Early Evaluation of the Cray XT5 *

P. H. Worley †
R. F. Barrett ‡
J. A. Kuehn §

Abstract

A Cray XT5 system has recently been installed at Oak Ridge National Laboratory (ORNL). This system differs from the existing XT4 system at ORNL in its compute node architecture, utilizing two quad-core Opteron Barcelona processors instead of a single quad-core Opteron Budapest processor. It also differs in its sheer scale, having approximately 150,000 processor cores, almost 5 times as many as in the XT4.

We present performance data for the Cray XT5, comparing with data from the Cray XT4 and other high performance computing systems. The focus is on single node computational benchmarks, basic MPI performance, both within and between nodes, and the impact of scale on MPI performance.

1 Introduction

A Cray XT5 system was recently installed in the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL). This system includes 18,722 compute nodes connected in a custom, three-dimensional torus (25x32x24). Each compute node contains 8 processor cores and 16 GB of memory, for an aggregate of 149,776 processor cores and 299,552 TB of memory. Each compute node contains two 2.3 GHz quad-core Opteron processors (AMD 2356 "Barcelona") linked with dual HyperTransport connections and DDR2-800 memory. While memory access performance is "non-uniform" (NUMA) at a node level, uniform memory access can be enforced by restricting thread placement and associated memory to a single socket in the dual socket node architecture. The system interconnect utilizes the 6-port Cray SeaStar2+ network interface controller. Each SeaStar2+ port is capable of 9.6 GB/s peak bidirectional bandwidth and 6 GB/s sustained. The SeaStar2+ is connected to the processor via HyperTransport

The XT5 system is an augmentation to an existing Cray XT4 system, called *Jaguar*. The XT4 system differs from the XT5 in its compute node architecture, utilizing a single 2.1 GHz quad-core Opteron processor (AMD 1354 "Budapest") and 8 GB of memory. It also differs in scale, having "only" 7,832 compute nodes, or a total of 31,328 processor cores and 62,656 TB of memory. Both the XT4 and XT5 systems currently run version 2.1 of the Cray Linux Environment (CLE), Cray's version of the SuSE Linux operating system.

This paper examines performance characteristics of the XT5, contrasting these with characteristics of the XT4 and other high performance computing systems such as the IBM BlueGene/P. The focus is primarily on single node performance and on MPI interprocess communication performance. The results are descriptive and quantitative, and currently we do not have a good explanation for all of the data that we observe. Hopefully these data will aid application developers in porting and optimizing their

*This research was 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-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)

‡Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5700, Oak Ridge, TN 37831-6164 (rbarrett@ornl.gov)

§Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5700, Oak Ridge, TN 37831-6164 (kuehn@ornl.gov)

codes, setting some expectations as to the performance they might expect. We also hope that some of the performance anomalies described in the paper will be diagnosed and addressed by the vendor or by NCCS staff.

2 Methodology

This paper is the most recent in a series of early evaluation papers [14, 10, 24, 11, 12, 22, 2, 1]. Early evaluations are "targeted" evaluations, with caveats. We look at subsystem performance using microkernel and kernel benchmarks, then attempt to validate these results with application benchmarks. Evaluations typically focus on the expected workload for the target system, and benchmarks are chosen to reflect both the workload and the special features of the system under investigation. The defining characteristic of an early evaluation is the relatively short duration of the data collection and the often unstable nature of the hardware and software of the system during the data collection. While runtime and compile-time optimizations are examined carefully, we are not able to modify the codes as part of the evaluation typically. For the XT5 evaluation, this latter point is especially important. Enabling the application codes to utilize the full system is requiring many developers to re-examine the choice of algorithms and implementations [19], and the codes that the authors are most familiar with are not yet ready to be used for performance evaluations of the XT5.

Data for most experiments were collected in October, 2008 on a 60 node XT5 development system, February, 2009 on the full XT5 systems, and in late April, 2009, verifying and augmenting the data collected in February.

3 Single Node Benchmarks

The following kernel benchmarks are used to determine the computational characteristics of a single compute node. We are in particular interested in how the dual socket node architecture affects performance.

3.1 DGEMM

Figure 3.1 describes the double-precision floating point performance of a matrix multiply using the DGEMM [8] 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. Data are described for compute nodes from three different systems: the current XT5 and XT4 systems at ORNL and an earlier XT4 system with 2.6 GHz dual-core opteron processors and a slower memory subsystem.

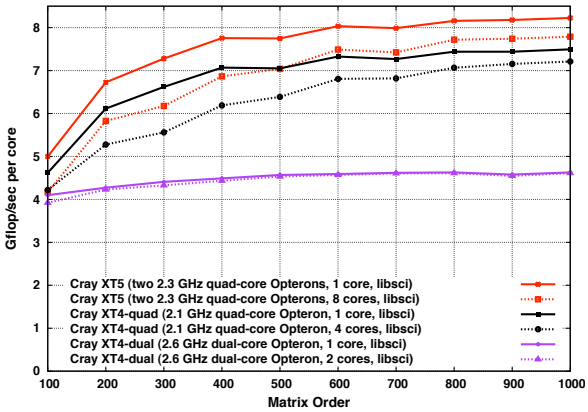

FIGURE 3.1: Matrix Multiply (DGEMM) Performance

Figure 3.1 contains graphs for two experiments for each system, one executing the benchmark on a single core and the other executing multiple 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 all cores. For the multiple-core experiments, the lowest observed computation rate for each matrix size is reported.

