Comparison of Cray XT3 and XT4 Scalability *

P. H. Worley †

Abstract

The Cray XT3 and XT4 have similar architectures, differing primarily in memory performance and in bandwidth between the node and interconnect. This paper evaluates and compares the scalability of the XT3 and XT4. Kernel benchmarks are used to verify and to quantify the performance differences between the systems. Application benchmarks are used to examine the impact of these differences on scalability. Both kernel and application benchmarks are used to identify how to use the systems most efficiently.

1 Introduction

The National Center for Computational Sciences at Oak Ridge National Laboratory (ORNL) is the site of an XT3 with 5212 compute nodes and an XT4 with 6296 compute nodes. For both systems the compute node contains a dual core AMD Opteron processor running at 2.6 GHz and 4 GB of memory. The interconnect for both systems is a custom, three-dimensional (3D) toroidal network utilizing the Cray SeaStar network interface controller (NIC) to connect the compute node, via HyperTransport, with the network.

Both systems use the Catamount light-weight kernel operating system on the compute nodes and a Linux-based operating system on the service and login nodes. Catamount supports two execution modes: SN (single/serial node) mode, in which only one core is active, and VN (virtual node) mode, in which both cores are active. In SN mode, the active core has full access to all of the memory in the compute node. In VN mode, the memory is partitioned and each core has access to half of the node memory. In VN mode, one core ("core 0") is responsible for NIC access, while the other core ("core 1") interrupts core 0 to handle internode communication on its behalf. Communication between user processes running on core 0 and core 1 (in the same node) is implemented with memory copies.

The XT3 and XT4 differ in two ways that affect performance. First, the XT3 compute node uses DDR-400 memory, while the XT4 compute node uses DDR2-667 memory, representing effective memory bandwidths to each core of 6.4 GB/s and 10.6 GB/s, respectively. Second, the XT3 uses the SeaStar NIC, while the XT4 uses the SeaStar2 NIC. The SeaStar2 increases the peak network injection bandwidth of each node from 2.2 GB/s to 4 GB/s when compared to SeaStar, and increases the sustained network performance from 4 GB/s to 6 GB/s.

In March 2007, the ORNL XT3 and XT4 systems were combined into a single (heterogeneous) system. The default is that jobs submitted to this system are assigned whatever nodes are available, possibly mixing XT3 and XT4 nodes. However, the user can request that jobs run on only XT3 nodes or on only XT4 nodes. Some of the data presented in this paper were collected on the XT3 and XT4 systems before the merger, but the majority were collected on the combined system using either XT3-only or XT4-only compute nodes and identical versions of the system software.

This paper examines the performance characteristics of the XT3 and of the XT4, focusing on the impact of the system differences on application scalability. Kernel benchmarks are used to verify and to quantify the system differences. Application codes drawn from the climate community are used to examine scalability on the two systems, as well as to identify the performance impact of SN vs. VN mode

*This research was sponsored by the Climate Change Research Division of the Office of Biological and Environmental Research and by the Office of Mathematical, Information, and Computational Sciences, both in the Office of Science, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Batelle, 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.

†Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (worleyph@ornl.gov)

and the performance impact of a number of runtime options.

Note that in the remainder of this paper we use the terms (processor) core and processor interchangeably, referring to each core as a processor.

2 Methodology

Data for most experiments were collected in April 2007 using version 6.1.6 of the Portland Group compilers and version 1.5.31 of the Cray Programming Environment. Kernel and application benchmarks use the Message Passing Interface (MPI) [9] for interprocessor communication. One kernel benchmark also measures communication performance when using SHMEM [8].

There are number of compile and runtime options that affect performance. The options considered most often in this paper are as follows.

1. Compiler flags. We examined compiler optimization options including the default, `-O2`, `-O3`, `-fast`, `-fastsse`, `-Mipa=fast`, and combinations of these. See the `pgf90` compiler documentation for descriptions of these flags [17].

2. Page size. The default page size is 2 MB. Specifying `-small_pages` when submitting a job changes the page size to 4 KB.

3. Mode. We examined performance for both SN and VN modes.

4. Process placement. We used the `MPICH_RANK_REORDER` environment variable to choose different algorithms for mapping logical processes to processors when running in VN mode. For example, assume that we want to run using 8 processors (on 4 nodes). The default option assigns processes 0 and 4 to node 0, processes 1 and 5 to node 1, etc. (*wrap placement*). Option 1 assigns processes 0 and 1 to node 0, processes 2 and 3 to node 1, etc. (*SMP-style placement*). Option 2 assigns processes 0 and 7 to node 0, processes 2 and 6 to node 1, etc. (*folded rank placement*). Option 3 allows the user to specify a custom assignment. In this paper we focus on the wrap and SMP-style placement options.

5. MPI collectives. We examined the impact of setting the `MPI_COLL_OPT_ON` environment variable. According to the MPI `man` page, this "enables collective optimizations using non-default, architecture specific algorithms for some MPI collective operations."

6. MPI memory copy. We examined the impact of setting the `MPICH_FAST_MEMCPY` environment variable. According to the MPI `man` page, this "enables an optimized memcpy routine in MPI."

3 Kernel Benchmarks

The following kernel benchmarks are used to determine the performance characteristics of system subsystems in ways easily interpreted in the context of application codes. We focus in particular on determining ways to use the systems most efficiently, and on determining whether the same best practices apply equally to the XT3 and to the XT4.

3.1 DGEMM

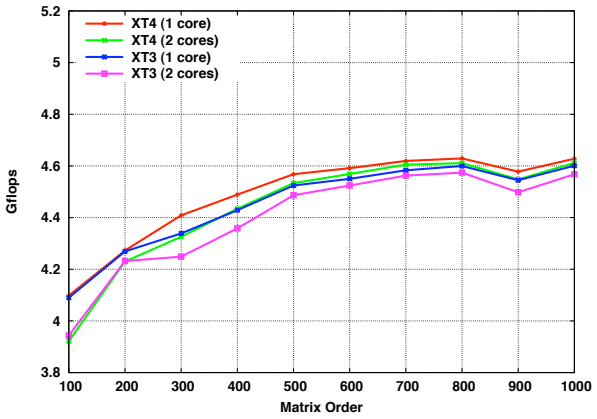

FIGURE 3.1: Matrix Multiply (DGEMM) Performance

Figure 3.1 describes the double-precision floating point performance of a matrix multiply using the DGEMM [7] routine from the Cray *libsci* library. Matrix multiply has a high ratio of floating point operations to operands and good register and cache locality, when implemented carefully. A DGEMM benchmark is often used to define the "achieveable peak performance" of a processor. As the XT3 and XT4 use the same processor, we would expect little difference between the performance on the two systems for this benchmark.

Figure 3.1 contains graphs for two experiments for each system, one executing the benchmark on a single core and the other executing two instances of the benchmark, simultaneously, one on each core.

The second experiment measures the performance impact of resource contention, if any, caused by running the experiment on both cores. For the two-core experiments, the lowest observed computation rate for each matrix size is reported.

There is little performance difference between the four DGEMM experiments, each achieving approximately 4.6 GFlop/s, or 88% of the 5.2 GFlop/s peak performance for the Opteron processor. What differences there are indicate a slight performance advantage for using the XT4 over the XT3 and for using only one core instead of both cores, both consistent with differences in memory performance.

