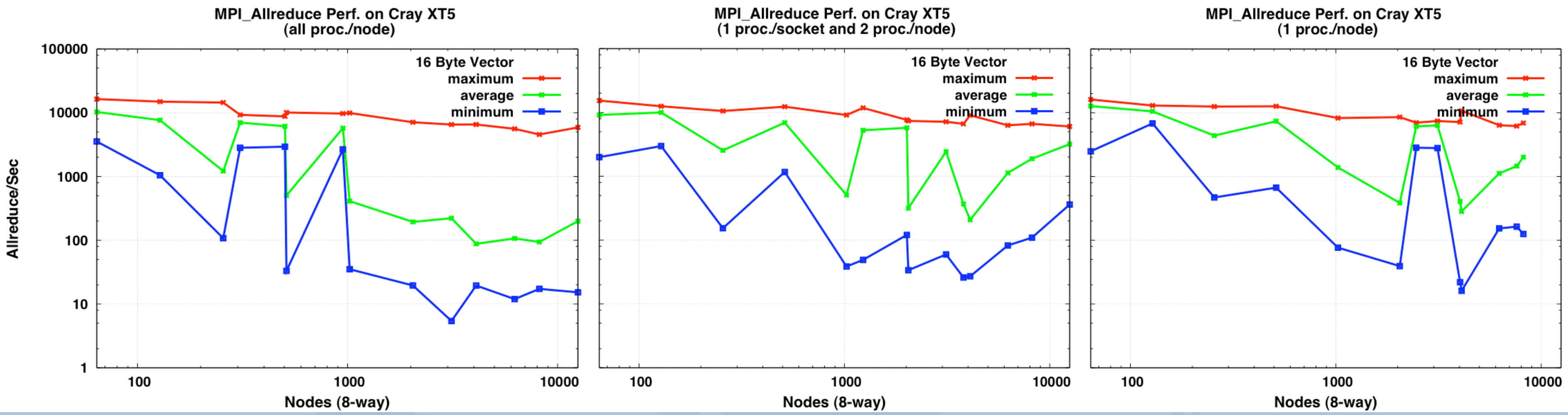


Measured performance of single MPI_Allreduce (global sum of 2 real*8 values), single MPI_Allreduce after cache invalidation, and average performance of 10 MPI_Allreduces, all using all processor cores in a node. Performance graphed as inverse runtime, for easier display, for best, average, and worst case performance over 120 experiments. Note that best performance scales very well, but is not typical for larger process counts (in these experiments). Data for different node counts collected on different partitions.

