Comparative Analysis of Interprocess Communication on the X1, XD1, and XT3 *

P. H. Worley †
S. Alam, T. H. Dunigan, Jr.,
M. R. Fahey, J. S. Vetter

Oak Ridge National Laboratory

Abstract

All three of Cray's most recent products, X1/X1E, XD1, and XT3, are built around high performance interconnects. In this paper we contrast and compare interprocess communication performance for these three systems. We describe peak point-to-point and collective communication performance, looking at the performance impact of network topology. We also analyze the performance of two application benchmarks whose performance is sensitive to communication latency and bandwidth, respectively, to identify the practical implications of the communication benchmark results.

1 Introduction

The X1 was the first of Cray's new scalable vector systems [5, 10]. The X1 is characterized by high-speed custom vector processors, high memory bandwidth, and a high-bandwidth, low-latency interconnect linking the nodes. Oak Ridge National Laboratory (ORNL) installed a 32 processor (MSP) Cray X1 in March 2003. This grew to a 128 processor system in July 2003, to a 256 processor system in October 2003, and to a 512 processor system in June 2004. A 1024 processor X1E will be installed during the summer of 2005.

The Cray XD1 (formerly from Octigabay) is an AMD Opteron-based cluster with a low latency RapidArray interconnect [6, 12]. ORNL installed two 72 processor Cray XD1s for evaluation in October 2004. The systems were reconfigured as a single 144 processor system in May 2005.

The Cray XT3 is Cray's third-generation massively parallel processing system [7, 16]. The system builds on a single processor node, using the AMD Opteron, and uses a custom chip to provide interprocessor communication. ORNL installed a 96 processor system in January 2005. A 1900 processor system was installed in March 2005. This second system grew to 3800 processors in April 2005, and will grow to 5200 processors during the summer of 2005.

The performance of the current class of HPC architectures is dependent on the performance of the memory hierarchy, including that of the interconnect between nodes in a cluster. All three of the current Cray products provide high performance interconnects, but they utilize different technologies, resulting in differences in latency, bandwidth, scalability, and price point. In this paper we examine the impact of these different approaches on performance, and performance peculiarities, of the interconnects of all three Cray systems. We use a number of standard and custom microbenchmarks to examine the basic characteristics of communication performance. We then examine the performance of two application codes that are known to be sensitive to the latency and bandwidth, respectively, in order to identify the practical impact of the differences in communication characteristics.

Note that all three systems are undergoing rapid development. The XT3 latency is expected to drop

*This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, 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)

from 30 microseconds to under 5 microseconds over the next year. The XD1 has an optional fat tree configuration that should increase peak bandwidth significantly. The X1 at ORNL is being upgraded to an X1E, which will double the contention for internode bandwidth, decreasing peak communication performance but increasing peak computational rate and decreasing price per processor. Thus the performance reported will change over time, and this paper represents a time-sensitive snapshot of the communication performance characteristics.

The outline of the paper is as follows. Section 2 is an overview of the three Cray interconnects. It also includes a brief description of all of the systems mentioned in the paper. Section 3 is a description of the MPI point-to-point microbenchmark results. Section 4 is a description of the collective operator microbenchmark results. Section 5 is a description of the application benchmark results. Section 6 contains concluding remarks.

2 Interconnect Descriptions

The focus here is on the interconnects of the three systems. However, larger issues of system design, including the processor and symmetric multiprocessor (SMP) node, are also important to understanding the interconnect performance, and will be summarized where appropriate.

2.1 Cray X1

The X1 has a hierarchical design with the basic building block being the multi-streaming processor (MSP), which is capable of 12.8 GFlop/s for 64-bit operations (or 25.6 GFlop/s for 32-bit operations). Each MSP is comprised of four single-streaming processors (SSPs), each with two 32-stage 64-bit floating-point vector units and one 2-way superscalar unit. The SSP uses two clock frequencies, 800 MHz for the vector units and 400 MHz for the scalar unit. Each SSP is capable of 3.2 GFlop/s for 64-bit operations. The four SSPs share a 2 MB "Ecache."

Four MSPs, 16 memory controller chips (M-chips), and 32 memory daughter cards form a Cray X1 node. The memory banks of a node provide 204 GB/s of bandwidth, enough to saturate the paths to the local MSPs and service requests from remote MSPs. Local memory latency is uniform for all processors within a node. Each bank of shared memory is connected to a number of banks on remote nodes, with an aggregate bandwidth of roughly 50 GB/s between nodes.

The Cray X1 nodes are connected using X1 routing modules. Each node has 32 1.6 GB/s full duplex links. Each memory module has an even and odd 64-bit (data) link forming a plane with the corresponding memory modules on neighboring nodes. Eight adjacent nodes connected this way form a processor stack. An X1 cabinet is comprised of 16 node boards and 4 routing boards (or two processor stacks). Each routing board has 8 routing modules. The routing module ASIC is an 8-way non-blocking crossbar switch supporting worm-hole routing. Communication latency increases by about 500 ns per router hop. With 8 or fewer cabinets (up to 128 nodes or 512 MSPs), the interconnect topology is a 4-D hypercube. Larger configurations use an enhanced 3D torus, where one dimension of the torus, the processor stack, is fully connected.

For most of these experiments the ORNL system was running with programming environment version PrgEnv5.3.0.2 and operating system version UNICOS/mp 2.5.38. For more information on the system specifics and on the performance characteristics of the ORNL Cray X1, see [10, 1, 11].

2.2 Cray XD1

The experiments described here were run on six chassis of early access Cray XD1 nodes. Each chassis has 6 SMP nodes, each node with 2 64-bit AMD Opteron 248 series processors. Each processor includes a single Opteron core, integrated memory controller, three 16b 800 MHz HyperTransport (HT) links, a 64KB L1 instruction cache, a 64KB L1 data cache, a 1MB L2 cache, and 4 GB of memory. The Opteron core has a 2.2 GHz clock, three integer units, and one floating-point unit that is capable of two floating-point operations per cycle [2], for a peak rate of 4.4 GFlop/s per processor. The total system had 72 processors and 288 GB of memory. (The system has since been upgraded to twelve chassis and 144 processors.)

Custom *RapidArray Communications Processors* interface the Opteron HyperTransport bus to the RapidArray interconnect fabric. In the *direct connect topology* used in the ORNL system there are two network fabrics, *main* and *expansion*. The main fabric has two RapidArray links per node, achieving 4 GB/s per node and 48 GB/s aggregate bandwidth per chassis. There are also 12 external Rapid Array interchassis links, for an aggregate interchassis bandwidth of 24 GB/s. Utilizing the expansion fabric as well doubles the number of links and the peak bandwidths. For the ORNL six chassis system, each

chassis was connected to every other chassis by two Rapid Array interchassis links for each fabric. An optional switch can be used to instead construct a fat tree network, supporting up to 8 GB/s bandwidth. This was not available on the ORNL system at the time of these experiments.

For the experiments presented in this paper the ORNL system was running Linux version 2.4.21 with a modified Linux scheduler that supports synchronized system interrupts, which improves application performance scalability. The MPI communication library was a port of MPICH 1.2.5.

For more information on the system specifics and on the performance characteristics of the ORNL Cray XD1, see [12].

2.3 Cray XT3

The XT3 system utilizes a single processor node, or processing element (PE), and a custom interconnect. The XT3 uses a blade approach for achieving high processor density per system cabinet. On the XT3, a compute blade hosts four compute PEs, and eight blades are contained in one chassis. Each XT3 cabinet holds three chassis, for a total of 96 processors per cabinet.

The ORNL XT3 uses 64-bit AMD Opteron model 150 processors. This model includes a single Opteron core, integrated memory controller, three 16b 800 MHz HyperTransport (HT) links, a 64KB L1 instruction cache, a 64KB L1 data cache, a 1MB L2 cache, and 2 GB of memory. The Opteron core has a 2.4 GHz clock, three integer units, and one floating-point unit that is capable of two floating-point operations per cycle [2], for a peak rate of 4.8 GFlop/s per processor.

Each Opteron processor is directly connected to the XT3 interconnect via a Cray SeaStar ASIC (application specific integrated circuit). The SeaStar is a routing and communications device. It acts as the gateway to the XT3's high-bandwidth, low-latency interconnect. The PE is connected to the SeaStar chip with a 6.4 GB/s HT path. The router in SeaStar provides six high-speed network links to connect to six neighbors in a 3D torus/mesh topology. Each of the six links has a peak bandwidth of 7.6 GB/s. With this design, the Cray XT3 bypasses communication bottlenecks such as the PCI bus. The interconnect carries all message passing traffic as well as I/O traffic.

The XT3 at ORNL is currently a 40 cabinet system. These PEs are connected in a 10 x 16 x 24 (X x Y x Z) configuration with a torus in X and Z dimensions and a mesh in the Y dimension. The topology of the ORNL XT3 is illustrated in Fig. 2.1. Each "logical" 16x24 processor cabinet in the figure is made up of four 4x24 physical cabinets. The X-dimension ordering of cabinets is 0-2-4-6-8-9-7-5-3-1.