Note that the optimal performance was achieved by using the default page size. Running with `-small_pages` degraded performance by approximately 1%.

3.2 PSTSWM

The Parallel Spectral Transform Shallow Water Model (PSTSWM) [21, 23] represents an important computational kernel in spectral global atmospheric models. PSTSWM exhibits little reuse of operands as it sweeps through the field arrays; thus it exercises the memory subsystem as the problem size is scaled and can be used to evaluate the impact of memory contention in multi-core nodes. PSTSWM is also a parallel algorithm testbed, and all array sizes and loop bounds are determined at runtime.

We began by examining the impact of compiler flags and page size on performance. For PSTSWM, compiling with `-fast` improved performance by a factor of 2 to 3 compared to the default optimization. All optimization levels at least as high as `-O2` achieved identical performance. Page size did not affect performance significantly. These results hold for both the XT3 and XT4, and for both SN and VN modes. For the experimental data described below we compiled with `-fast` and ran with `-small_pages`.

Figure 3.2 compares PSTSWM performance between the XT3 and the XT4. In the first graph comparisons are presented as computation rate versus horizontal resolution for a fixed number of vertical levels (18). The problem sizes T5, T10, T21, T42, and T85 are horizontal resolutions. Each computational grid in this sequence is approximately 4 times larger than its predecessor. The second graph in Fig. 3.2 compares the computation rate for a single horizontal resolution (T85) for a range of numbers of vertical levels. As with DGEMM, data from two experiments are examined: performance when using a single core and performance when using both cores to run independent instances of the PSTSWM benchmark simultaneously.

The difference in XT3 and XT4 memory performance has a clear impact on PSTSWM performance in these studies, as does contention for memory when using both cores. Note that the impact of memory contention is qualitatively the same for the XT3 and the XT4, for example, degrading performance for T85 with 88 levels by 27% and 25%, respectively.

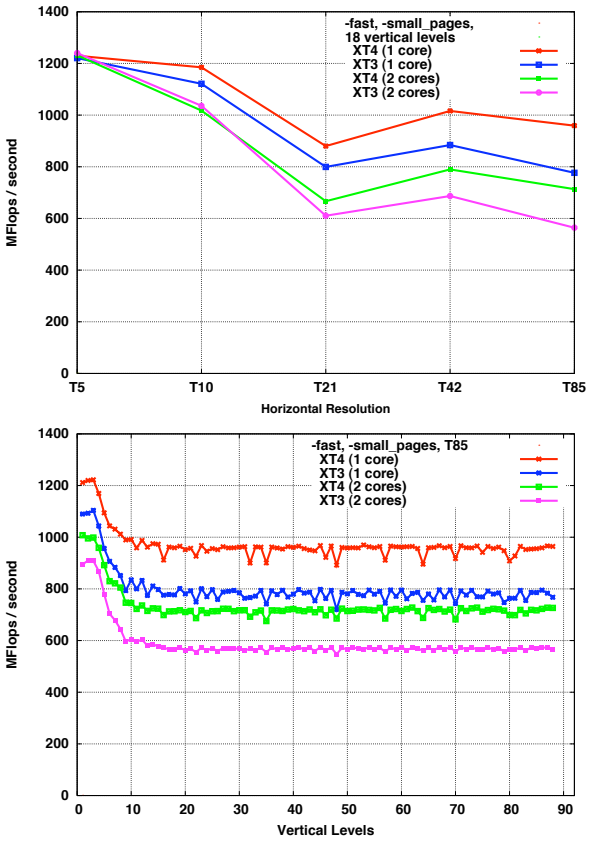

FIGURE 3.2: Serial Performance of PSTSWM

3.3 COMMTEST

COMMTEST is a suite of codes developed at ORNL that measure the performance of MPI interprocessor communication. COMMTEST differs somewhat from other MPI benchmark suites in its focus on determining the performance impact of communication protocol and packet size in the context of "common usage". However, the performance we report here should be similar to that measured using other interprocessor communication benchmarks.

Experiments were of two types: measuring communication performance between two processors and measuring communication performance between two

subsets of processors, where pairs of processors, one in one subset and one in the other, are communicating simultaneously. The benchmark measures both bidirectional performance, using a "ping-ping" communication pattern, and undirectional performance, using half the roundtrip time in a "ping-pong" communication pattern.

In the results described below, the benchmarks were compiled with `-fast` optimization and run with `-small_pages`. We also used SMP-style process placement, as this process placement was most compatible with the ordering assumed by the benchmark when evaluating the impact of contention in communication performance. The results reported are the optimal performance observed over all communication protocols.

We experimented with the setting the `MPICH_FAST_MEMCPY` environment variable, but it had no performance impact in these experiments.

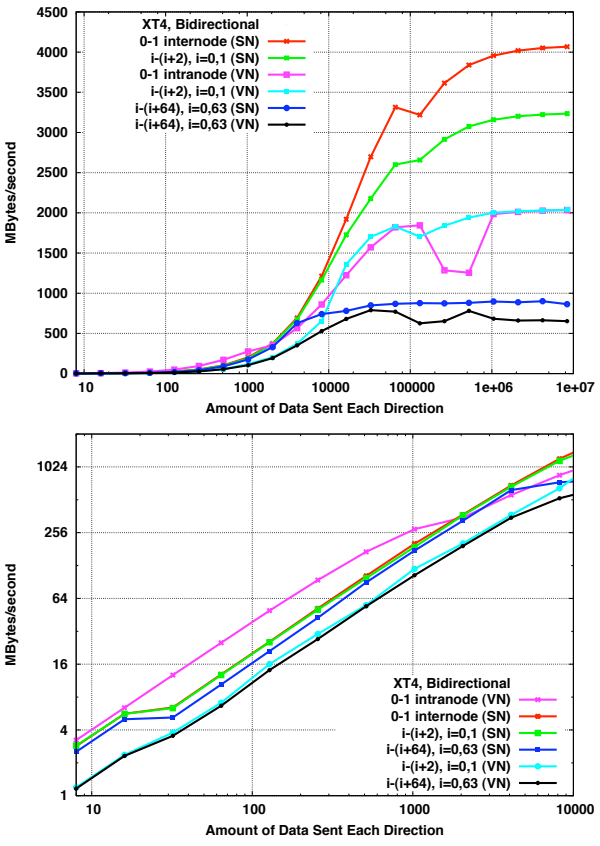

FIGURE 3.3: MPI Bidirectional Bandwidth on the XT4

Figure 3.3 describes bidirectional communication performance on the XT4 when communicating between two processors (0 and 1), between two subsets each with two processors (processor 0 communicating with processor 2 and processor 1 communicating with processor 3), and between two subsets each with 64 processors (processor i communicating with processor $(i + 64)$, $i = 0, \ldots, 63$). For experiments involving multiple pairs of processes, the lowest per pair bandwidth is reported. The allocation of the nodes used in these experiments was not controlled, but the default allocation algorithm attempts to allocate nodes that are "contiguous" in the network topology. Experiments were run in both SN and VN modes, and the 0-1 experiment for VN mode measured communication between processors (cores) within a node. The first graph in the figure uses log-linear axes, while the second displays the same data for message sizes up to 10,000 bytes using log-log axes. From these results, SN mode 0-1 internode communication achieves twice the performance of VN mode 0-1 intranode communication. Note that both processors in one node communicating with both processors in a neighboring node (also) achieves half the per pair bandwidth as the SN mode 0-1 intranode communication, which is equivalent to achieving the same aggregate bandwidth between the two nodes. The two-pair SN mode experiments demonstrate link contention, as do the 64-pair SN and VN mode experiments. In other experiments, not described here, the link contention can be shown to be a function of the 3D torus interconnect, but this behavior is not something that most users will be able to exploit when optimizing their codes.