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

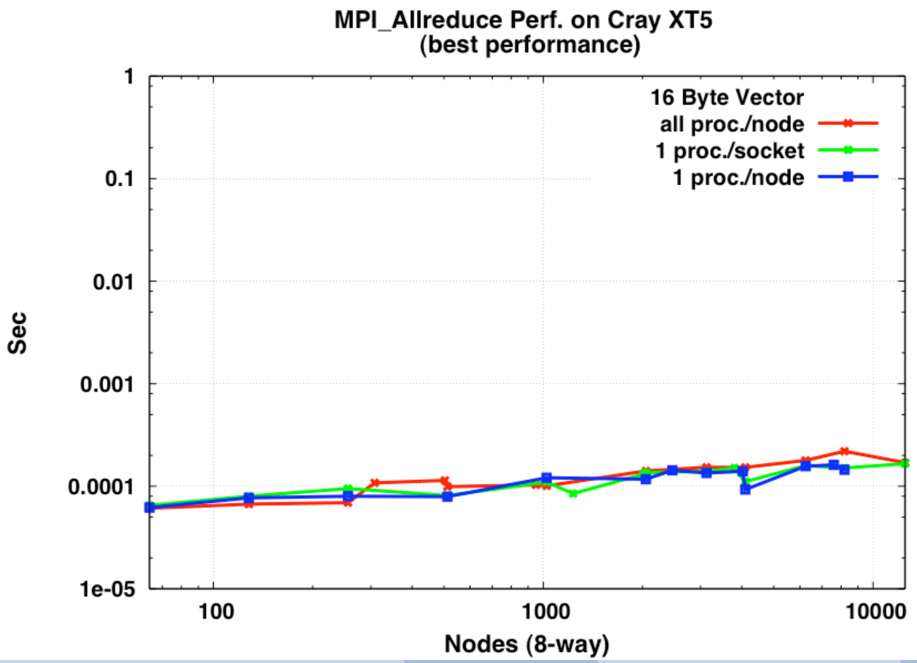
ALLREDUCE on Cray XT5

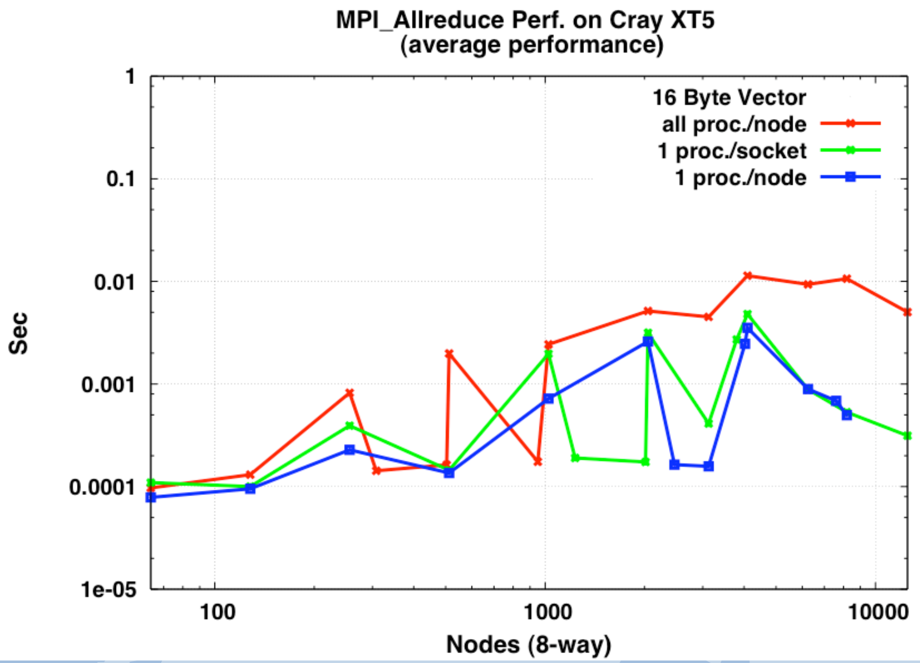


Measured performance of single MPI_Allreduce (global sum of 2 real*8 values) for 8 processes per node, 2 processes per node, and 1 process per node. Optimal performance is similar, but average performance is somewhat better when assigning fewer processes per node (fewer total number of processes for a fixed node count).