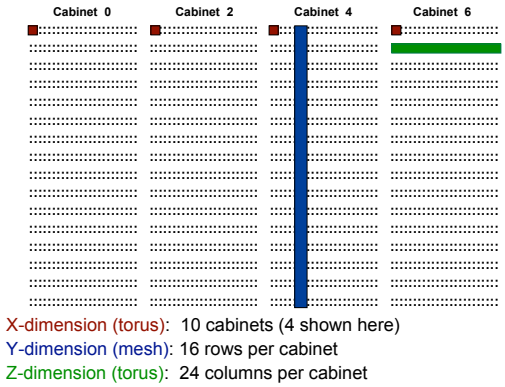


FIGURE 2.1: Topology of XT3 at ORNL

For these experiments the ORNL system was running the Catamount operating system version 1.15 on the compute PEs and SuSe Linux version 2.4.21 on the service PEs. The MPI communication library was a port of MPICH 1.2.5. For more information on the system specifics and on the performance characteristics of the ORNL Cray XT3, see [16].

2.4 System Summaries

Table 1 summarizes some of the processor and SMP node specifications for the three Cray systems. We also include data for an SGI Altix 3700, an IBM p690 cluster with an HPS interconnect, the IBM SP at the National Energy Research Scientific Computing Center (NERSC), and the HP/Compaq AlphaServer SC at the Pittsburgh Supercomputer Center (PSC) that we use for comparisons later in the paper.

3 Point-to-Point Benchmarks

The first step in our evaluation was to measure MPI latency and bandwidth in point-to-point communications. Table 2 summarizes the performance we observed with custom microbenchmarks developed at ORNL. These benchmarks are similar in spirit to most communication microbenchmarks, and the results should be comparable to those in other benchmark suites.

These data indicate the following.

	X1	XD1	XT3	Altix 3700	p690 cluster	SP	Alpha-Server SC
Processor	Cray MSP	AMD Opteron	AMD Opteron	Intel Itanium2	IBM Power4	IBM Power3-II	Compaq Alpha EV68CB
MHz	800	2200	2400	1500	1300	375	1000
L1	16KB	64KB	64KB	32KB	32KB	64KB	64KB
L2	2MB	1MB	1MB	256KB	1.5MB	8MB	8MB
L3				6MB	128MB/node		
GFlop/s/proc.	12.8	4.4	4.8	6.0	5.2	1.5	2.0
proc./node	4	2	1	2	32	16	4
memory/node	16GB	8GB	2GB	16GB	32-128GB	16-64GB	4GB
total proc.	512	144	3800	256	864	6080	764

Table 1: System Descriptions

MPI	X1	XD1	XT3	Altix	p690 cluster	SP	Alpha-Server SC
Latency (8 Byte msg., 1 way, microseconds)							
intra-node	7.3	1.7	–	1.1	3	8.6	4.9
inter-node	7.3	1.7	29	1.1	6	17	4.6
Bandwidth (1 MByte msg., unidirectional, MB/s)							
intra-node	9503	1087	–	1595	1580	600	733
inter-node	9364	1342	1111	1397	936	320	265
Bandwidth (1 MByte msg., bidirectional, MB/s)							
intra-node	17145	1095	–	2286	2402	NA	715
inter-node	16936	2173	2150	2561	985	356	254

Table 2: Measured MPI Performance

- MPI latency is small on both the XD1 and the Altix.
- MPI latency is relatively large on the X1, and is very large on the XT3 (currently).
- MPI bandwidth is much higher on the X1 than on the other systems.
- Bidirectional bandwidth is significantly higher than unidirectional bandwidth (up to twice as much) on all of the Cray systems except intra-node on the XD1. The XD1 results reported here use the current default in which MPI communication between two processors in the same node goes through the RapidArray network fabric. There is also an option on the XD1 to use a shared memory implementation of MPI for intra-node communication. This could change the intra-node performance characteristics.

As will be shown later in the paper, the SHMEM communication library [13] and Co-Array Fortran [15] both provide much lower communication latency than MPI on the X1. SHMEM is also available on the XD1 and XT3, but it does not provide performance superior to MPI on either system at the current time.

3.1 Distance

The next aspect of communication performance that we examine is *distance*, i.e. the extent to which the physical distance between processors affects communication performance when all of the other allocated processors are idle.

Cray X1. The X1 experiments were run on 92 contiguously-numbered MSPs in a nondedicated system. Figure 3.1 contains graphs of both bidirectional and unidirectional bandwidth measured for

a number of different message sizes (8B - 8MB). While results are somewhat noisy, the variation is less than 20% for all message sizes with no obvious distance-related dependencies, with the exception of intra-node communication (distance 1,2,3) for message sizes between 32B and 2048B. Note that unidirectional bandwidth is 50%-60% of bidirectional bandwidth for the largest and smallest message sizes, but rises to 75% for messages of size between 512B and 2048B.

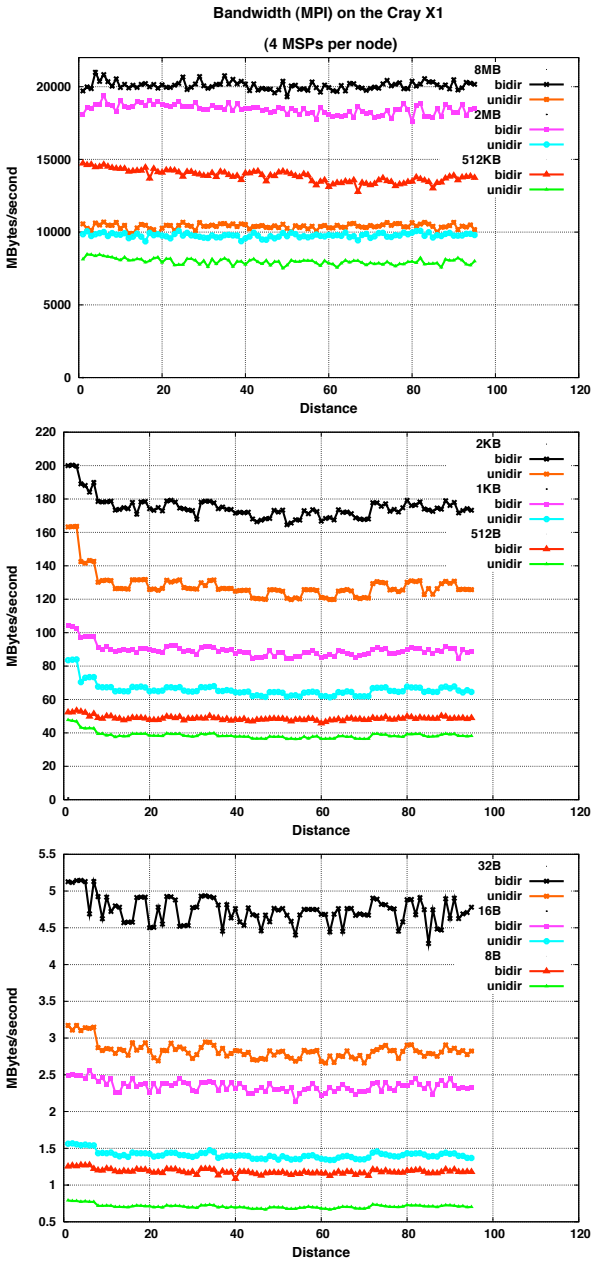


FIGURE 3.1: X1 Distance Sensitivity

Cray XD1. The XD1 distance experiments were run on 5 chassis where consecutive process ids were assigned to consecutive processors in a chassis. (The scheduler by default does not assign processes in this manner.) Figure 3.2 contains graphs of bidirectional bandwidth only, plotted with both linear and logarithmic Y-axes. While a difference between performance within a chassis and between chassis is apparent for small messages, distance does not impact performance significantly, except that intra-node bandwidth is approximately half that of inter-node bandwidth for the largest message sizes. This may change if the shared memory implementation of MPI is used within a node.

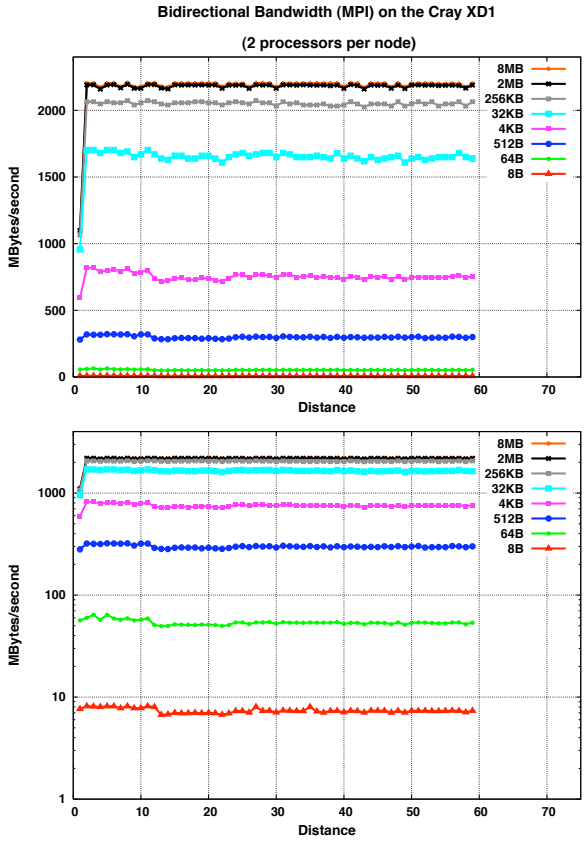


FIGURE 3.2: XD1 Distance Sensitivity

The graph in Fig 3.3 compares performance both with and without using the expansion fabric, indicating that enabling the expansion fabric does not improve performance currently. (Data from experiments using the expansion fabric only go out to a distance of 47.) The graphs in Fig 3.4 compare unidirectional and bidirectional bandwidth for a subset of the message sizes. Unidirectional bandwidth is approximately 60% of bidirectional bandwidth for large messages and 50% for small messages.