Data from the current XT4 and XT5 systems are qualitatively identical, with the quantitative differences arising from the difference in processor clock rate. Contention decreases performance from 89% of peak to 85% for the largest measured matrix size. In contrast, the dual-core XT4 node suffers little degradation from contention, but achieves significantly less performance. This is still 88% of the peak performance for this older Opteron processor.

3.2 HPCC Benchmarks

The next three benchmarks are drawn from the High Performance Computing Challenge benchmark suite [9, 17]: FFT, Random Access, and STREAM. DGEMM is also an HPCC benchmark, but the version of the DGEMM benchmark and the experimental methodology employed in the previous section are distinct from HPCC.

1. The FFT benchmark measures the performance of computing a double precision complex one-dimensional Discrete Fourier Transform. It is based on the FFTE package developed by D. Takahashi [21].

2. The STREAM benchmark is a synthetic benchmark that measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernel. STREAM was developed by J. McCalpin [18].

3. The RandomAccess benchmark measures the rate of integer random updates of memory. It was intially developed by D. Koester and R. Lucas [16].

Table 1 describes the ratio of performance on the XT5 to that on the XT4 for each of these benchmarks. Performance was measured for a single processor core, for a single core when all cores are computing, for a single quad-core processor, and for a single node. Note that in the node comparisons the XT5 has twice as many active processor cores as does the XT4.

FFT exhibits a reasonable temporal locality in that intermediate results remain in registers or cache during the computation. For this benchmark, the XT5 node exhibits a moderate performance improvement over the XT4 node, slightly more than the difference in processor clock rate. In contrast, STREAM is characterized by a high level of spatial locality in its memory reference pattern, but little temporal locality. For STREAM, the XT5 demonstrates a small penalty from increased memory contention, probably due to memory controller/channel limitations. RandomAccess performance is sensitive to memory latency, and the XT5 demonstrates an advantage over the XT4 for this benchmark similar to that for the FFT benchmark.

Table 2 describes the degradation in per core performance when running on all cores as compared to running on one core. This degradation decreased on the XT5 as compared to the XT5 for the benchmarks that were not memory bandwidth limited. Benchmarks (and applications) like STREAM that require high memory bandwidth will exhibit decreased per core performance when all cores are utilized.

3.3 POP

The Parallel Ocean Program (POP) [20, 15] 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 Community Climate System Model (CCSM) [3, 6]. Here we use version 1.4.3 and a problem with a computational grid of size $320 \times 384 \times 40$ to examine the performance impact of memory contention.

Parallelization in POP is based on domain decomposition and uses the Message Passing Interface (MPI) [13] for interprocess communication. Production runs for this problem size would typically use between 64 and 1024 MPI processes, depending on processor and MPI communication performance. On eight or fewer processes on current high performance computing (HPC) systems, interprocess communication overhead is negligible. In contrast, memory performance is important due to the large memory requirements when running on this small number of processes and due to a relatively low ratio of floating point operations to loads and stores.

Figure 3.2 describes POP performance on the Cray XT5 in terms of simulated years per day of computation (SYPD), a normalized inverse time metric. Performance is compared between running with 1, 2, 4, 8, and 64 processes when using 1, 2, and 4 cores per quad-core processor. We also looked at performance for one process per node. As this is very similar to performance when using one process per quad-core processor, we will not comment on it further.

When increasing the process count from one to two, and placing the two processes on separate quad-core processors (in the same node), performance nearly doubles. In contrast, when both processes are assigned to the same quad-core processor, speed-up is only 1.6. When using 4 processes, the speed-up relative to one process performance is 4.1, 3.4, and 2 when using 1, 2, and 4 cores per quad-core processor, respectively. Note that the speed-up on 4 cores was 2.3 last October. Thus performance degraded by approximately 15% for this experiment between October 2008 and February 2009. Data collected in April is the similar to that collected in Feburary. Performance for the other experiments (1 and 2 processes per quad-core processor) did not change between October, February, and April. Note the superlinear speed-up when using 4 processes and one process per quad-core processor, reflecting the improved cache utilization as the process count increases for this fixed size problem.

For eight processes, the speed-up compared to one process performance is 8.4, 7.4, and 4.4, respectively, with an October 2008 speed-up for 4 processes per processor of 5.0. For 64 processes, using all cores in 8 nodes, the speed-up is 42.3 and 45.2 for February

	Core Performance (1 core 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

Table 1: HPC Challenge Benchmarks: Ratio of XT5 to baseline XT4

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

Table 2: HPC Challenge Benchmarks: MultiCore Performance Degradation, Rate(multi)/Rate(single)

2009 and October 2008, respectively. The improved agreement between the October and February data appears to be due to the steadily decreasing memory requirements per process, as communication overhead is still very small in these experiments. Note that using all of the cores in a node is much faster than when using fewer for a fixed number of nodes, even with the performance degradation due to memory contention. We also ran the POP experiments with large page support enabled (not shown here), and it did not change POP performance.