The second graph in Figure 3.3 shows that purely intranode communication has the lowest latency, but that simultaneous communication with both processors in one node to both processors in a different node has twice the latency as when only one processor per node is communicating.

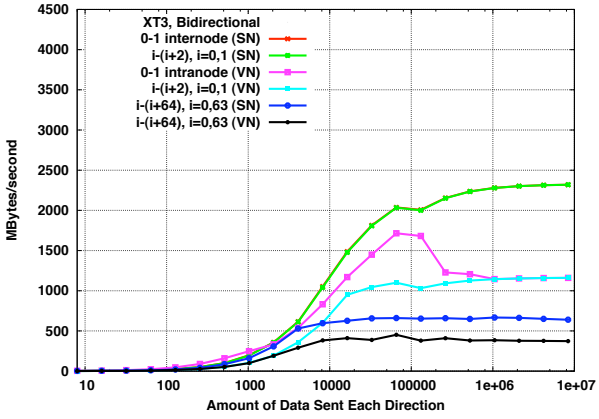

FIGURE 3.4: MPI Bidirectional Bandwidth on the XT3

Figure 3.4 shows XT3 data for the same experi-

ments. The small message size performance is very similar to that on the XT4, and the log-log plot is omitted. For large message sizes, the XT4 communication bandwidth is between 1.5 and 2 times greater than that on the XT3. Interestingly, the two-pair SN mode performance is identical to that of the single pair SN mode performance on the XT3, indicating that the SeaStar in the XT3 is not able to saturate the shared network link in this experiment, while the SeaStar2 in the XT4 can. Otherwise, the XT3 and XT4 experiments are similar qualitatively.

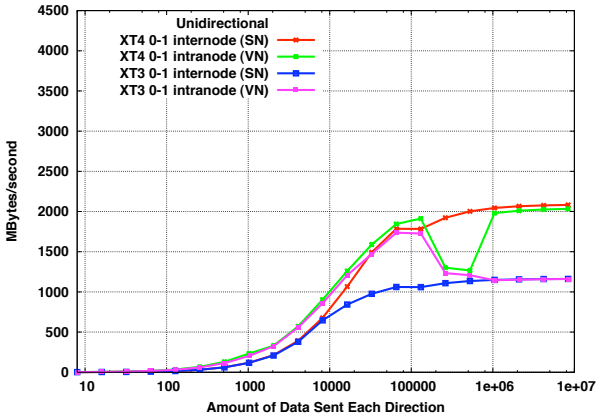

FIGURE 3.5: MPI Unidirectional Bandwidth on the XT3 and XT4

Figure 3.5 shows unidirectional communication performance for the single pair experiments on the XT3 and XT4 and for SN and VN modes. For large message sizes, the XT4 achieves twice the bandwidth of the XT3, for both intra- and internode communication. Internode unidirectional bandwidth is half that of the bidirectional bandwidth for all message sizes, while intranode bandwidth is unchanged (indicating that the memory copies used to implement intranode communication are not bidirectional, possibly because copies in both directions are initiated by core 0). For small message sizes, the XT3 and XT4 demonstrate similar unidirectional communication performance.

Data from the COMMTEST benchmarks also indicate the communication protocols that achieve the optimal performance. The following results hold for both the XT3 and the XT4.

- For a single pair (0-1):
  - For large message sizes, performance is relatively insensitive to choice of protocol.
  - For small message sizes, the best protocol is `MPI_Isend/MPI_Recv` for SN mode and `MPI_Sendrecv` for VN mode.
- For 64 pairs simultaneously:
  - For large message sizes, performance is optimized by preposting receives and using ready sends (for both SN and VN modes).
  - For small message sizes, `MPI_Sendrecv` is competitive for both SN and VN modes.

3.4 HALO

The HALO benchmark [18] simulates the nearest neighbor exchange of a 1-2 row/column "halo" from a two-dimensional (2D) array. This is a common operation when using domain decomposition to parallelize, for example, a finite difference ocean model. There are no actual 2D arrays used, but instead the copying of data from an array to a local buffer is simulated and this buffer is transferred between nodes. HALO is actually a suite of benchmarks, implementing the basic halo exchange operator utilizing a number of different messaging layers, and a number of different implementations for each layer.

We first use the HALO benchmark suite to examine which MPI-1 communication protocol is most efficient, and also to compare MPI-1 with SHMEM. The first graph in Figure 3.6 plots the results when using `-fast` compiler optimization and VN mode, `-small_pages`, and SMP-style process placement runtime options on 16 processors of the XT4. The x-axis indicates the number of 4 byte words in a single row or column in the halo. Here the focus is on small (latency dominated) halo exchanges, as data (not shown) indicates that achieved bandwidth for large halo exchanges is the same for all MPI protocols. Persistent MPI protocols are also tested by the HALO benchmark suite, and the performance for these protocols was identical to the nonpersistent equivalents. The data from running on the XT3 and XT4, from running in SN and VN modes, and from running with and without `-small_pages` are all qualitatively similar, with the `isend/irecv protocol` achieving the best performance in all cases. However, the performance comparison is somewhat different when using the (default) wrap process placement, as indicated in the second graph in Fig. 3.6 . Here `isend/irecv` is the worst MPI-1 protocol for a range of halo sizes, and performance is worse overall compared to using SMP-style process placement. The wrap placement results are also qualitatively the same for the XT3 and XT4, for SN and VN modes, and for large and small page sizes.

These data show that process placement can change performance characteristics, and it is important to take this into account when optimizing codes. These data also indicate that SHMEM performance in not competitive with MPI in this benchmark on the XT3 and XT4. Note that the HALO benchmark examines a number of different SHMEM implementations. We described only the best SHMEM results.

Figure 3.7 is a comparison of HALO exchange performance between the XT3 and XT4 and between SN and VM node modes when using the optimal MPI-1 protocol and SMP-style process placement. Small pages were used in these experiments, but the data is nearly identical to those collected using large pages. These data indicate that, for small halo exchanges, performance is identical on the XT3 and XT4. For large haloes, the improved memory and network bandwidth on the XT4 gives the XT4 a factor of 1.5 performance improvement over the XT3. SN mode performance is better than VN mode for all halo exchange sizes, by a factor of two for small haloes and a factor of 1.6 for the largest haloes, on both the XT3 and the XT4.