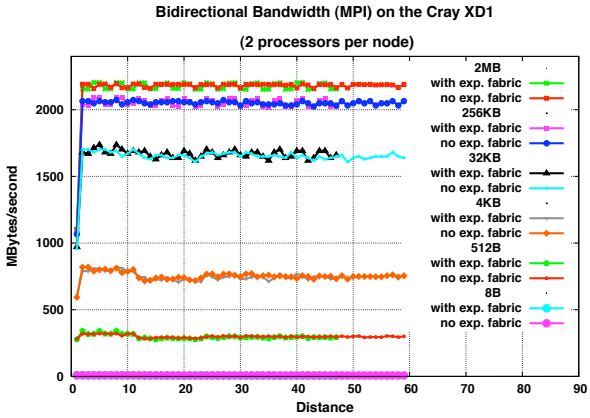


FIGURE 3.3: Performance Impact of XD1 Expansion Fabric

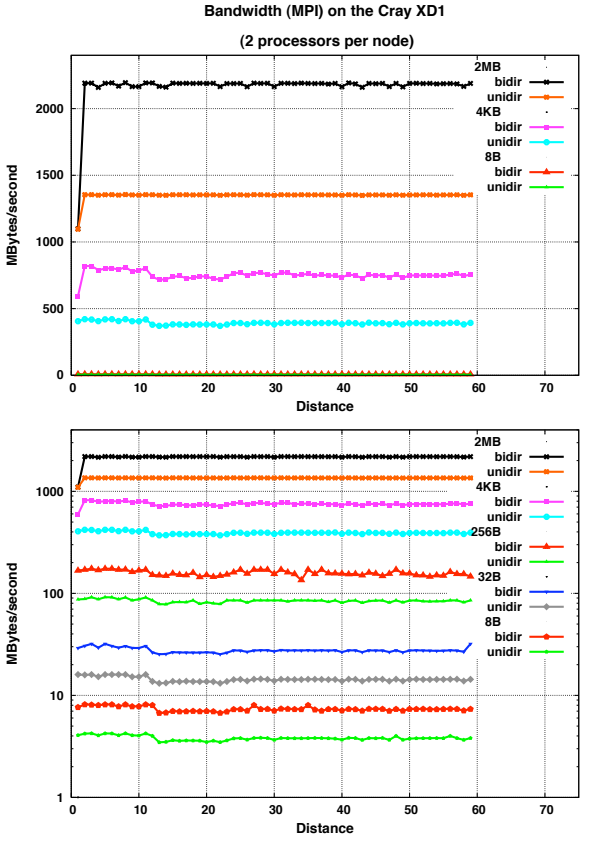


FIGURE 3.4: XD1 Uni- vs. Bi-directional Performance Comparison

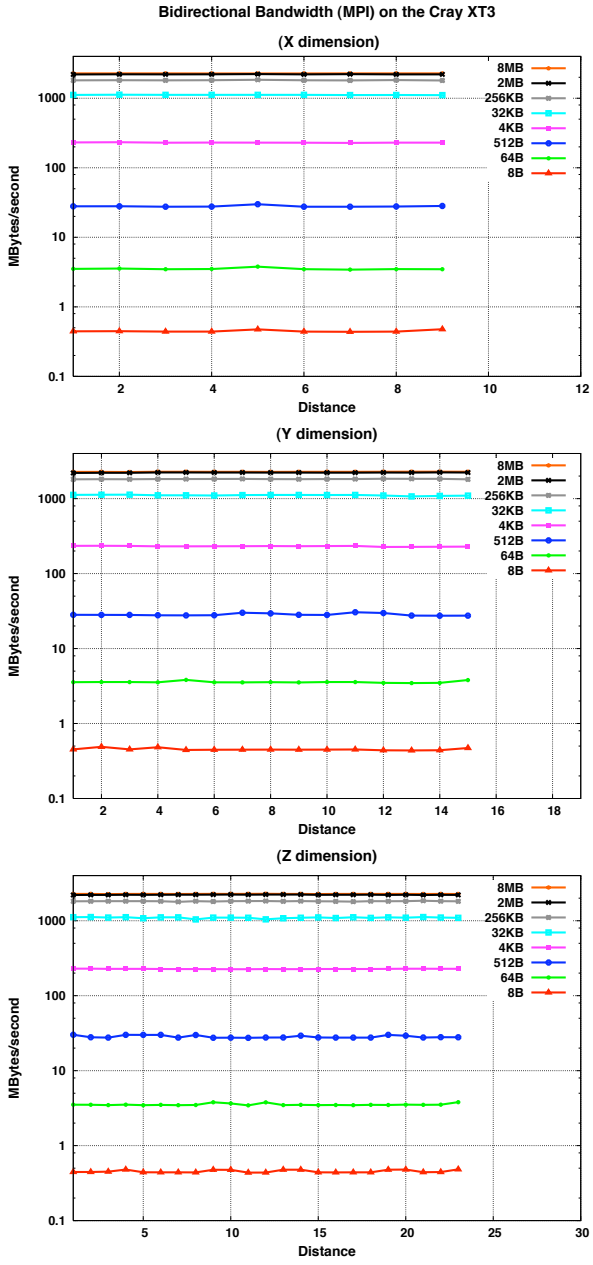


FIGURE 3.5: XT3 Distance Sensitivity: X, Y, Z

Cray XT3. For the XT3 distance experiments we first examined each coordinate direction separately. These results are contained in Fig. 3.5. Even plotted with logarithmic Y-axes, it is clear from these data that neither distance nor coordinate direction impact performance in these experiments.

To verify that these experiments are sufficient to characterize the impact of distance, Fig. 3.6 contains plots for a distance experiment over 96 processors of a 4x24 YxZ subgrid. Figure 3.7 compares unidirectional and bidirectional bandwidth for the same YxZ subgrid. Here unidirectional bandwidth is almost exactly 50% of the bidirectional bandwidth in all cases.

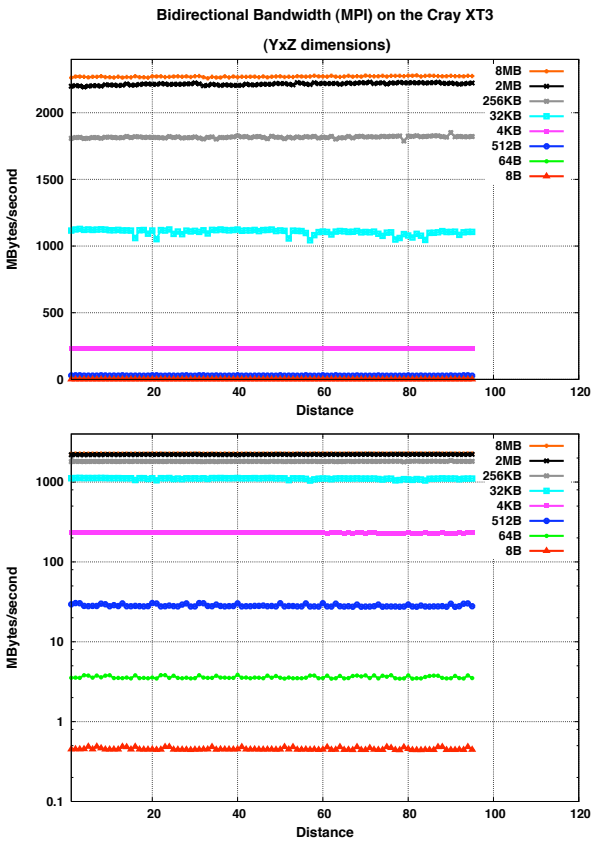


FIGURE 3.6: XT3 Distance Sensitivity: YxZ

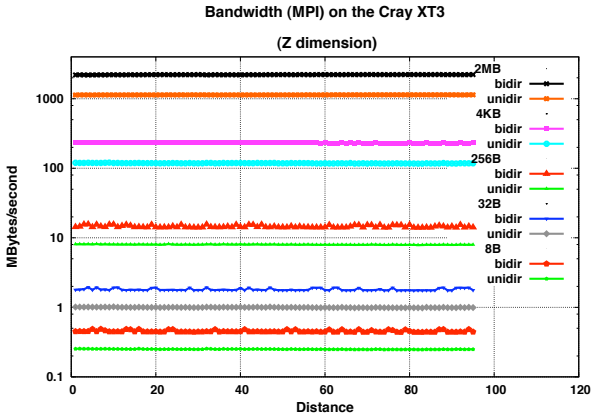


FIGURE 3.7: XT3 Uni- vs. Bi-directional Performance Comparison

Distance Summary. MPI bandwidth (and latency) is not sensitive to the distance between communicating processors on any of the systems, except for intranode communication on the X1 and XD1. (Remember however that latency on the XT3 is very high currently and this may mask sensitivity to distance for small messages.) MPI unidirectional bandwidth is 50% to 60% that of bidirectional bandwidth, except on the X1 for 512B to 2048B message sizes where it is 75%. MPI peak bandwidth on the X1 is much higher than that on the other Cray systems and MPI latency on the XD1 is much lower than that on the other Cray systems, both in agreement with the earlier point-to-point microbenchmark results. While XT3 MPI performance is degraded by the current high latency, the peak bandwidth is comparable to that on the XD1. Also, the XD1 expansion fabric did not enhance communication in these experiments.