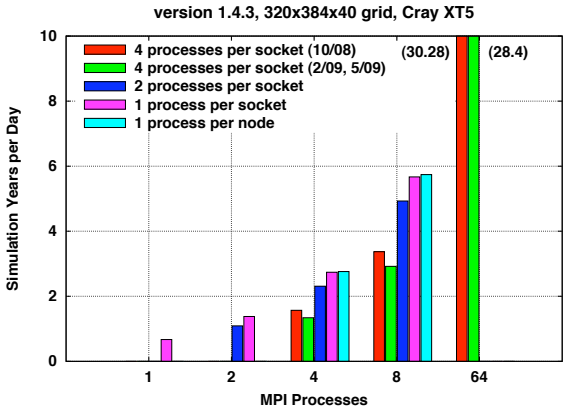

FIGURE 3.2: Parallel Ocean Program Performance on the XT5

Figure 3.3 compares POP performance on Cray XT4 and Cray XT5 nodes. For a single process, performance on the XT5 is nearly the same as that on the XT4. However, when using 4 processes and all cores in a quad-core processor, performance on the XT5 in February 2009 and October 2008 was 1.15 and 1.3 times greater that on the XT4, respectively, both of which are greater than the difference in the clock speed. Similar comparisons hold for eight processes.

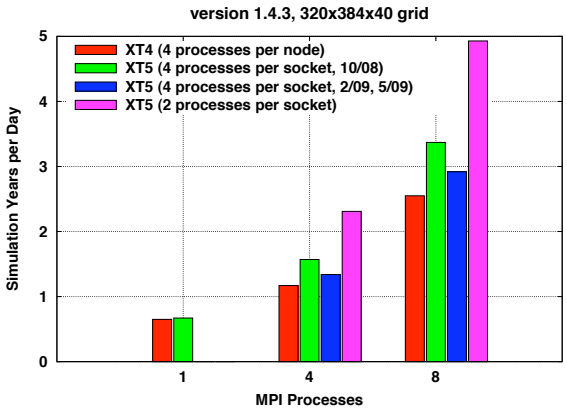

FIGURE 3.3: Parallel Ocean Program Performance

3.4 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 [4, 5]. CAM is used in both weather and climate research. In particular, CAM serves as the atmospheric component of the CCSM. Here we use version 3.1p2 with the finite volume dynamics solver on a computational grid of size 96×144.

CAM is a mixed-mode parallel application code, using both MPI and OpenMP protocols [7]. As in POP, MPI parallelization in CAM is based on do-

main decomposition. For the most part, OpenMP parallelism is used to further refine the domain decompostion. Here we use CAM in the same we used POP in the previous section, and examine performance when using MPI parallelism only. Production runs for this problem size would typically use between 32 and 256 MPI processes, but as many as 1024 processor cores when also utilizing OpenMP parallelism.

Figure 3.4 describes CAM performance on the Cray XT5 in terms of SYPD. Performance is compared between the XT4 and the XT5 for 4 and 8 MPI processes, and for 1, 2, and 4 processes per quad-core processor for the XT5. As with POP, MPI communication overhead is negligible for this combination of problem size and number of processes. In these experiments performance on the XT5 is 1.38 times faster than that on the XT4 on a single quad-core processor, and 1.36 times faster on two quad-core processors. Using the same number of nodes (but only two cores per processor on the XT5) increases the advantage to 1.44 and 1.45, respectively. So, unlike the POP experiments, there is less evidence of memory contention degrading performance. For this process count, the runtime is dominated by the computation of the physical processes, which is more compute intensive than the computation in POP. We also ran experiments with OpenMP (not shown here), keeping the total number of threads (processor cores) fixed. For this level of parallelism, OpenMP did not improve CAM performance.

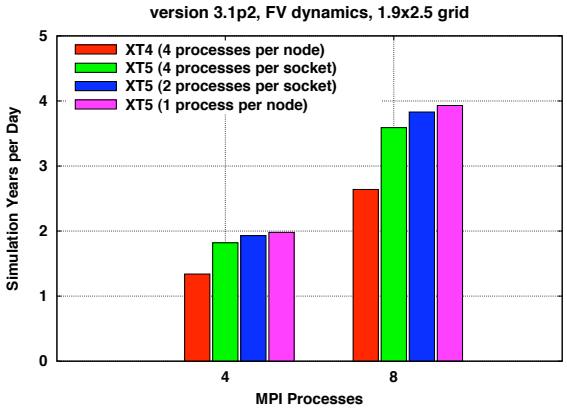

FIGURE 3.4: Community Atmosphere Model (CAM) Performance

3.5 Single Node Performance Summary

The DGEMM benchmark represents a "sanity" check, and performance on the XT5 was as expected. FFT, RandomAccess, and Stream demonstrate decreased performance advantage of the XT5 node over the XT4 as contention for (main) memory increases. For POP and CAM, XT5 per node performance is better than XT4 per node performance no matter how we measured it. However, what appears to be memory contention can degrade performance, especially for POP, in agreement with the HPCC results. POP performance when using all cores in a quad-core processor has degraded by 15% since October 2008. 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.