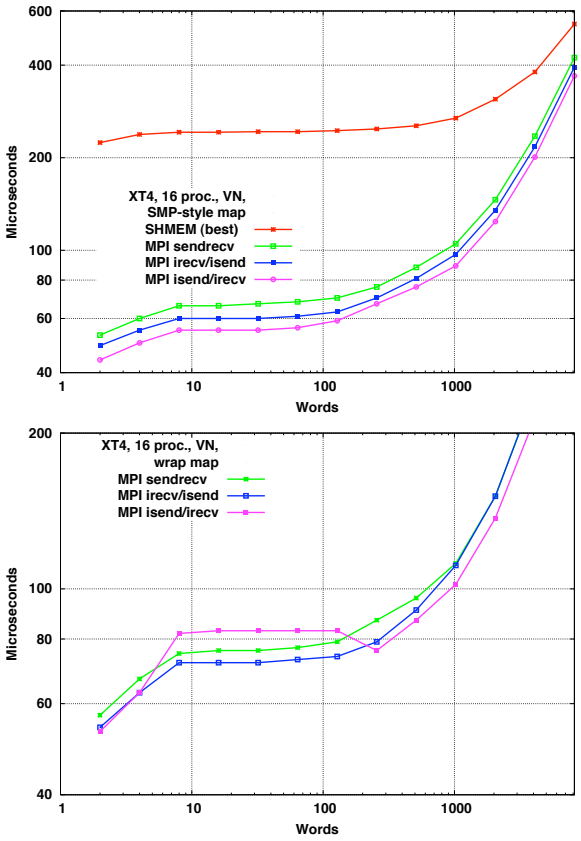

FIGURE 3.6: HALO Protocol Performance Comparison

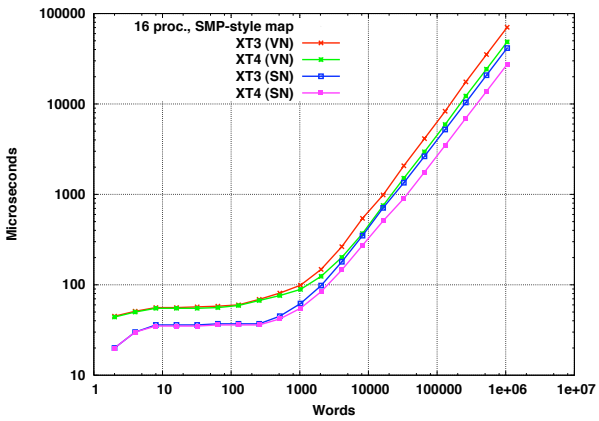

FIGURE 3.7: HALO Performance Comparison

3.5 Kernel Summary

Qualitatively, performance is very similar on the XT3 and the XT4, with identical optimal compile-time and runtime optimizations, and identical optimal interprocessor communication options. The improved memory bandwidth and network bandwidth on the XT4 are clearly evident in the data from the PSTSWM, COMMTEST, and HALO benchmarks.

4 Application Benchmarks

The following benchmarks are used to verify the kernel results in the context of full application codes. The kernel results are also used to help understand the application performance data.

4.1 CAM

The Community Atmosphere Model (CAM) is a global atmosphere circulation model developed at the National Science Foundation's National Center for Atmospheric Research with contributions from researchers funded by the Department of Energy and by the National Aeronautics and Space Administration [3, 4]. CAM is used in both weather and climate research. In particular, CAM serves as the atmospheric component of the Community Climate System Model (CCSM) [1, 5].

CAM is a mixed-mode parallel application code, using both MPI and OpenMP protocols [6]. CAM's performance is characterized by two phases: dynamics and physics. The dynamics phase advances the evolution equations for the atmospheric flow. The physics phase approximates subgrid phenomena, including precipitation processes, clouds, long- and

short-wave radiation, and turbulent mixing [4]. Control moves between the dynamics and the physics at least once during each model simulation timestep. The number and order of these transitions depend on the numerical algorithms used in the dynamics.

CAM includes three dynamical cores (dycores), one of which is selected at compile-time: a spectral Eulerian solver [12], a spectral semi-Lagrangian solver [19], and a finite volume semi-Lagrangian solver [13]. The following experiments describe results for both the spectral Eulerian and the finite volume dycores.

As a community model, it is important that CAM run efficiently on different architectures, and that it be easily ported to and optimized on new platforms. CAM contains a number of compile-time and runtime parameters that can be used to optimize performance for a given platform, problem or processor count [14, 15, 20]. The options examined in this paper are as follows.

- Physics load balancing. Grid points with a given horizontal location, differing only in the vertical coordinate, are referred to as a *column*. The current physical parameterizations in CAM are based on columns, and physics computations at a given timestep are independent between columns. The time required to process a column is a function of geographical location and simulation time, and excellent static load balancing schemes are known. However, the best load balancing scheme is at odds with the domain decompositions utilized by the dycores, thus requiring significant interprocessor communication to implement. Two options are considered here: no load balancing (and no interprocessor communication) and the optimal load balancing (requiring all processors to communicate with all other processors).

- Communication protocol for load balancing. The communication protocol used to implement the interprocessor communication required by the load balancing scheme is a runtime option. On the Cray XT systems suported options include `MPI_Alltoallv` and 19 different MPI two-sided point-to-point implementations.

- Communication protocols for spectral dynamics. Within each call of the spectral dynamics, computation moves back and forth between a longitude-latitude-vertical grid point space and the spectral coefficient space. The dependencies in the transforms between these two spaces require changing the decomposition from one-dimensional (1D) over latitude to 1D over longitude and back again. The communication protocols used to implement interprocess communication within the spectral dynamics are also runtime options, with the same choices available as for the physics load balancing.

For the spectral Eulerian dycore we used version 3.0p1 of CAM, available from

`http://www.ccsm.ucar.edu/models/atm-cam/`

and a benchmark problem with a longitude-latitude-vertical grid of size $256 \times 128 \times 26$ and a wavenumber-latitude-vertical grid of size $85 \times 128 \times 26$. This problem resolution is referred to as *T85L26*. This is the same problem resolution and CAM dycore as used in the CCSM for the fourth IPCC (Intergovernmental Panel on Climate Change) assessments [10]. Because the spectral dycore supports only a 1D decomposition over latitude, the number of MPI processes cannot be greater than 128 for this problem. However more than 128 processors can be used if OpenMP parallelism is exploited. As OpenMP is not supported on either the XT3 or XT4 currently, we are severely limited in the number of processors that we can use for this benchmark. To address this problem, we modified the code, as follows. In runs with the spectral dynamics, the time spent in the physics is typically twice that spent in the dynamics. The physics also supports an arbitrary 2D decomposition. To expose more parallelism, we modified version 3.0p1 to support using different numbers of processors in the dynamics and in the physics. For example, when allocating 256 processors, the dynamics would still use only 128 processors but the physics would be able to use all 256 processors. (This mimics the hybrid MPI/OpenMP implementation in that the dynamics exhibits little OpenMP parallelism after exploiting the full MPI parallelism.) This approach works only if physics load balancing is enabled, allowing work to be redistributed between the dynamics and physics phases. When deciding to run the dynamics on a subset of P processors, the question arises as to which P processors to use. Two options are examined here: (1) $\{0, \ldots, P-1\}$ (*stride 1*) and (2) $\{0, 2, \ldots, 2P-2\}$ (*stride 2*). When using the stride 2 option with SMP-style process placement and VN mode, only one processor will be active per node, essentially computing the dynamics in SN mode.