3.2 Contention

While the distance experiments were an interesting examination of basic performance characteristics, pairs of processes rarely communicate in isolation. A more realistic scenario is when multiple pairs are communicating simultaneously, perhaps as one stage in a collective communication operation. The next set of experiments attempt to examine the worst possible performance in such a scenario, when $p/2$ processor pairs are exchanging messages simultaneously, with processor i communicating with processor $i+p/2$ for $i = 0, ..., (p/2-1)$. The X-axes used in the figures in this section are labelled as "Amount of Data Sent in Each Direction". Thus an X value of 1 MB indicates that each processor in a pair sent a message of length 1 MB to its partner, for a total of 2 MB of data being sent simultaneously, 1 MB in each direction.

Cray X1. For the X1 experiments we used 192 consecutive processors, looking at achieved bandwidth per processor pair under contention when 1, 2, 4, 8, 16, 32, 64, and 96 processor pairs are communicating simultaneously for a range of message sizes. The results are contained in Fig. 3.8. The first graph is a linear-linear plot while the second is a linear-log plot, both of the same data. The two views enable the examination of large and small message performance, respectively.

Contention reduces single pair performance for large messages significantly (from 20 GB/sec to 500 MB/s in the worst case), and there is no evidence that a lower bound on the performance degradation has been identified yet. However, it is clear that there is little contention for messages smaller than approximately 1024 Bytes. It is also clear that there a number of distinct performance curves (intranode, no more than 32 processors, no more than 128 processors, greater than 128 processors), reflecting the

underlying network topology.

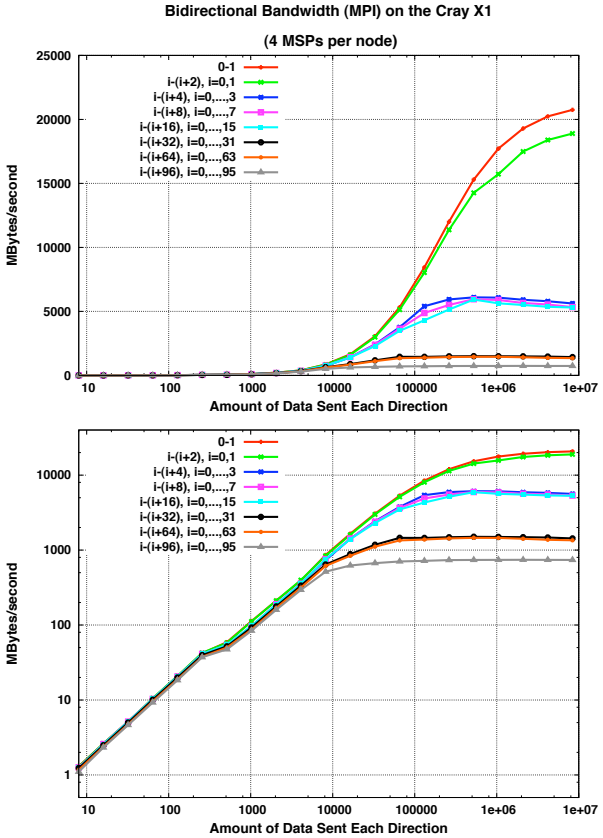


FIGURE 3.8: X1 Communication Performance under Contention

Cray XD1. For the XD1 we again controlled the placement of processes on processors. We looked at bandwidth per processor pair under contention when 1, 2, 4, 12, 18, 24, and 30 processor pairs are communicating simultaneously. Note that 1,2, and 4 pairs reside in the same chassis, 12 pairs involve all processors in 2 chassis, 18 pairs use 3 chassis, 24 pairs use 4 chassis and 30 pairs use all processors in 5 chassis. The results are contained in Fig. 3.9. Contention within a chassis is the same as the contention when two processors in the same SMP node are communicating, i.e. bandwidth per processor pair is halved. When two and four chassis are involved, the performance drops to approximately one eighth of the maximum performance. (For the direct connect topology the four chasis experiment is equivalent to running two independent two-chassis experiments simultaneously, and we would expect the results to be identical.) The performance for the three chassis experiment is worse than that for the two chassis experiments, but performance for the 5 chassis experiment is better than that for the two chassis experiment. It is not clear at the current time why the direct connect topology would cause this effect.

Note that performance degradation due to contention is evident even for small messages, with processor pairs residing in the same chassis achieving the best performance. Again, performance for the 5 chassis experiment is better than that for the other multiple chasis experiments for small messages. Now, however, all of the other multiple chassis experiments demonstrate the same performance. Given that these experiments were all run over a two day period that did not include any reboots, and that the 5 chassis experiment was the last to be run (so if any nodes became "slow" or links went down, the 5 chassis experiment would also be affected), the data appears to be internally consistent.

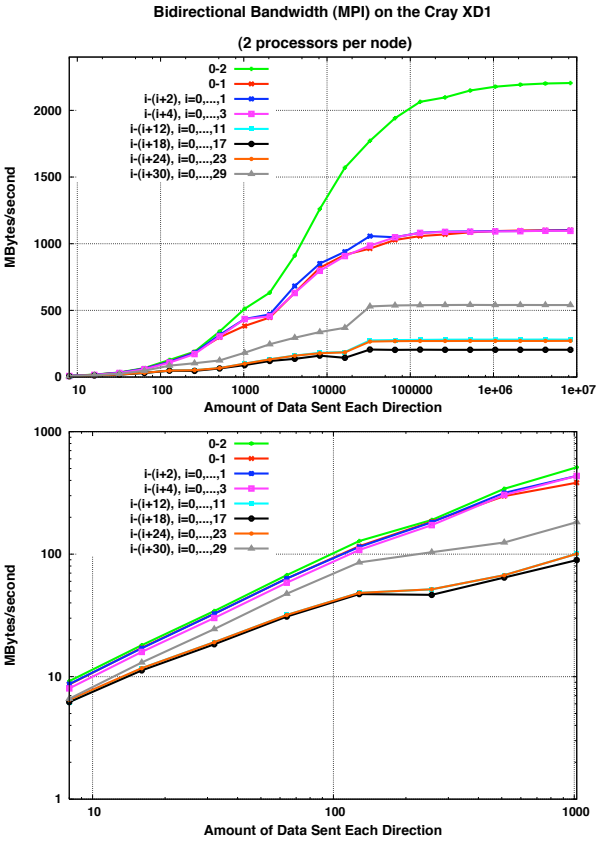


FIGURE 3.9: XD1 Communication Performance under Contention

For Fig. 3.10 we repeated the experiment but with the expansion fabric enabled. One chassis was unavailable at the time of the experiment, and only the 1, 2, 3, and 4 chassis experiments were run. These data indicate that the expansion fabric does

not have any impact on the contention experiment results.

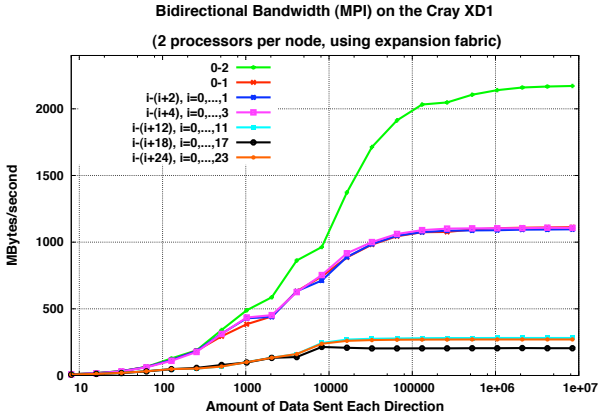


FIGURE 3.10: Communication Performance under Contention with the XD1 Expanision Fabric

To verify the nature of these results, we also ran contention experiments using only one processor per SMP node. In this case a 6 pair experiment now uses 2 chassis, a 9 pair experiment uses 3 chassis, etc. The results are plotted in Fig. 3.11. As the intranode contention is eliminated and the interchassis message volume is halved, the performance on a per SMP node basis is doubled for large message sizes as compared to the results described in Fig. 3.9, as expected. The experiments preserve the unexpected differences between 3 and 5 chassis experiments.

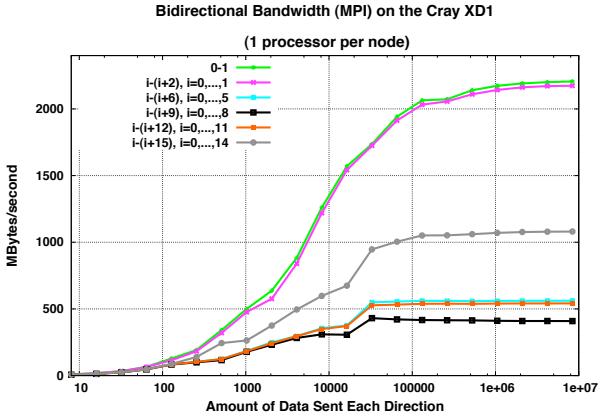


FIGURE 3.11: XD1 Node Communication Performance under Contention

Cray XT3. For the XT3, we begin by examining the maximum contention in each coordinate directions, using 5 processor pairs in the X dimension, 8 pairs in the Y dimension, and 12 pairs in the Z dimension, followed by 32, 48, 64, and 96 pairs in 4x16, 4x24, 8x16, and 8x24 YxZ subgrids, respectively. These results are plotted in Fig. 3.12. The Z dimension contention appears to represent the worse case behavior, and there is no performance degradation for small messages.