4 Communication Benchmarks

MPI is the most common mechanism used for internode communication by applications running on the Cray XT systems at ORNL. When not using OpenMP parallelism, MPI is also the most common mechanism used when communicating between processes within the compute node. As such, understanding MPI performance characteristics is an important first step in analyzing and optimizing application code performance.

4.1 Point-to-Point

Our first MPI benchmark measures bandwidth rates for MPI point-to-point commands. Data were collected for two types of experiments: measuring communication performance between two processes and measuring communication performance between two subsets of processes, where pairs of processes, 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 following, we discuss bidirectional bandwidth only.

The benchmark also measures performance for 20 different implementations of the exchange operator using MPI two-sided commands. The only omission of consequence is that MPI persistent commands are not included among these implementations. In the following we describe the optimal performance observed for a given message size over all examined implementations.

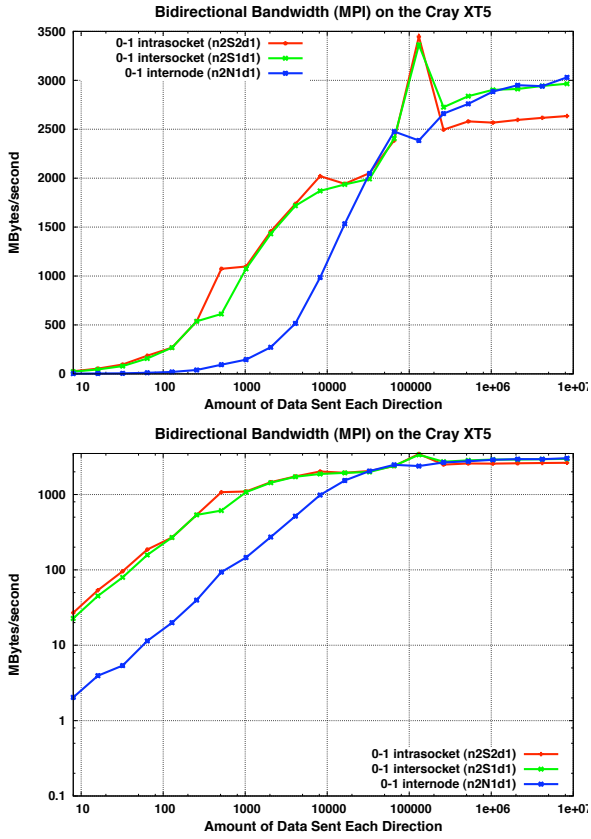

FIGURE 4.1: Single pair exchange experiments (log-linear and log-log)

Figure 4.1 describes performance when a single process pair exchange data. Measurements include performance between two processes in the same quad-core processor, two processes in different processors but in the same node, and processes in different, but neighboring, nodes. The top graph is a log-linear plot, while the bottom is a log-log plot, both of the same data.

Figure 4.2 describes performance for a single process pair when multiple process pairs exchange data. Experiments include:

1. two process pairs in the same quad-core processor;
2. four process pairs in the same node such that each core in one quad-core processor communicates with a distinct core in the other processor;
3. two process pairs such that one core in each quad-core processor communicates with the analogous core in the neighboring node;
4. eight process pairs such that each core in one node communicates with the analogous core in the neighboring node;
5. sixteen process pairs such that each core in two neighboing nodes n and $n+1$ communicate with the analogous cores in neighboring nodes $n+2$ and $n+3$;
6. thirty-two process pairs such that cores in the four nodes $n, \ldots, n+3$ communicate with cores in the four nodes $n+4, \ldots, n+7$;
7. sixty-four process pairs such that cores in the four nodes $n, \ldots, n+7$ communicate with cores in the four nodes $n+8, \ldots, n+15$.

For up to 4 nodes (16 process pairs), the nodes were consecutive in a single dimension of the torus (4×1). For 8 and 16 nodes, the physical topology was 4×2 and 4×4, respectively. We experimented with different orderings of the nodes, and this made no difference in the observed performance.

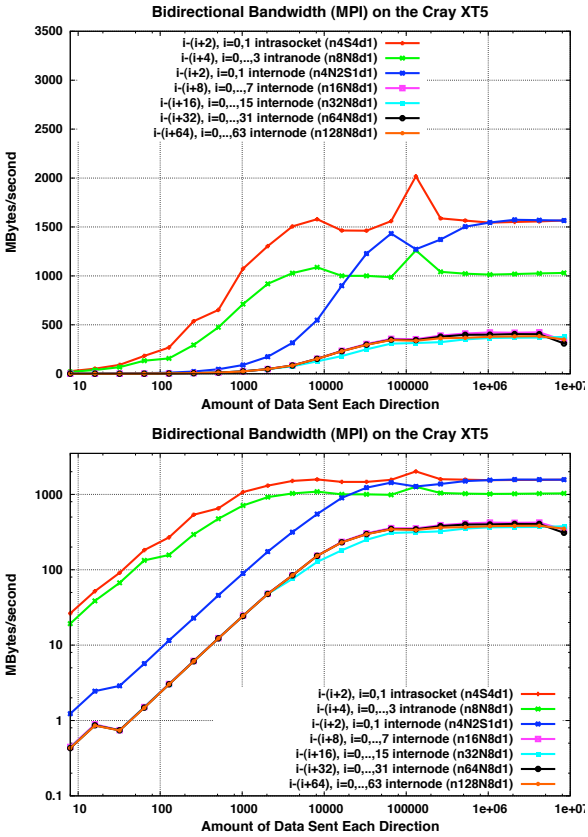

FIGURE 4.2: Single pair performance for multiple pair experiments (log-linear and log-log)

From these data, performance for a single process pair is the same for the two intranode experiments up to messages of size 128 KBytes. For large

message sizes, a higher bidirectional rate is achieved when communicating between the quad-core processors than within a quad-core processor. Internode performance is lower than intranode performance for messages of size 128 KB and smaller, representing an approximately 10 times higher latency between nodes than within nodes. For the largest message sizes, internode performance is equal to or better than intranode performance.