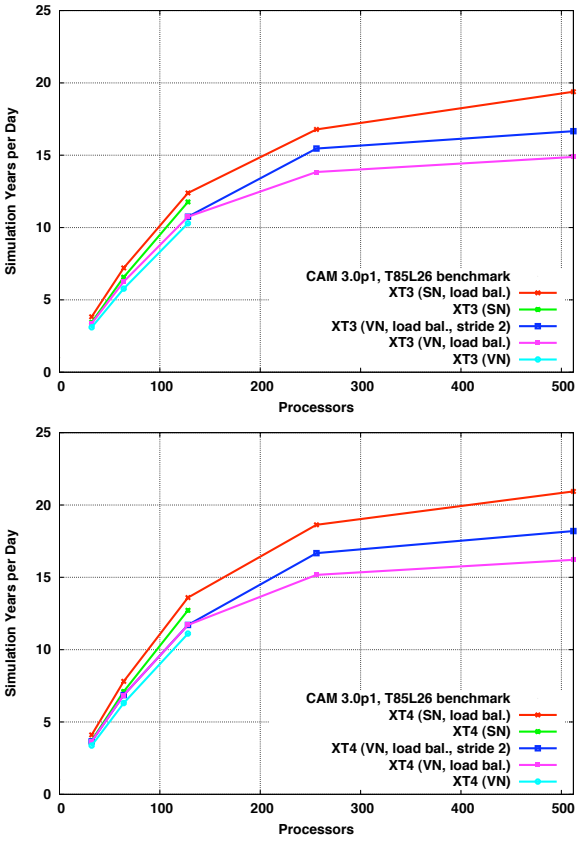

FIGURE 4.1: CAM Performance for T85L26 Benchmark

Figure 4.1 is a graph of the performance of the modified version of CAM 3.0p1 when compiled with `-fast` and run with `-small_pages` and SMP-style process placement. In these experiments, MPI collectives were used to implement the physics load balancing and dynamics domain decomposition remapping. These data demonstrate that increasing the number of processors assigned to the physics enables scaling beyond 128 processors, even though the dynamics is still limited to 128-way parallelism. Load balancing, even with the additional communication overhead, is a performance enhancement up to 128 processors (and is a requirement for using more than 128 processors). Using stride 2 dynamics processor subsetting improves performance by 12% when compared to using stride 1. Even with stride 2, SN mode is 10% to 15% faster than VN mode. Both results indicate that the memory contention and/or the MPI performance degradation observed in the kernel experiments affects CAM performance in VN mode. Note that the results are qualitatively the same on the XT3 and the XT4, though performance on the XT4 is superior.

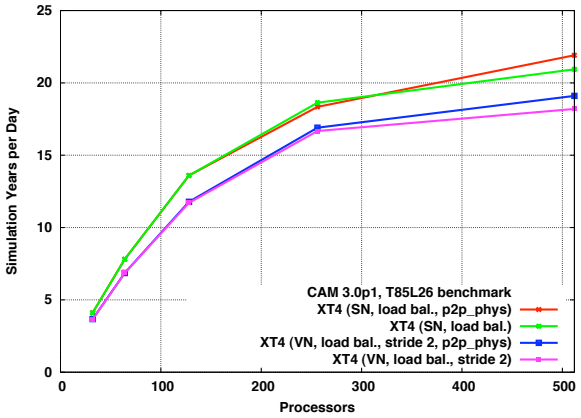

FIGURE 4.2: Performance Optimization for T85L26 Benchmark

In an attempt to further improve CAM performance, we experimented with replacing MPI collective calls with point-to-point implementations. Figure 4.2 is a graph of the performance when replacing just the `MPI_Alltoallv` used in the physics load balancing communication with a point-to-point implementation based on an `MPI_Isend/MPI_Recv` communication protocol. (Other protocols were slower in these experiments.) The point-to-point implementation is faster than the MPI collective implementation for the largest processor count, and is as fast for all of the other processor counts. Note that the collectives in the dynamics never involve more than 128 processors, which is probably why it was not advantageous to replace the collectives within the dynamics with point-to-point implementations. Though not shown, similar results were observed in experiments on the XT3.

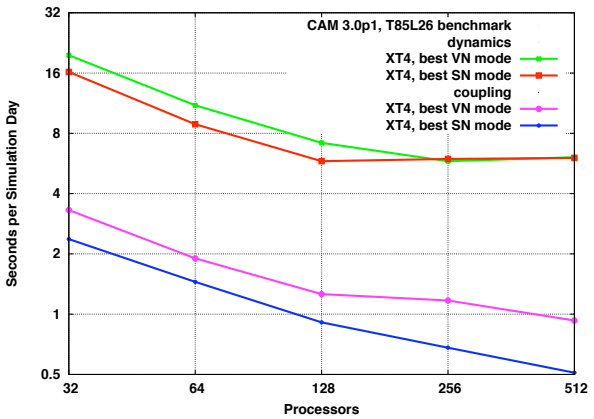

FIGURE 4.3: Phase Analysis for T85L26 Benchmark

Figure 4.3 is a phase analysis, comparing the cost of the dynamics phase and the cost of the dynamics-physics coupling for the best performing SN and VN

mode experiments on the XT4. Note that the cost of the dynamics-physics coupling is dominated by the cost of the interprocessor communication used to implement physics load balancing. Here, the dynamics cost is constant when using more than 128 processors and is identical for SN and VN modes, as hoped. The coupling cost is still decreasing for SN mode, but appears to have reached a lower bound for VN mode, reflecting the usual MPI performance degradation observed in VN mode. The point-to-point implementation was used in the coupling phase when using 512 processors. When using the MPI collective implementation, the cost of the coupling phase is 50% greater for 512 processors than for 256 processors for both SN and VN modes. A similar analysis holds for experiments on the XT3.

To benchmark the finite volume dycore we used version 3.1 of CAM, available from the same location as version 3.0p1. Our benchmark problem uses a 361×576 horizontal computational grid with 26 vertical levels. This resolution is referred to as the "D-grid". While the D-grid resolution is greater than that used in current computational climate experiments, it represents a resolution of interest for future experiments. The finite volume dycore supports both a 1D latitude decomposition and a 2D decomposition of the computational grid. The 2D decomposition is over latitude and longitude during one phase of the dynamics and over latitude and vertical in another phase, requiring two remaps of the domain decomposition each timestep. For small processor counts the 1D decomposition is faster than the 2D decomposition, but the 1D decomposition must have at least three latitudes per MPI task and, so, is limited to a maximum of 120 MPI tasks for the D-grid benchmark. Using a 2D decomposition requires at least three latitudes and three vertical layers per MPI task, so is limited to 960 MPI tasks for the D-grid benchmark. As with the spectral dycores, OpenMP can be used to exploit multiple processors per MPI task, on systems that support OpenMP. Unlike for the spectral dycores, the dynamics for the finite volume dycore is typically twice as expensive as the physics. Thus there is little to be gained by using more processors in the physics than in the dynamics, and we have not used the modification described earlier in the D-grid benchmark experiments.