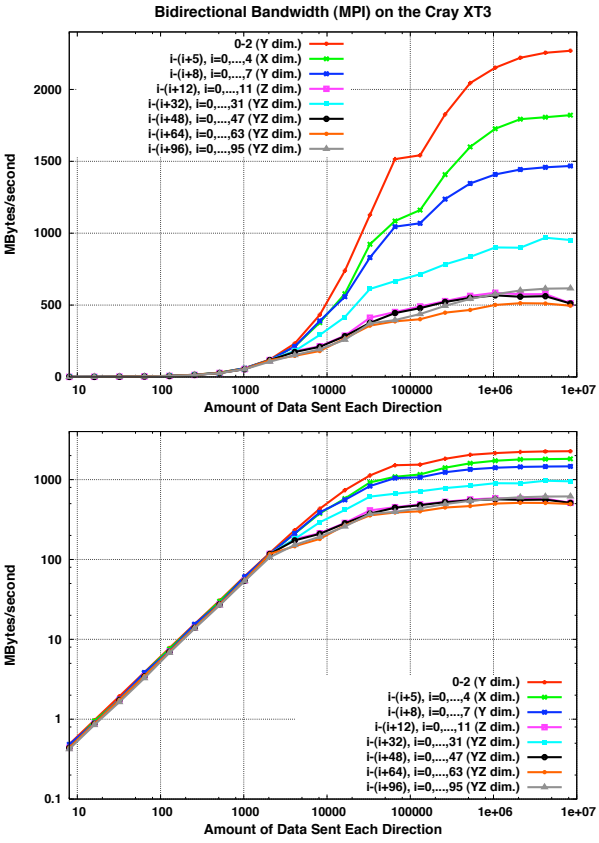


FIGURE 3.12: XT3 Communication Performance under Contention

While not necessarily significant, it is interesting to note that contention in the different dimensions is not identical, as illustrated in Fig. 3.13. In the X dimension, 4 pairs show more contention than 5 pairs, possible due to the ability to better exploit the torus when using 5 pairs. The 4 pair performance is the same in the X and Y dimensions. The performance in the Z dimension is more complicated. The processor numbering in the Z dimension does not reflect physical layout for the middle eight processors (8-15), rather it is in reverse ordering. The curves for the "alternative" ordering use the physical ordering. As can be seen, this increases contention, decreasing performance. If we were to use a similar alternative ordering in the 48, 64, or 96 pair YxZ contention experiments, we expect performance would drop to this lower performance level as well.

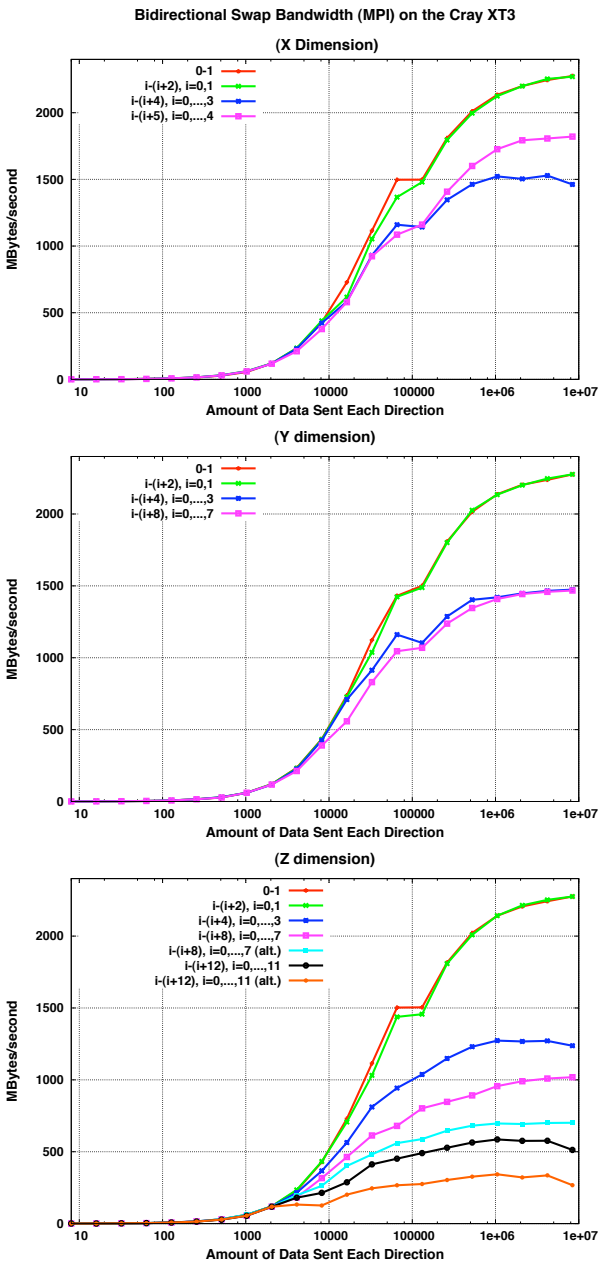


FIGURE 3.13: XT3 Communication Performance under Contention for each Dimension

Platform Intercomparisions. Figure 3.14 contains plots of bidirectional bandwidth with and without contention. Unlike the earlier graphs, the Y-axes are aggregate bandwidth, i.e. worst case single pair bandwidth multiplied by the number of simultaneous pairs. The first graph compares the three Cray systems when pairs are a distance of 32 apart for the X1 and XT3 and 30 apart for the XD1. For the XT3 results we used a 4x16 YxZ processor subset with the alternative ordering, to maximize contention. Here we again observe the latency advantage of the XD1 and the maximum bandwidth advantage of the X1. We also observe a higher aggregate bandwidth on the XT3 than on the XD1 under contention, even though the single pair performance is identical for the two systems.

In the second graph we compare performance for the X1 and XT3 with that for the SGI Altix and IBM p690 cluster when the distance between pairs is 64. For the XT3, the processor subset is the default produced by the scheduler, a 4x24 YxZ subset and a 4x8 YxZ subset in the next physical cabinet. With a separation of 64, the default ordering has similar contention characteristics to that of the alternative ordering and we used the default ordering. Here the Altix demonstrates the lowest latency. However, when compared with the results in the first plot, the XD1 latency results appear to be comparable. The X1 maximum bandwidth again is the best among the four systems. The bandwidth for the Altix and the XT3 are similar for the largest message sizes.

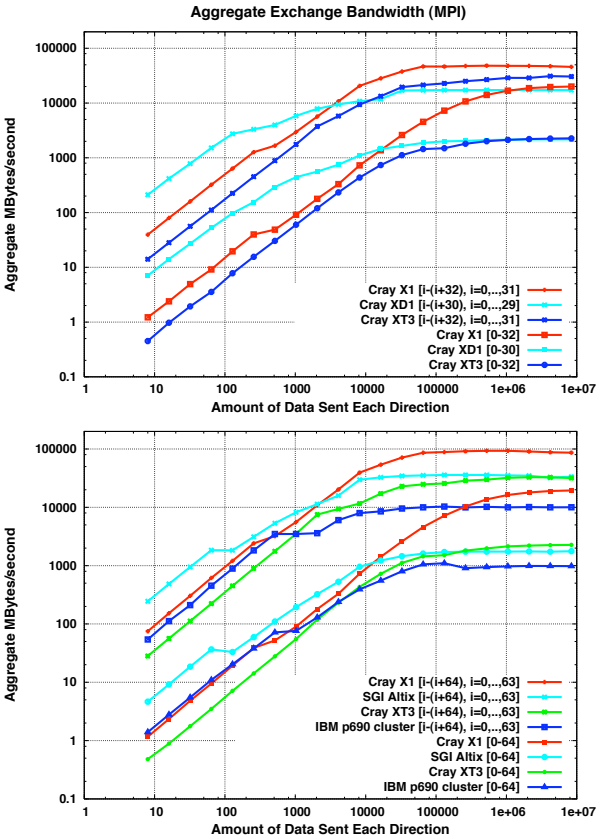


FIGURE 3.14: Communication Performance: Platform Intercomparisons

Contention Summary. The contention tests show that contention can limit the bandwidth achieved by a single processor pair when multiple processor pairs are communicating simultaneously. On the XT3 the details of this effect depend on the contention "direction". On the XD1 using 1 processor per node achieves twice the per pair bandwidth of using two processors for the same number nodes. This is equivalent to saying that the aggregate bandwidth is a function of the number of communicating node pairs, not the number of processor pairs. The X1 MPI peak bandwidth is much higher than that for the other systems, with and without contention, as per the specifications, while the XT3 and Altix achieve similar maximum aggregate bandwidth (for the given contention experiment). The XD1 performance is degraded by contention more than that of the XT3. For these experiments the XD1 used the direct connect topology. The XD1 contention performance may be improved if a fat tree topology is used instead.

4 Collective Operator Benchmarks

The distance and contention experiments attempt to measure best and worst case MPI performance on the respective interconnects. In applications, it is good practice to use standard collective communication operators whenever possible, allowing code sharing with other developers and exploitation of high efficiency implementations in optimized communication libraries. Here we examine the performance of two common operators: halo update and allreduce.

4.1 HALO

Wallcraft's HALO benchmark [17] simulates the nearest neighbor exchange of a 1-2 row/column "halo" from a 2-D array. This is a common operation when using domain decomposition to parallelize (say) a finite difference ocean model. There are no actual 2-D arrays used, but instead the copying of data from an array to a local buffer is simulated, and this buffer is transferred between nodes. Small message performance is dominated by latency, whereas bandwidth limits performance for larger messages.

Figure 4.1 contains plots of HALO performance for implementations based on different MPI communication protocols. Here 16 processors in a logical 4x4 processor torus are exchanging data. As halo update communicates only with neighbors in the logical processor grid, performance for the 16 processor experiment should represent performance for a range of processor counts, as long as logical neighbors are approximately physical neighbors and contention and distance do not increase significantly with processor count. (The torus connectivity is one possible reason for this assumption to be invalid for large processor counts.)