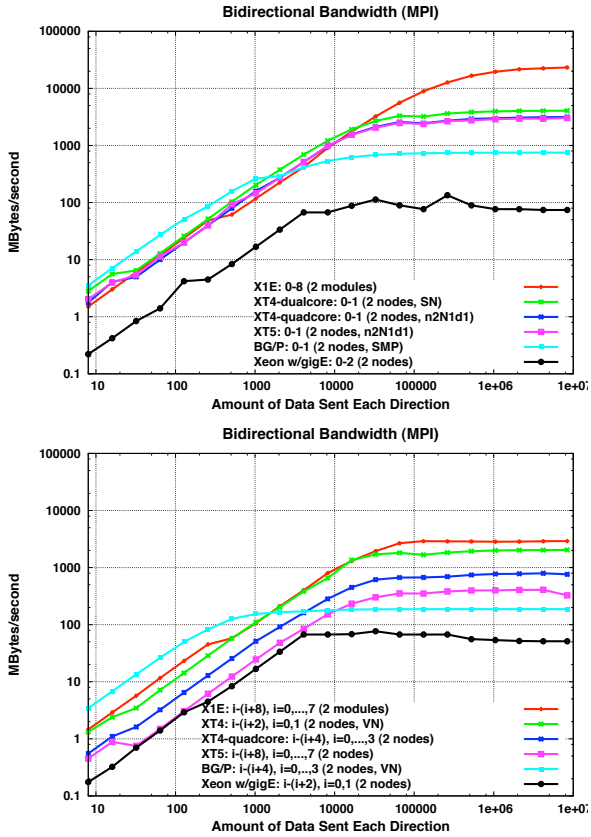

FIGURE 4.3: Point-to-point performance: Platform comparisons (log-log)

For the simultaneous exchange experiments, achieved bandwidth within a quad-core processor when two pairs are communicating is approximately 50% higher than the bandwidth between quad-core processors when 4 pairs are communicating. This result holds for the entire range of message sizes. For the largest messages sizes, performance for two process pairs is the same for internode and intranode experiments. For smaller message sizes, intranode performance is again approximately 10 times better.

For the simultaneous exchange experiments, one pair, two pairs and eight pairs achieve the same total internode bandwidth, with the available bandwidth being divided equally between the different process pairs. For more simultaneous pairs, involving more than two compute nodes, performance is identical to that in the eight pair experiments, indicating that these experiments were not able to saturate links in the network. Internode latency of single pair is (also) half that of two pair, but is 1/5 that of 8 pair (and not 1/8). Latency is (also) not affected by number of nodes communicating, in these experiments.

Figure 4.3 compares internode performance between different platforms for a single process pair when a single pair is communicating and when all processor (cores) in one node communicate with all of the processor (cores) in a neighboring node. For the largest message sizes performance on the Cray X1E is significantly better than on the other systems; the advantage is smaller for the simultaneous exchange experiments. The dual-core XT4 running the Catamount operating system achieves the next best performance. For a single process pair, the quad-core XT4 and the XT5 achieve the same performance. For the simultaneous exchange experiments, the quad-core XT4 and the XT5 demonstrate the same total internode bandwidth for almost all message sizes, resulting in the performance observed by a single pair of processes on the XT5 being half that observed on the quad-core XT4. For small message sizes, the best performance is observed on the IBM BG/P.

4.2 Barrier

Figure 4.4 describes performance of the MPI_Barrier command on the Cray XT5. The benchmark measures the performance of MPI_Barrier 10,000 times. The top graph describes the best observed and average barrier performance over these 10,000 experiments for a range of node counts, and for both one process per node and 8 processes per node. The bottom graph contains the same data as well as the worst case performance. Note that these data were not collected on a dedicated system.

Looking at the best performance, there is a small preference for power-of-two process counts. There is also very little performance difference between using one process per node and all processes per node for the same number of nodes. Average performance is approximately half that of the best performance. The worst case is much worse (up to 1000 times worse). Worst case performance is worse when using 8 processes per node than 1 process per node, but experiments with 1 process per node can still suffer significant performance degradation. The ratio of worst case to best case appears to increase with the

node count, but very large worst case performance also can occur for small process counts.

Figure 4.5 compares average MPI_Barrier performance for one process per node between the XT5, a dual-core XT4 running the Catamount operating system, and an IBM BG/P. For barriers over MPI_COMM_WORLD, the BG/P can use a hardware barrier, which is much faster than not using it on the BG/P and much faster than what is observed on the XT4 and XT5 systems. The average MPI_Barrier runtime on the Cray XT5 is approximately 1.46 times greater than that on the dual-core XT4.