Figure 4.4 is a graph of CAM performance for the D-grid benchmark. Here CAM was compiled with `-fast` and run with `-small_pages`, SMP-style process placement, and physics load balancing. All experiments were also run using large pages, but small pages improved performance by a small amount. MPI collectives were used in the physics load balancing. Interprocessor communication within the finite volume dynamics is handled by a dycore-specific messaging layer utilizing point-to-point MPI commands, and the optimal settings were used in the results described here.

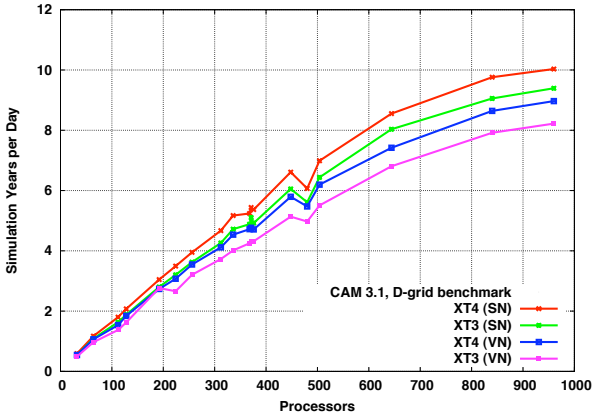

FIGURE 4.4: CAM Performance for D-grid Benchmark

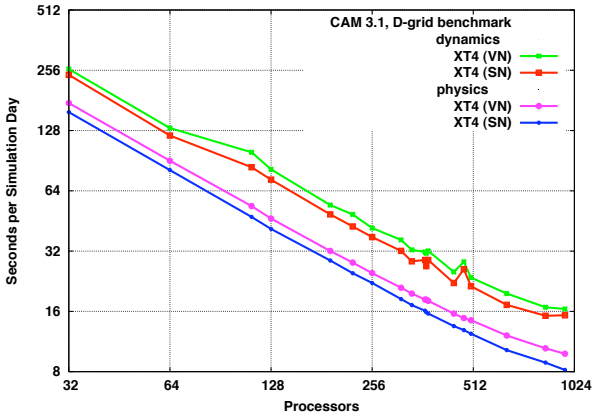

FIGURE 4.5: Phase Analysis for D-grid Benchmark

As with the other benchmarks, performance on the XT3 and XT4 are qualitatively identical. SN mode is faster than VN mode for the same number of processors, and the runs on the XT4 are faster than runs on the XT3. Performance is continuing to scale out to the limit of the algorithmic parallelism. Figure 4.5 is a phase analysis for the XT4 experiments. (Results for the XT3 are similar.) Here physics is continuing to scale, but the dynamics phase appears to have reached a lower bound. Note that in this phase analysis, the dynamics-physics coupling is included in the physics phase, and comprises 13% (SN) and 17% (VN) of the physics cost for the maximum number of processors. Unlike the T85L26 benchmark results, the cost of this coupling is not increas-

ing for the largest processor counts, and the MPI collective implementation is scaling well. In future work, we will examine in more detail the performance of the communication protocols used in the finite volume dynamics, and experiment with point-to-point implementations of the dynamics-physics coupling, but nothing in the current benchmark data indicates that these types of optimization will improve performance significantly for this benchmark.

4.2 POP

The Parallel Ocean Program (POP) [16, 11] is a global ocean circulation model developed and maintained at Los Alamos National Laboratory (LANL). It is used for high resolution studies and as the ocean component in the CCSM. The code is based on a finite-difference formulation of the 3D flow equations on a shifted polar grid. POP performance is characterized by the performance of a baroclinic phase and a barotropic phase. The 3D baroclinic phase scales well on all platforms due to its limited nearest-neighbor communication. In contrast, the barotropic phase is dominated by the solution of a 2D, implicit system, whose performance is very sensitive to network latency and typically scales poorly on all platforms. For our evaluation we used version 1.4.3 of POP with a few additional parallel algorithm tuning options (due to Yoshida) [22]. The current production version of POP is version 2.0.1. While version 1.4.3 and version 2.0.1 have similar performance characteristics, the intent here is to use version 1.4.3 to evaluate system performance characteristics, not to evaluate the performance of POP.

We consider results for the one tenth degree benchmark problem. This is a displaced-pole grid with the pole of the grid shifted into Greenland to avoid computations near these singular points. The grid resolution is 0.1 degree (10km) around the equator, increasing to 2.5km near the poles, utilizing a 3600×2400 horizontal grid and 40 vertical levels. This resolution resolves eddies for effective heat transport and is used for ocean-only or ocean and sea ice experiments.

We examined a number of different compile and runtime options in our initial optimizations. For the following experiments we specified `-Kieee -O3 -fastsse -tp k8-64` when compiling and `-small_pages` and SMP-style process placement when running the following experiments. None of these settings were strong optimums, and experiments using `-fast`, default process placement and large pages demonstrated comparable performance. We also experimented with defining `MPI_COLL_OPT_ON` and `MPICH_FAST_MEMCPY`, but observed no improvement in performance.

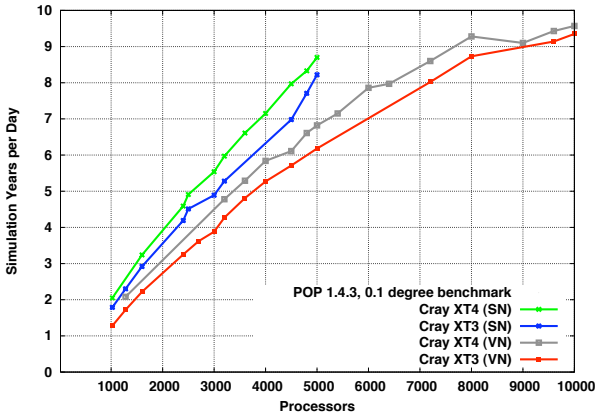

FIGURE 4.6: POP Performance Comparison

Figure 4.6 compares the performance of POP on the XT3 and XT4 and when running in SN and VN modes. SN mode is significantly faster than VN mode for the same number of processors. However, running in VN mode is faster than in SN mode for the same number of compute nodes, which is the relevant metric since POP's performance scalability allows us to use the entire system for this benchmark problem. Running on the XT4 is also faster than running on the XT3, though the advantage is small for large processor counts.

Figure 4.7 compares the performance of the POP baroclinic and barotropic phases on the XT3 and XT4 and when running in SN and VN modes. As expected, the baroclinic phase scales very well. The XT4 is faster than the XT3 for this phase, and SN mode is faster than VN mode, both explained by the faster computational rate enabled by the improved memory performance of the XT4 and of SN mode. In contrast, the cost of the barotropic phase is at best constant for VN mode, and approaching that for SN mode. This phase is also subject to performance perturbations. SN mode is much faster than VN mode for the barotropic phase, and POP runtime in VN mode is dominated by the barotropic runtime when using more than 5000 processors. The XT4 demonstrates a slight performance advantage over the XT3, on average, for the barotropic phase.