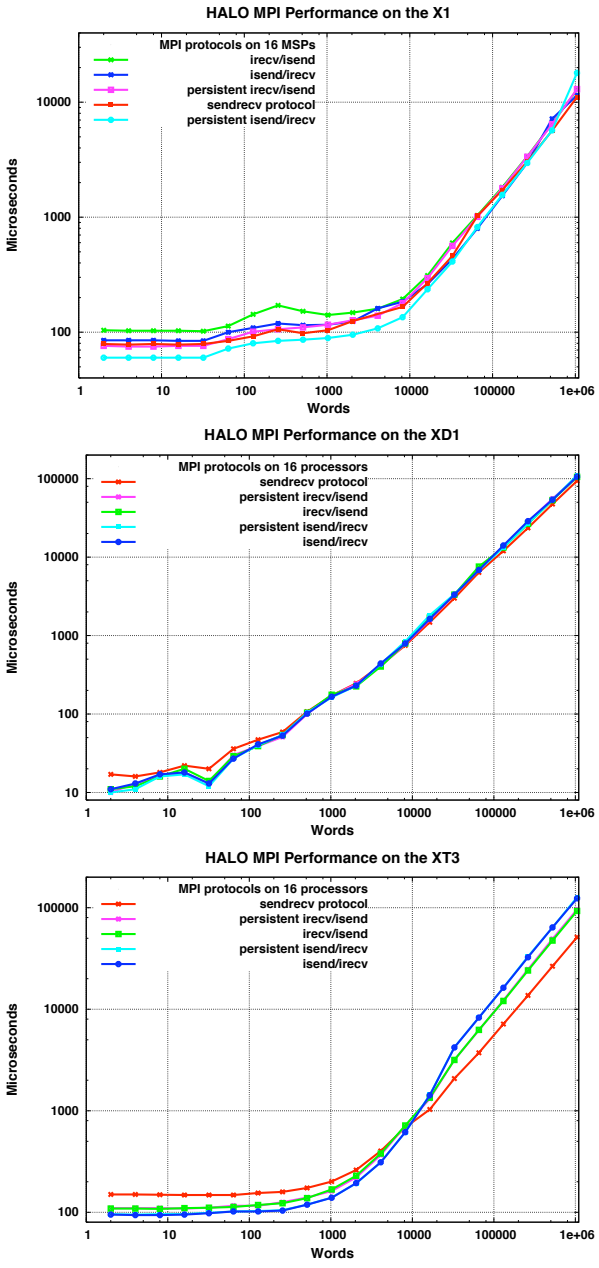


FIGURE 4.1: HALO MPI Implementation Comparisons

From these data we see that the MPI proto-

col can make a difference, especially for small messages. On the X1, persistent isend/irecv is clearly the best protocol until message sizes are over 128KB (16KWords), at which point the sendrecv protocol is optimal. In contrast, on the XD1 and XT3 the performance of the persistent protocols is identical to the nonpersistent analogues. On the XT3, isend/irecv is best up to messages of size 128KB, and is the worst proctocol for large message sizes, and the opposite is true for the sendrecv protocol.

MPI is not the only messaging layer available on the X1. SHMEM and Co-Array Fortran can also be used (in the same code with MPI) in order to move data between processors. HALO includes implementations using SHMEM and Co-Array Fortran. Figure 4.2 compares the performance of these different implementations. Using SHMEM and Co-Array performance on the X1 is similar to MPI performance on the XD1 for small messages. MPI, SHMEM, and Co-Array Fortran achieve equivalent (excellent) performance for large message sizes. SHMEM is available on the XD1 and XT3 as well, but does not have performance superior to that of MPI currently.

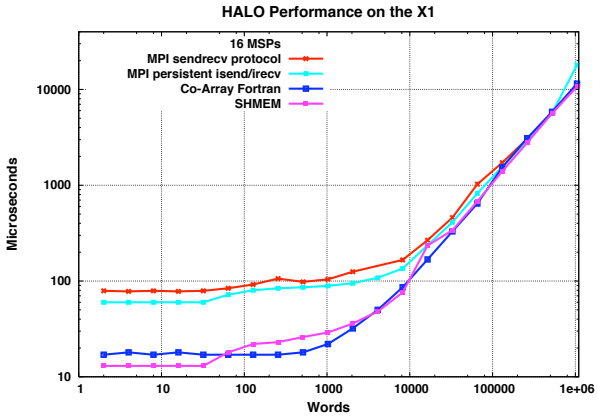


FIGURE 4.2: HALO Alternative Implementations on the X1

Figure 4.3 compares HALO MPI performance on a number of different platforms. Note that the IBM p690 data is for performance within a 32-processor SMP node, so reflects the performance the IBM shared memory implementation of MPI. The high bandwith performance of the X1 and XT3 is again evident, as is the low latency of the XD1. The small message performance on the X1 is significantly worse than what the earlier experiments would have indicated. However, there are buffer copies and other overheads involved in the benchmark, and these may not be vectorized efficiently (if at all) for the small message sizes.

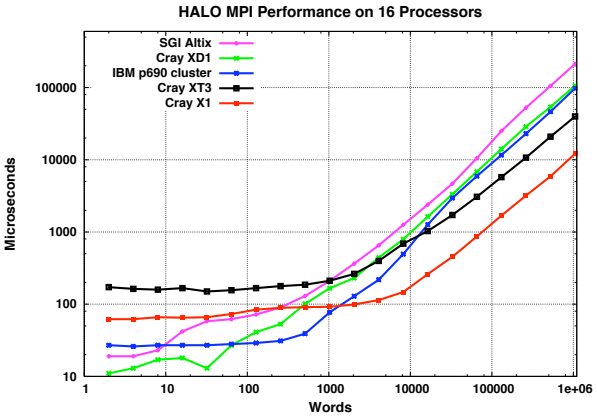


FIGURE 4.3: HALO MPI Performance: Platform Intercomparisons

4.2 Allreduce

The allreduce collective operation is an integral part of many parallel algorithms used in scientific application codes. In a reduction, each process in a group owns a vector whose elements are combined (using some commutative and associative operator) with the corresponding elements owned by other processes, producing a new vector of the same length. An allreduce is a reduction in which the result is replicated on all members of the original group of processes.

Cray X1. Figure 4.4 describes the performance of an allreduce using a doubleword sum operator on the Cray X1 for vectors of length 1, 8192, and 2097152. The first graph compares the performance when using the MPI collective command MPI_Allreduce and the best algorithm from among MPI_Allreduce and a number of point-to-point MPI implementations, demonstrating that MPI_Allreduce is near-optimal in all cases. The second graph examines the performance sensitivity of MPI_Allreduce to the state of the cache. While the performance of MPI point-to-point implementations of collectives are typically not sensitive to the state of the cache, collectives implemented using lower level protocols often are. Here we see that performance is halved when invalidating the cache before the calling MPI_Allreduce for a vector of length 1, but performance in unchanged for large vectors. Note that which experiment is more realistic, with or without cache invalidation, is application specific. The third graph in Fig. 4.4 compares the performance of MPI_Allreduce in November 2003 and May 2005, showing the significant improvement for all three vector lengths. While similar improvements may not be in store for the other,

newer, Cray systems, periodic rexaminations are important when evaluating new systems.

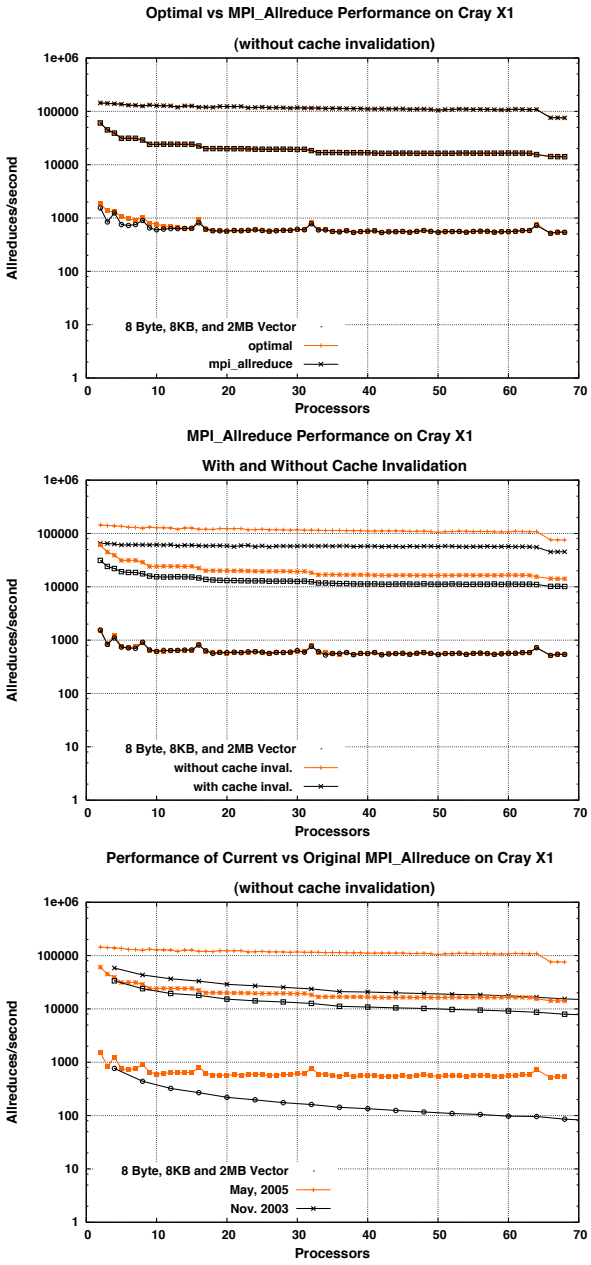


FIGURE 4.4: X1 Allreduce Performance

Cray XD1. Figure 4.5 compares the performance of the "optimal" algorithm with that of MPI_Allreduce on the XD1. Unlike on the X1, MPI_Allreduce is not best for the larger vector lengths. While not atypical, because small vectors are most in applications and optimizations are targeted for these cases, it is our hope that the XD1 implementation of MPI_Allreduce will also be optimized for large vectors in the future.