Figure 4.6 compares minimum, average, and maximum MPI_Barrier performance for one process per node between the XT5 and the dual-core XT4 running the Catamount operating system. Minimum runtimes are comparable on the XT4 and XT5 systems. It is the maximum times that differentiate MPI_Barrier performance on the two systems.

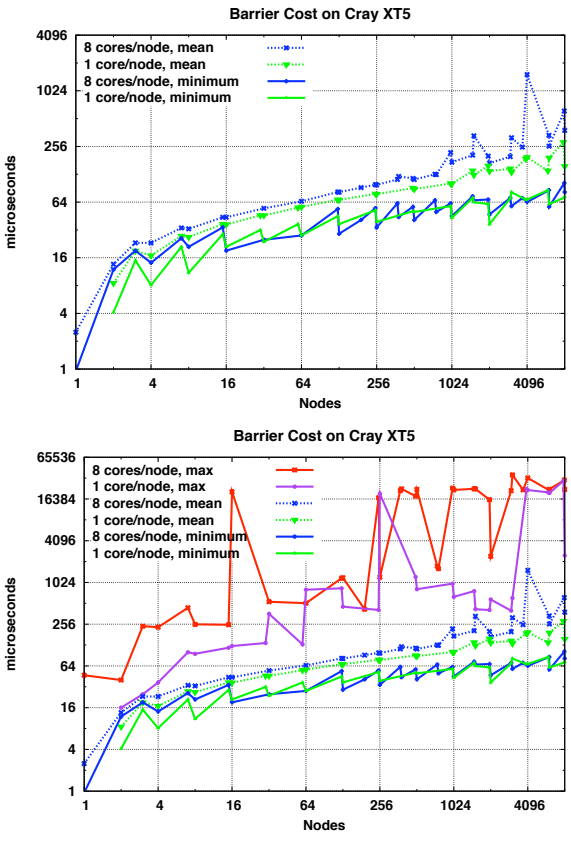

FIGURE 4.4: MPI_Barrier performance on the Cray XT5

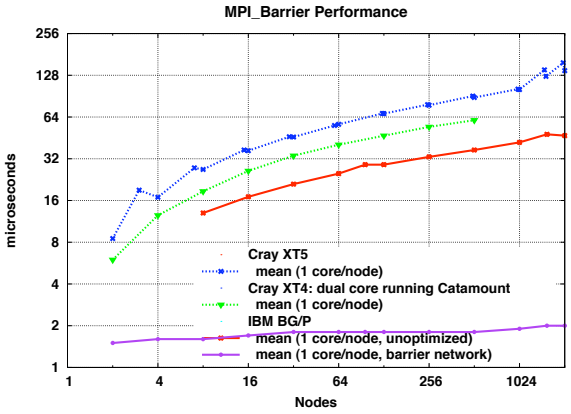

FIGURE 4.5: MPI_Barrier performance: Platform Comparison

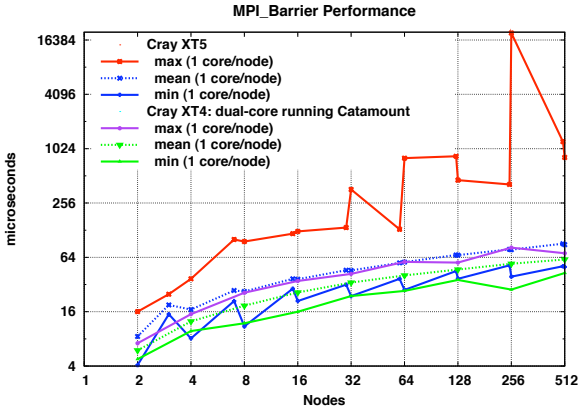

FIGURE 4.6: MPI_Barrier performance: XT4 versus XT5

4.3 HALO

The HALO benchmark [23] 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 used the HALO benchmark suite to examine which MPI two-sided communication protocol is most efficient. On the basis of these data we used a single communication protocol in subsequent experiments.

For this paper, we examined the impact of process count on HALO performance. The halo ex-

change is "logically" a local operator, so we might expect performance not to vary with process count. However, the mapping of processes to processors determines the actual locality of the operator.