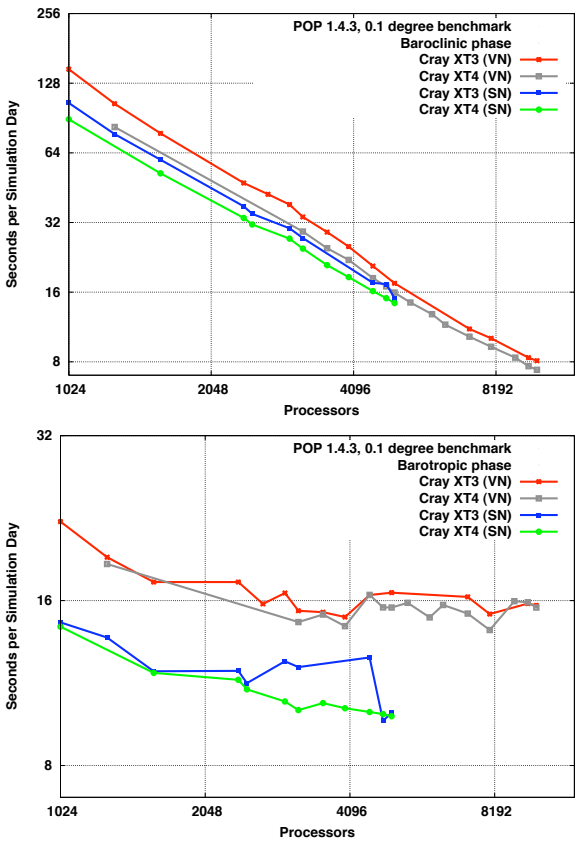

FIGURE 4.7: POP Baroclinic and Barotropic Performance Comparison

Results from both the COMMTEST and HALO microkernels indicate much lower internode latency in SN mode than in VN mode. As performance of the barotropic phase is dominated typically by latency-sensitive calls to `MPI_Allreduce`, this performance characteristic appears to explain the significant advantage of SN mode over VN mode for this phase. We used this insight to develop the following optimization for when running in VN-mode:

- During initialization, create a subcommunicator made up of all core 0 processors.
- Replace the call to `MPI_Allreduce` in the barotropic phase with the following.
  - Core 1 sends its local sum to core 0 on the same node. Core 0 adds the contribution from core 1 to its local sum.
  - All of the core 0 processors call `MPI_Allreduce`, using the "core 0" subcommunicator.
  - Core 0 sends the results to the core 1 on the same node.

Using this algorithm, the `MPI_Allreduce` in the barotropic phase is partially run in SN mode, while the rest of the code runs in VN mode. Performance of the modified algorithm is shown in Fig. 4.8. Figure 4.9 compares the performance of the barotropic phase of the modified and unmodified algorithms. While not identical to SN mode performance, it is a significant improvement.

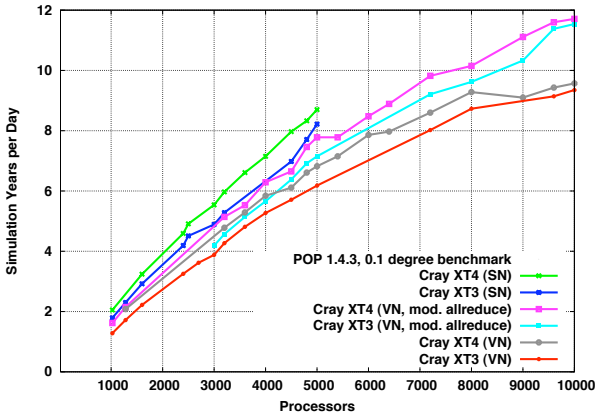

FIGURE 4.8: Performance of POP Using Modified Algorithm

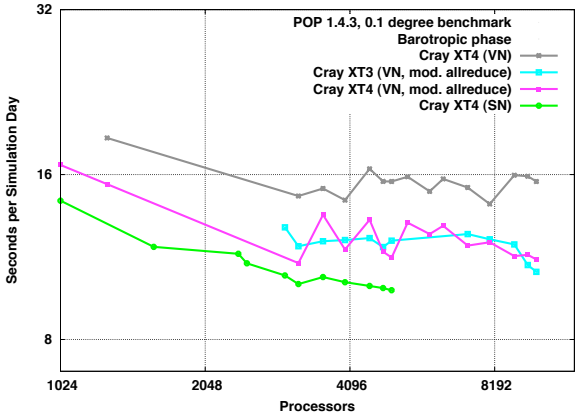

FIGURE 4.9: Performance of Barotropic Phase Using Modified Algorithm

Version 2.0.1 of POP has options not available in version 1.4.3. One of these is the ability to run the barotropic phase on a subset of the processors. While not supported currently, version 2.0.1 could be modified to allow that subset to be all core 0 processors, eliminating the overhead of the additional communication between core 0 and core 1 before every call to `MPI_Allreduce` in the current modified algorithm. A new overhead will be the cost of the mapping to the barotropic decomposition, but this happens only once per barotropic call and should be

cheaper than the current approach. Another optimization is the Chronopoulos-Gear (C-G) variant [2] of the Conjugate Gradient algorithm. C-G uses half the number of inner products as the standard algorithm, thus calling `MPI_Allreduce` half as many times during the barotropic phase. We backported this algorithm into version 1.4.3. Combining the C-G algorithm with the modified `MPI_Allreduce` algorithm, we were able to increase POP version 1.4.3 throughput by a factor of 1.6, as shown in Figure 4.10. Note that results for 10,000 processors for SN mode and 22,500 processors for VN mode include both XT3 and XT4 compute nodes.

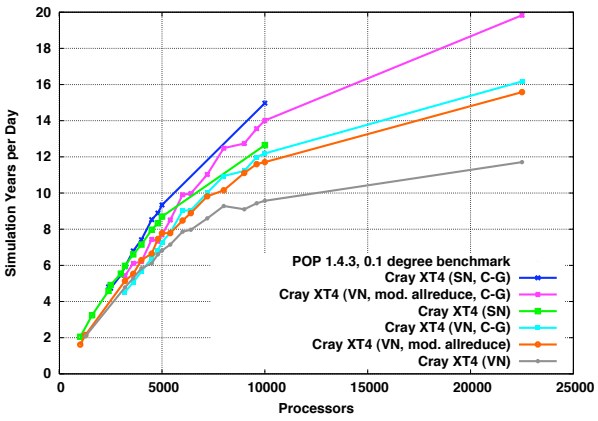

FIGURE 4.10: Performance Comparison of POP Variants

4.3 Application Benchmark Summary

Using the results of the kernel benchmark experiments, we were able to interpret and optimize the performance of both the CAM and POP benchmarks. In particular, understanding the impact of VN mode on MPI latency led to the development of SN mode equivalent algorithms for the spectral dynamics in CAM and for the `MPI_Allreduce` algorithm as used in the barotropic phase in POP.

5 Conclusions

Kernel benchmarking provided the following insights into XT3 and XT4 performance.

1. Data from the DGEMM benchmark experiments showed the expected agreement between the XT3 and the XT4, due to their using the same processor. The small performance differences are explained by the differences in memory performance between the XT3 and XT4 and when using one or both cores.