SciDAC
Scientific Discovery through Advanced Computing

ALLREDUCE on Cray XT5



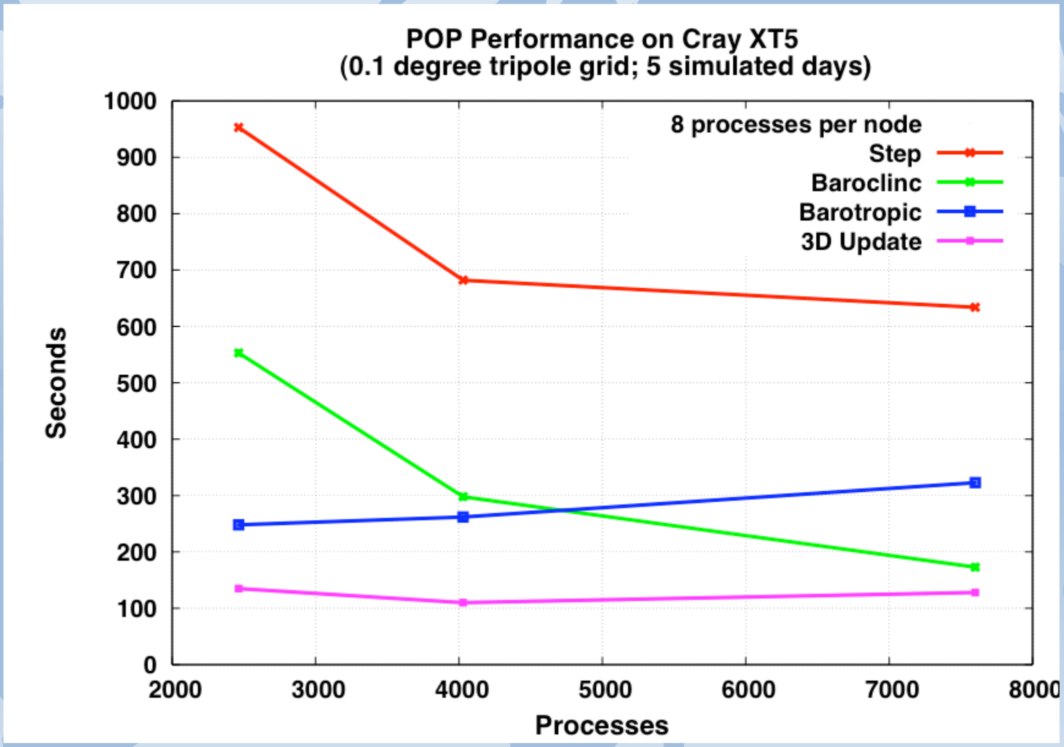


Best and average performance of single MPI_Allreduce (global sum of 2 real*8 values) for 8 processes per node, 2 processes per node, and 1 process per node. Optimal allreduce performance better than average halo update performance. Average alreduce performance is worse than halo update to small haloes when using 8 processes per node. Less obvious for other experiments.

OAK RIDGE National Laboratory

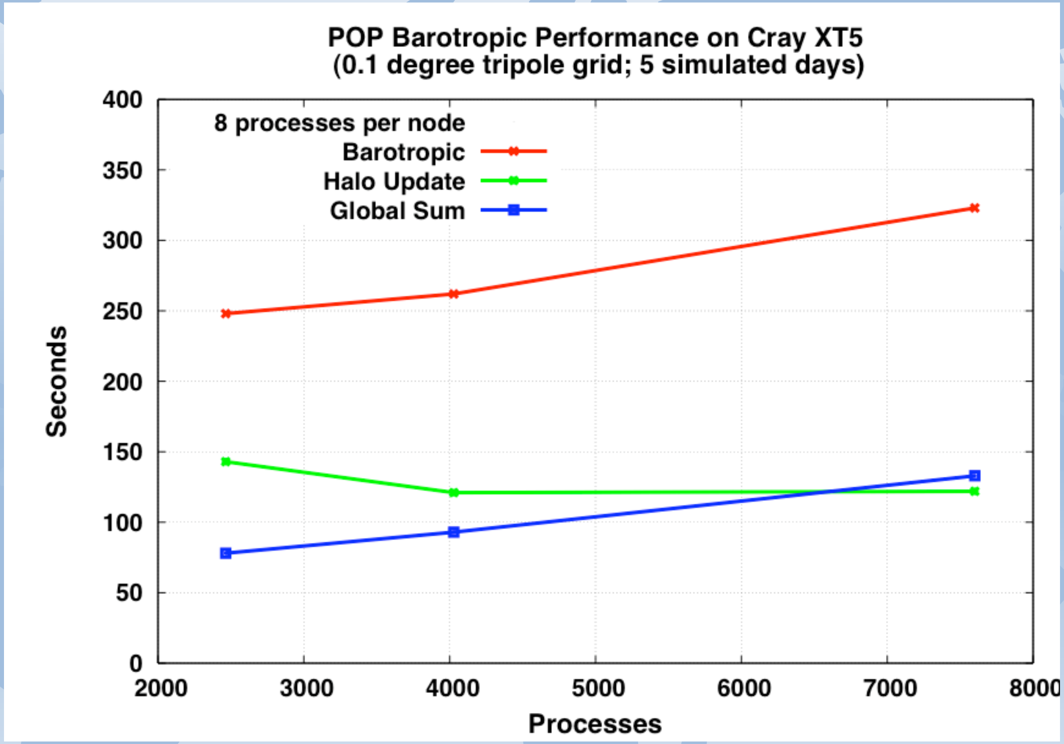
Office of Science U.S. DEPARTMENT OF ENERGY

POP on Cray XT5



Performance of CCSM4 version of POP for a 0.1 degree tripole grid with space-filling-curve decomposition. POP run concurrently with other CCSM components. Results reported for 5 simulated days for Baroclinic, Barotropic, and 3D Update phases of main timestepping loop (Step).

POP Barotropic on Cray XT5



Performance of Barotropic phase of POP, which is dominated by a conjugate gradient (Chronopoulos-Gear variant) solve. Performance of C-G solve is dominated by a halo update (part of residual calculation) and global sum (inner product). > 250,000 CG iterations reflected in these timings.