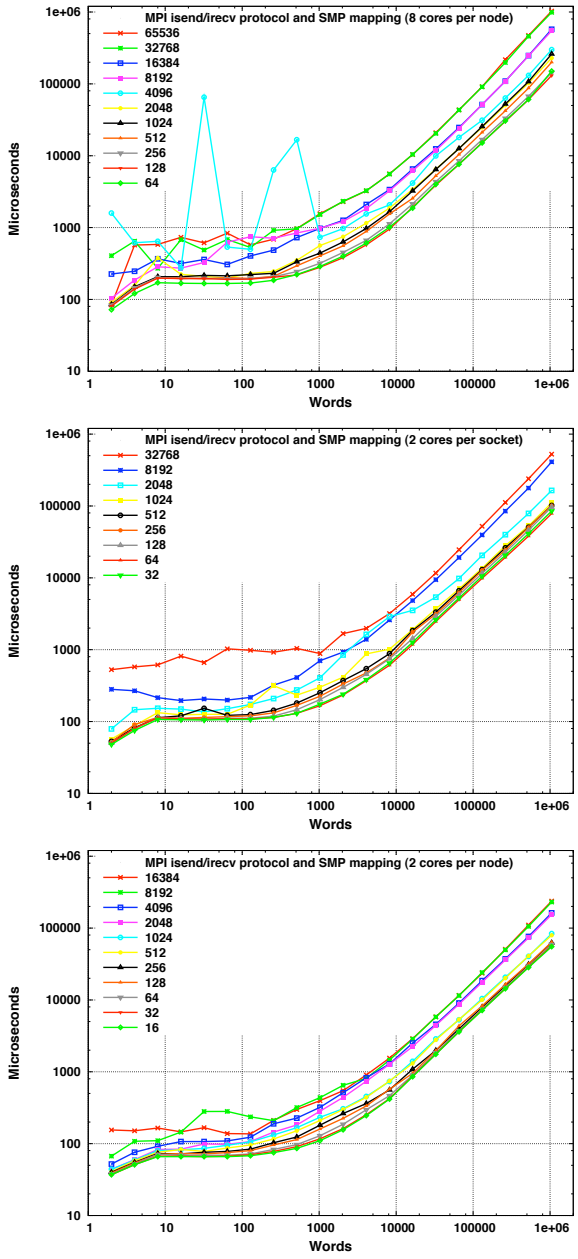

FIGURE 4.7: HALO performance scaling on the Cray XT5

Figure 4.7 describes HALO performance as a function of the size of the halo being exchanged for diffferent process counts. The top graph describes the performance when 8 processes are assigned to each node; the middle graph describes performance whan 4 processes are assigned to each node, two processes to each quad-core processor; the bottom graph describes the performance when 1 process is assigned to each node. Ignoring the few "noisy" data in some of the small halo exchanges, cost appears to grow nearly monotonically as a function of process count, even for relatively small halos. This is most evident in the top two graphs, with eight and four processes assigned to each node. For these experiments, the default "SMP" assignment of processes to nodes was used, e.g., for the first graph processes 0 through 7 were assigned to node 0, processes 8 through 15 were assigned to node 1, etc. This mapping does not minimize the internode communication for the underlying two-dimensional virtual process grid. However, the growth appears to be too regular to be attributable to simply this source. Moreover, performance when assigning one node to each process also grows monotonically with the process count. It is difficult to attribute this to network performance, given our inability to saturate network links in other experiments. Rather, this could be a signature of performance degradation due to OS jitter, especially given the greater sensitivity to process count for smaller halo exchanges.

4.4 Communication Performance Summary

Performance is consistent between the different point-to-point experiments. From these experiments, the performance bottleneck on the XT5 is communication from the node into the network, and not in the network itself. Experience in a multi-user environment appears to indicate that the aggregate network traffic can saturate network links, but we have not been able to duplicate that behavior with these benchmarks. The experiments also indicate that bandwidth into the network is divided equally between competing MPI processes. The movement from XT4 compute nodes with 4 processor cores to XT5 compute nodes with 8 processor cores can halve the MPI internode performance observed by a single process. MPI latency also increases when multiple MPI processes are communicating simultaneously. Techniques to minimize contention, such a using OpenMP parallelism to decrease the number of MPI processes assigned to a single compute node or using communication algorithms that are aware of the node architecture and aggregate messages locally before communicating between nodes, are important to consider on the XT5.

Experiments with collective operators indicate additional performance issues. The best observed

MPI_Barrier performance is comparable to that on XT4-dual core when running the Catamount operating system, and only 1.4 times slower than that on the IBM BG/P when not using the hardware barrier network. Where the XT5 performance is different is in the worst case performance, which can be a thousand times slower than the best observed performance. This performance perturbation is likely not in the MPI library, but rather something external. It does not have the usual signature of operating system jitter, and is currently being investigated by staff at both Cray and ORNL. In contrast, operating system jitter may be a partial explanation of the increase in runtime in the HALO benchmark as the number of processes increases.

5 Conclusions

The new XT5 system has great promise as a platform for large scale computational science. In order to make maximum use of the system, application developers will need to reexamine the performance characteristics of their codes, looking for ways to decrease memory traffic. Techniques for decreasing the amount and frequency of internode MPI communication, such as exploiting OpenMP parallelism or SMP-aware MPI algorithms, should also be considered.

In contrast, it is the responsibility of the vendor and computer center staff to diagnosis the source of performance variability, and to minimize it to the extent possible. Without a such a diagnosis, application developers can not develop mitigation strategies.

6 Acknowledgements

This research used resources (Cray XT4, Cray XT5, and IBM BG/P) 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 Authors

Richard F. Barrett is a senior R&D staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory. His research interests span several areas required for creating effective scientific applications on current and future highest performance computing platforms. Of special interest are the use of programming models and languages, such as explicit message passing, partitioned global address space languages, and developing languages such as those in the DARPA High Productivity Computer Systems Program; code development tools; performance modeling, analysis, and optimization; computer architectures; inter-process communication mechanisms; the solution of large scale linear systems; and the bridge between research and production computing..
E-mail: rbarrett@ornl.gov.