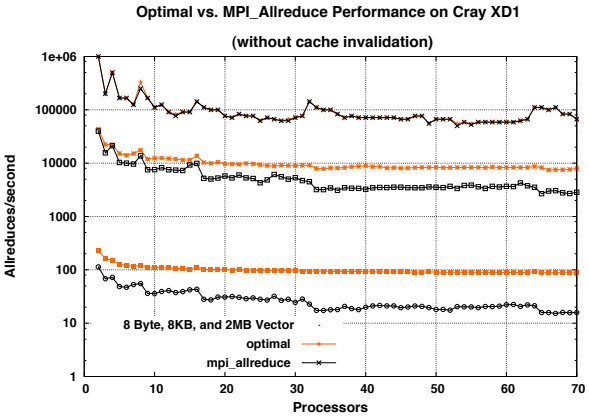


FIGURE 4.5: XD1 Allreduce Performance: Best Point-to-Point vs. MPI_Allreduce

The graphs in Fig. 4.6 compare optimal allreduce and MPI_Allreduce performance with and without cache invalidation, respectively. While there is some impact, it is less than on the X1 and affects performance for the smallest vector length only.

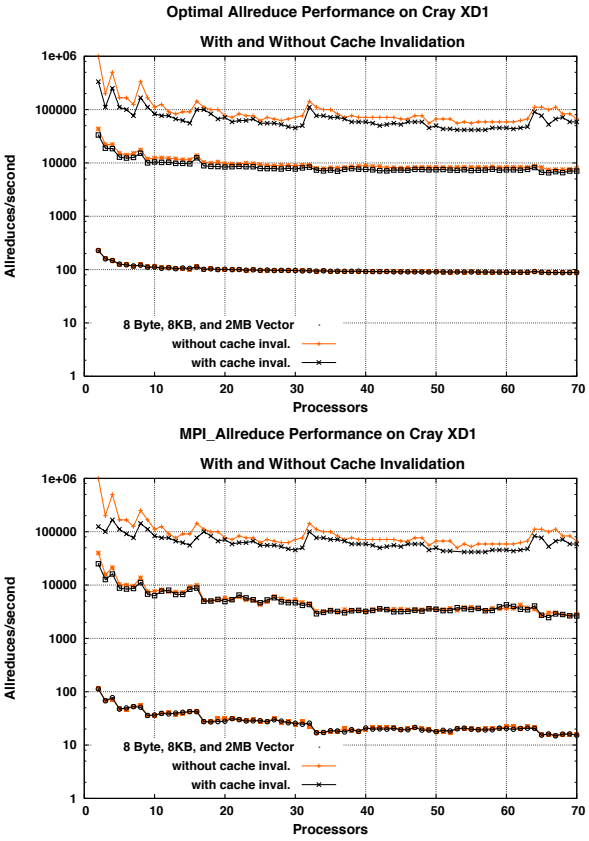


FIGURE 4.6: XD1 Allreduce Performance: With and Without Cache Invalidation

The graphs in Fig. 4.7 compare optimal allreduce and MPI_Allreduce performance when using one or both processors in each node, respectively. There is no difference for MPI_Allreduce. In contrast, the performance of the optimal algorithm is somewhat better when using only one processor per node for the largest vector length. As in the contention experiments, using a shared memory implementation for communication between processors in the same node may change this behavior.

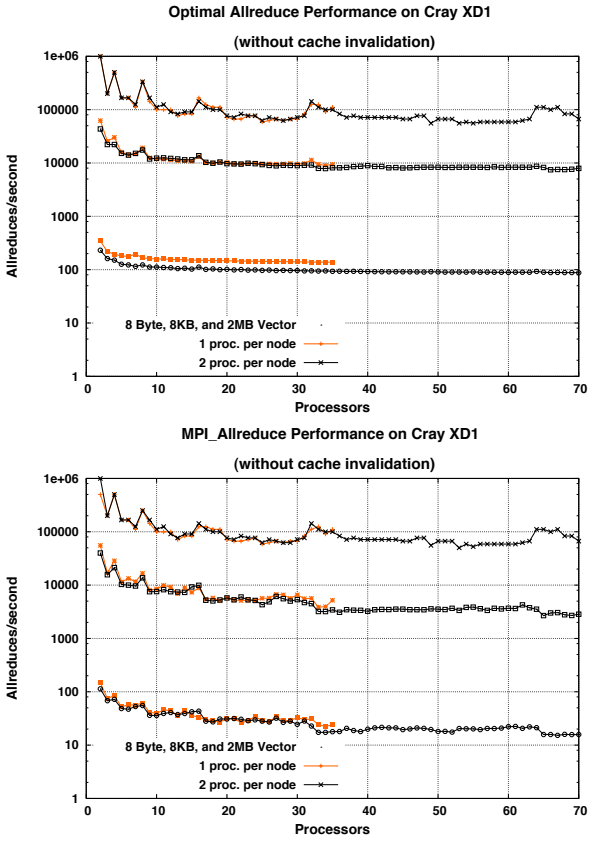


FIGURE 4.7: XD1 Allreduce Performance: One vs. Two Processors per Node

The graphs in Fig. 4.8 compare optimal allreduce and MPI_Allreduce performance when using one or both network fabrics, respectively. As in the previous experiments, there is no difference for MPI_Allreduce and there is a (slight) difference for the optimal algorithm for the larger vector length. (Note that we added experiments for vectors of length 2048 and 524288.)

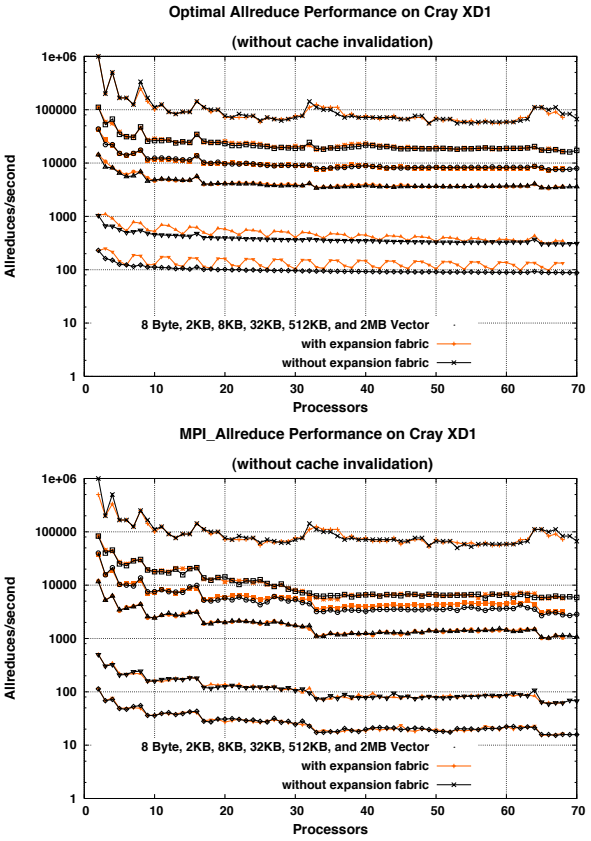
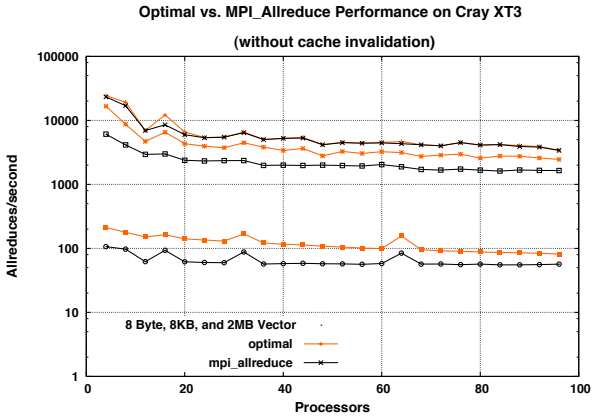
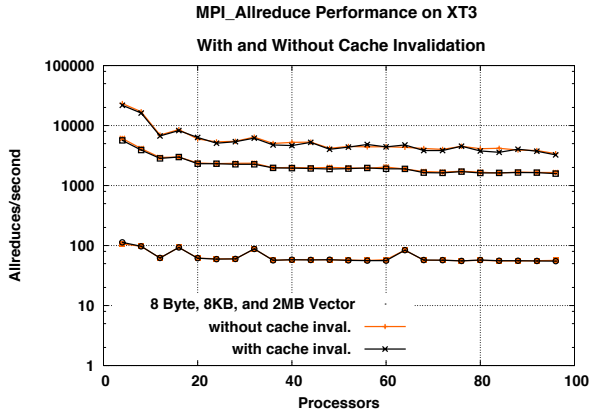


FIGURE 4.8: XD1 Allreduce Performance: With and Without Expansion Fabric

Cray XT3. Figure 4.9 describes the performance of the allreduce on the XT3. The first graph compares the performance of the optimal algorithm and MPI_Allreduce. As on the XD1, MPI_Allreduce is not optimized for large vector lengths currently. The second and third graphs examine the impact of cache invalidation on the performance of MPI_Allreduce and the optimal algorithms, respectively. The state of the cache has no significant performance impact on the XT3 for either allreduce implementation.