2. Data from the PSTSWM benchmark experiments demonstrated that the performance impact due to differences in memory performance (XT3 vs. XT4 and single core vs. dual core) can be significant for realistic computational kernels.

3. Data from COMMTEST benchmark experiments quantified the performance improvement from replacing the SeaStar NIC with the SeaStar2. The maximum bidirectional bandwidth in the two-pair VN mode experiments was exactly half that for the single pair SN mode experiment, indicating that the same maximum internode bandwidth was being achieved in both modes. The VN mode latency for the same experiment was double that for SN mode, a performance degradation that was also observed in the application benchmarking. The communication protocol evaluations provided guidance in the application benchmark optimizations.

4. Data from the HALO benchmark experiments provided additional information on the performance sensitivities of the different MPI communication protocols. These experiments also demonstrated the performance sensitivity of process placement in VN mode, both in achieveable performance and in determining optimal communication protocols.

These results guided the benchmarking, analysis, and optimization of the application benchmarks. In particular, based on our understanding of the performance impact of communication protocol, SN vs. VN mode, and process placement, we were able to improve significantly the scalability of two of the application benchmark problems. The other benchmark problem already demonstrated reasonable scalability to the limit of its algorithmic parallelism. Overall, performance scalability on the XT3 and XT4 is good, if the indicated sources of performance degradation are avoided. For all benchmarks, XT3 and XT4 performance characteristics were qualitatively identical, with good optimization strategies on one system also being good on the other.

The data presented in this paper did not indicate any advantage to using compiler optimization levels higher than `-fast`, nor any advantage to defining the environment variables

`MPI_COLL_OPT_ON` and `MPICH_FAST_MEMCPY`. Also, running with `-small_pages` did not affect performance significantly (positively or negatively). However, these results are application-specific, and could easily change if using different versions of the compilers and programming environment. In other experiments, not reported here, defining `MPI_COLL_OPT_ON` improved performance of an `MPI_Allreduce` when using an inefficient choice of process placement, effectively making the performance of the collective insensitive to the `MPICH_RANK_REORDER` setting. Our experience with process placement showed an advantage to SMP-style placement, if only because it simplified our algorithm development. In another benchmark not presented here, one using the PETSc library to solve a linear system, SMP-style process placement degraded performance significantly compared to using wrap placement. In summary, developers and users should examine (and re-examine) these optimization options in the context of their own codes and benchmark problems.

6 Acknowledgements

This research used resources (Cray XT3 and Cray XT4) of the National Center for Computational Sciences at 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.

7 About the Author

Patrick H. Worley is a senior R&D staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory. His research interests include parallel algorithm design and implementation (especially as applied to atmospheric and ocean simulation models) and the performance evaluation of parallel applications and computer systems. Worley has a PhD in computer science from Stanford University. He is a member of the Association for Computing Machinery and the Society for Industrial and Applied Mathematics.
E-mail: worleyph@ornl.gov.

References

[1] M. B. Blackmon, B. Boville, F. Bryan, R. Dickinson, P. Gent, J. Kiehl, R. Moritz, D. Randall, J. Shukla, S. Solomon, G. Bonan, S. Doney, I. Fung, J. Hack, E. Hunke, and J. Hurrel, *The Community Climate System Model*, BAMS, 82 (2001), pp. 2357–2376.

[2] A. Chronopoulos and C. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

[3] W. D. Collins, P. J. Rasch, B. A. Boville, J. J. Hack, J. R. McCaa, D. L. Williamson, B. P. Briegleb, C. M. Bitz, S.-J. Lin, and M. Zhang, *The Formulation and Atmospheric Simulation of the Community Atmosphere Model: CAM3*, Journal of Climate, 19(11) (2006).

[4] W. D. Collins, P. J. Rasch, and et. al., *Description of the NCAR Community Atmosphere Model (CAM 3.0)*, NCAR Tech Note NCAR/TN-464+STR, National Center for Atmospheric Research, Boulder, CO 80307, 2004.

[5] Community Climate System Model. http://www.ccsm.ucar.edu/.

[6] L. Dagum and R. Menon, *OpenMP: an industry-standard API for shared-memory programming*, IEEE Computational Science & Engineering, 5 (1998), pp. 46–55.

[7] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.

[8] K. Feind, *Shared Memory Access (SHMEM) Routines*, in CUG 1995 Spring Proceedings, R. Winget and K. Winget, ed., Eagen, MN, 1995, Cray User Group, Inc., pp. 303–308.

[9] W. Gropp, M. Snir, B. Nitzberg, and E. Lusk, *MPI: The Complete Reference*, MIT Press, Boston, 1998. second edition.

[10] Intergovernmental Panel on Climate Change. http://www.ipcc.ch/.

[11] P. W. Jones, P. H. Worley, Y. Yoshida, J. B. White III, and J. Levesque, *Practical performance portability in the Parallel Ocean Program (POP)*, Concurrency and Computation: Practice and Experience, 17 (2005), pp. 1317–1327.

[12] J. T. Kiehl, J. J. Hack, G. Bonan, B. A. Boville, D. L. Williamson, and P. J. Rasch, *The National Center for Atmospheric Research Community Climate Model: CCM3*, J. Climate, 11 (1998), pp. 1131–1149.

[13] S.-J. Lin, *A 'vertically Lagrangian' finite-volume dynamical core for global models*, Mon. Wea. Rev., 132 (2004), pp. 2293–2307.

[14] A. Mirin and W. B. Sawyer, *A scalable implemenation of a finite-volume dynamical core in the Community Atmosphere Model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 203–212.

[15] W. Putman, S. J. Lin, and B. Shen, *Cross-platform performance of a portable communication module and the NASA finite volume general circulation model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 213–224.

[16] R. D. Smith, J. K. Dukowicz, and R. C. Malone, *Parallel ocean general circulation modeling*, Phys. D, 60 (1992), pp. 38–61.

[17] The Portland Group, *PGI High-Performance Compilers and Tools*. http://www.pgroup.com.

[18] A. J. Wallcraft, *SPMD OpenMP vs MPI for Ocean Models*, in Proceedings of the First European Workshop on OpenMP, Lund, Sweden, 1999, Lund University. http://www.it.lth.se/ewomp99.

[19] D. L. Williamson and J. G. Olson, *Climate simulations with a semi-lagrangian version of the NCAR Community Climate Model*, Mon. Wea. Rev., 122 (1994), pp. 1594–1610.

[20] P. H. Worley and J. B. Drake, *Performance portability in the physical parameterizations of the Community Atmosphere Model*, International Journal of High Performance Computing Applications, 19 (2005), pp. 187–202.

[21] P. H. Worley and I. T. Foster, *PSTSWM: a parallel algorithm testbed and benchmark code for spectral general circulation models*, Tech. Rep. ORNL/TM–12393, Oak Ridge National Laboratory, Oak Ridge, TN, (in preparation).

[22] P. H. Worley and J. Levesque, *The performance evolution of the Parallel Ocean Program on the Cray X1*, in Proceedings of the 46th Cray User Group Conference, May 17-21, 2004, R. Winget and K. Winget, ed., Eagen, MN, 2004, Cray User Group, Inc.

[23] P. H. Worley and B. Toonen, *A users' guide to PSTSWM*, Tech. Rep. ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.