POP Barotropic Halo Update on Cray XT5

There is a structural load imbalance in the halo update in the Barotropic C-G solve:

- 2048 processes (8 processes per node):

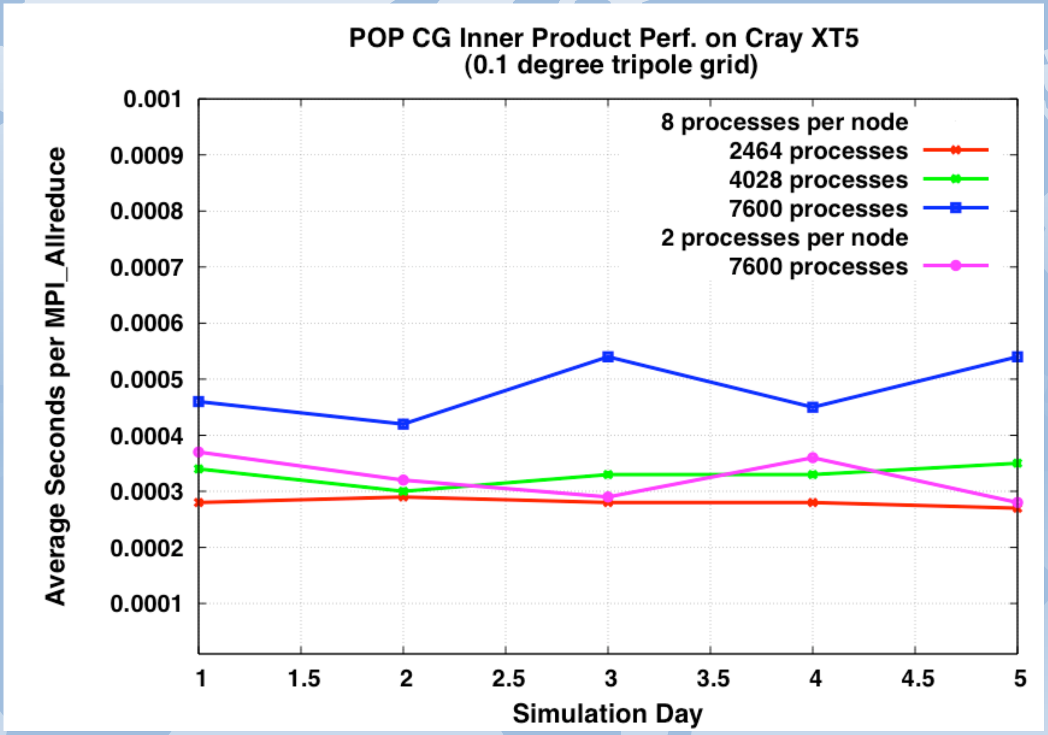
  108 idle processes; non-idle process timings: 12 to 128 seconds
- 4028 processes (8 processes per node):

  no idle processes; non-idle process timings: 9 to 109 seconds
- 7600 processes (8 processes per node):

  593 idle processes; non-idle process timings: 5 to 109 seconds
- 7600 processes (2 processes per node):

  593 idle processes; non-idle process timings: 18 to 50 seconds

POP Barotropic Global Sum on Cray XT5



Average performance of global sum per simulation day. A timing barrier was inserted before the global sum, to isolate it from load imbalance in immediately preceding halo update. Performance is improved if run with fewer processes per node (for the same total process count). Performance is not inconsistent with ALLREDUCE data.

POP Comments

1. Need kernel code to more accurately reflect POP halo update. Should look into further optimizing grid decomposition to minimize existing load imbalance.
2. Global sum performance is limiter at scale. Talking to Cray/NCCS staff to see if can be improved. (Scaling has been better in past, with simpler nodes and simpler operating systems.) Strong argument for OpenMP parallelism, and/or for looking for faster converging solver or one not so global sum sensitive.
3. Analysis very machine specific, but process is not. Also have (some) kernel data for BG/P, and am in process of repeating exercise on the BG/P system.

Summary of Suggestions

1. Consider performance portability and performance variability tolerance in design of algorithms to be used “at scale”.
2. Sketch out simple performance models and define representative kernels for evaluating costs associated with a given target architecture.
3. Determine under what conditions “your” approach is likely to perform poorly, and determine the likelihood of this (and whether you need to do something different).