Jeffrey A. Kuehn joined the Computer Science and Mathematics Division of Oak Ridge National Laboratory in February of 2005 as a Senior HPC Evaluation Researcher. He received his undergraduate and graduate degrees from University of Colorado at Boulder in Mechanical Engineering. His expertise is in software performance and system benchmarking. Jeff has played key roles in bringing over 30 "Top500" computing systems to production, many in the top 25. His role at ORNL encompasses the evaluation of next-generation high performance computing systems and technologies. Jeff's current research interests center around the development and use of microbenchmarks, system performance analysis, and software performance engineering.

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 simulation models used in climate and fusion energy research) and the performance evaluation of parallel applications and computer systems. He is currently a co-chair of the CCSM Software Engineering Working Group, the principal investigator for the Performance Engineering and Analysis Consortium End Station DOE INCITE project, and is an Associate Editor of the journal Parallel Computing. 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] S. Alam, R. Barrett, M. Eisenbach, M. Fahey, R. Hartman-Baker, J. Kuehn, S. Poole, R. Sankaran, and P. Worley, *The Cray XT4 Quad-core : A First Look*,

in Proceedings of the 50th Cray User Group Conference, May 5-8, 2008, R. Winget and K. Winget, ed., Eagan, MN, 2008, Cray User Group, Inc.

[2] S. R. Alam, R. F. Barrett, M. R. Fahey, J. A. Kuehn, J. M. Larkin, R. Sankaran, and P. H. Worley, *Cray XT4: An Early Evaluation for Petascale Scientific Simulation*, in Proceedings of the ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC07), Nov. 10-16, 2007, IEEE Computer Society Press, Los Alamitos, CA, 2007.

[3] 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.

[4] 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 (2006), pp. 2144–2161.

[5] 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.

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

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

[8] 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.

[9] J. Dongarra and P. Luszczek, *Introduction to the HPCChallenge Benchmark Suite*, Tech. Rep. UT-CS-05-544, Computer Science Department, University of Tennessee, Knoxville, Tennessee, 2005. http://icl.cs.utk.edu/hpcc/.

[10] T. H. Dunigan, Jr., *Kendall square multiprocessor: Early experience and performance*, Tech. Rep. ORNL/TM-12065, Oak Ridge National Laboratory, Oak Ridge, TN, March 1992.

[11] T. H. Dunigan, Jr., M. R. Fahey, J. B. White III, and P. H. Worley, *Early Evaluation of the Cray X1*, in Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC03), Nov. 15-21, 2003, IEEE Computer Society Press, Los Alamitos, CA, 2003.

[12] M. R. Fahey, S. Alam, T. H. Dunigan, Jr., J. S. Vetter, and P. H. Worley, *Early Evaluation of the Cray XD1*, in Proceedings of the 47th Cray User Group Conference, May 16-19, 2005, R. Winget and K. Winget, ed., Eagan, MN, 2004, Cray User Group, Inc.

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

[14] M. T. Heath, G. A. Geist, and J. B. Drake, *Early experience with the Intel iPSC/860 at Oak Ridge National Laboratory*, Tech. Rep. ORNL/TM-11655, Oak Ridge National Laboratory, Oak Ridge, TN, September 1990.

[15] 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.

[16] D. Koester and R. Lucas, *Random Access Benchmark*. http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/.

[17] J. Kuehn, J. Larkin, and N. Wichmann, *An Analysis of HPCC Resuls on the Cray XT4*, in Proceedings of the 49th Cray User Group Conference, May 7-10, 2007, R. Winget and K. Winget, ed., Cray User Group, Inc., Eagan, MN, 2007.

[18] J. D. McCalpin, *Memory Bandwidth and Machine Balance in Current High Performance Computers*, IEEE Computer Society Technical Committee on Computer Architecture Newsletter, (1995). http://tab.computer.org/tcca/news/dec95/dec95.htm.

[19] R. Mills, F. Hoffman, P. Worley, K. Perumalla, A. Mirin, G. Hammond, and B. Smith, *Coping at the User-Level with Resource Limitations in the Cray Message Passing Toolkit MPI at Scale: How Not to Spend Your Summer Vacation*, in Proceedings of the 51st Cray User Group Conference, May 4-7, 2009, R. Winget and K. Winget, ed., Eagan, MN, 2009, Cray User Group, Inc.

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

[21] D. Takahashi, *FFTE: A Fast Fourier Transform Package*. http://www.ffte.jp/.

[22] J. S. Vetter, S. R. Alam, T. H. Dunigan, Jr., M. R. Fahey, P. C. Roth, and P. H. Worley, *Early Evaluation of the Cray XT3 at ORNL*, in Proceedings of the 47th Cray User Group Conference, May 16-19, 2005, R. Winget and K. Winget, ed., Eagan, MN, 2005, Cray User Group, Inc.

[23] 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.

[24] P. H. Worley, T. H. Dunigan, Jr., M. R. Fahey, J. B. White III, and A. S. Bland, *Early evaluation of the IBM p690*, in Proceedings of the IEEE/ACM SC2002 Conference, Nov. 16-22, 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002.