Platform Intercomparisons. Figure 4.10 compares the performance of MPI_Allreduce on Cray, SGI, HP, and IBM systems. Note that despite the relatively high MPI latency on the Cray X1, X1 performance for the vector length 1 MPI_Allreduce is best when using more than 16 processors. It is clear that MPI_Allreduce on the X1 is *not* implemented with point-to-point MPI commands, but, rather, uses one of the more efficient messaging layers such as SHMEM or Co-Array Fortran. In contrast, the poor MPI latency on the XT3 is clearly evident in the MPI_Allreduce performance on the XT3.





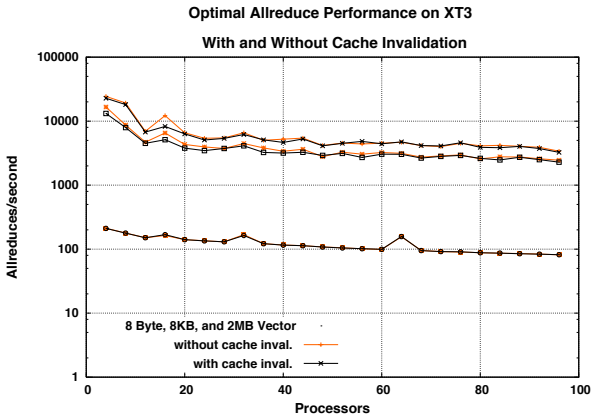
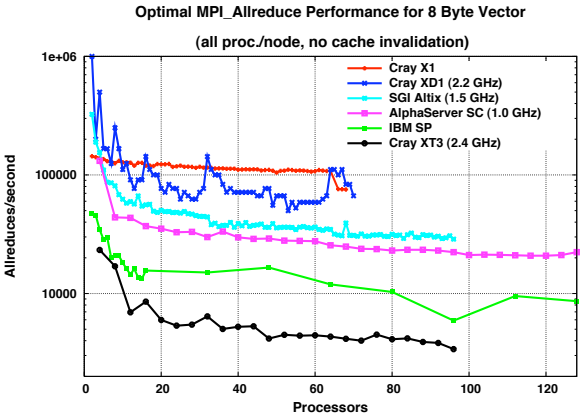
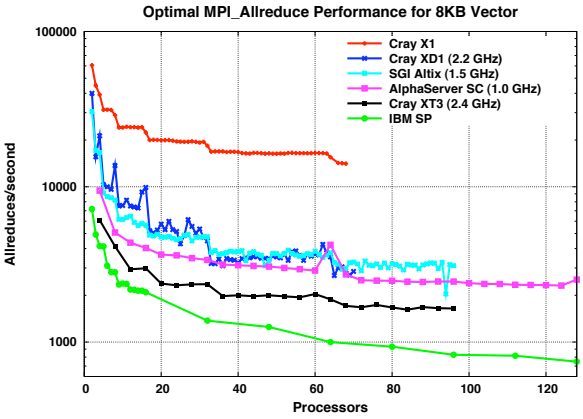


FIGURE 4.9: XT3 Allreduce Performance





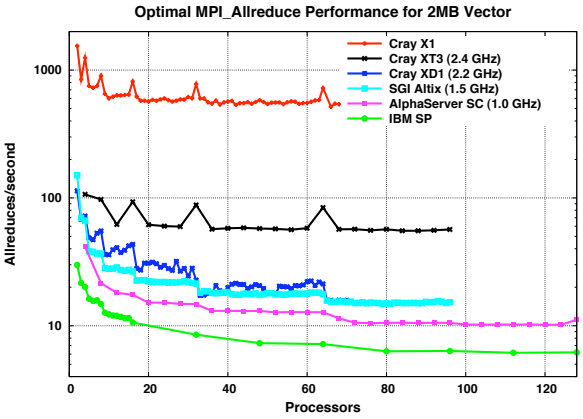


FIGURE 4.10: MPI_Allreduce Performance: Platform Intercomparisons

MPI_Allreduce performance on the X1 is even better compared to the other systems for larger vector lengths, due to both the higer bandwidth interconnect and the superior implementation. Also note that the XT3 MPI_Allreduce performance is superior to all systems except the X1 for the largest vector length, demonstrating excellent network bandwidth. XD1 and Altix performance is again similar, except that XD1 performance is significantly better for the vector length 1 MPI_Allreduce, despite having a larger MPI latency than the Altix.

5 Application Benchmarks

To evaluate the practical import of the microbenchmark results, we examine the performance of two application codes. Here the processor performance will be more important than interconnect performance for small numbers of processors. However, for fixed size problems and large number of processors, the importance of interconnect performance increases.

5.1 POP

The Parallel Ocean Program (POP) [8, 9, 14] is an ocean modeling code developed at Los Alamos National Laboratory (LANL) that is used for high resolution studies and as the ocean component in the Community System Climate Model [3] coupled climate model. POP solves the three-dimensional primitive equations for fluid motions on the sphere under hydrostatic and Boussinesq approximations. Spatial derivatives are computed using finite-difference discretizations.

Time integration of the model is split into two parts, or phases. The three-dimensional vertically-varying (baroclinic) tendencies are integrated explicitly using a leapfrog scheme. The very fast vertically-uniform (barotropic) modes are integrated using an implicit free surface formulation in which a preconditioned conjugate gradient solver is used to solve for the two-dimensional surface pressure.

The parallel implementation is based on a two dimensional (horizontal) domain decompasition of the three-dimensional computational domain. The parallel implementation of the baroclinc phase scales well, requiring significant computation but only limited nearest-neighbor communication. In contrast, the barotropic phase scales poorly. It is dominated by the conjugate gradient solution of the two-dimensional implicit system. This involves very little computation but significant communication. Communication in the barotropic is dominated by halo updates (for residual calculations) and vector length 1 allreduces (single 8 Byte floating point value, for inner product calculations), so is primarily latency sensitive at scale.

In the results that follow, a number of different versions of POP are used. The released version of POP was optimized for nonvector systems. However, in 2003 POP was ported to the Earth Simulator by Dr. Yoshikatsu Yoshida of the Central Research Institute of Electric Power Industry (CRIEPI). In a separate effort, POP was ported to the Cray X1 by John Levesque of Cray, using Co-Array Fortran to implement the halo update and the allreduce in the conjugate gradient solver. The X1 and Earth Simulator ports were merged and further optimized for the X1 by Patrick Worley and James B. White III of ORNL [19].

The version of POP used in these experiments is a pure MPI code (i.e., does not use SMP parallelism) or an MPI/Co-Array Fortran code. In the Cray X1 experiments POP is run with one process per MSP. A single fixed size benchmark is used, characterized by a one degree horizontal grid (320 x 384) and 40 vertical levels. This is referred to as the "by one" or "x1" benchmark problem. The domain decomposition is determined by grid size and the 2D virtual processor grid. Results for a given processor count are the best observed over all applicable processor grids.

Figure 5.1 compares POP performance across the different platforms. The first graph contains data for the Cray systems only, while the second graph includes data for the Earth Simulator and for HP, IBM, and SGI systems. (The Earth Simulator is a cluster of 640 SMP nodes where each node is a variant of the NEC SX-6 made up of 8 vector processors and 16 GB of memory. Each processor is characterized by a peak computational rate of 8 GFlop/s. The nodes are connected by 640x640 single-stage crossbar switches.) Note that two curves are given for the X1, one with a pure MPI implementation and one using both MPI and Co-Array Fortran. Use of Co-Array Fortran allows the X1 to perform significantly better than the other systems, including the Earth Simulator, on what is a very small problem for a vector system, scaling up to 224 processors. In contrast, when using MPI only, performance starts decreasing when using more than 96 processors. The MPI-only performance data was collected in May, 2004, before the better performing MPI_Allreduce described in section 4.2 became available. However, the halo update and allreduce are equally important to the performance scalability of POP, and an improved MPI_Allreduce is not sufficient to correct the scaling problem by itself. Performance on the XD1 appears to be similar to that on the Altix, up to 64 processors. XT3 performance scales to large processor counts, but performance falls below that of the IBM system when using more than 128 processors.

Figure 5.2 compares the time spent in the baroclinic and barotropic phases in the MPI-only and in the MPI with Co-Array Fortran implementations on the X1. The baroclinic times are identical, as they should be, because Co-Array Fortran in used only in the barotropic phase. For the MPI-only implementation the code becomes "baroptropic-bound" (i.e., communication bound) when using 96 or more processors. In contrast, by using Co-Array Fortran, POP continues to be compute bound out to 224 processors.

Figure 5.3 contains baroclinc and barotropic timings for the X1 (using Co-Array Fortran), the XD1, and the XT3. While not as good as on the X1, the barotropic phase on the XD1 scales well out to 64 processors. It is a little surprising that performance is not closer to that of the X1, but the barotropic

may not be communication bound on the XD1 yet. In contrast, the barotropic phase on the XT3 scales even worse than the MPI implementation on the X1, and POP is communication bound on the XT3 when using 128 or more processors.

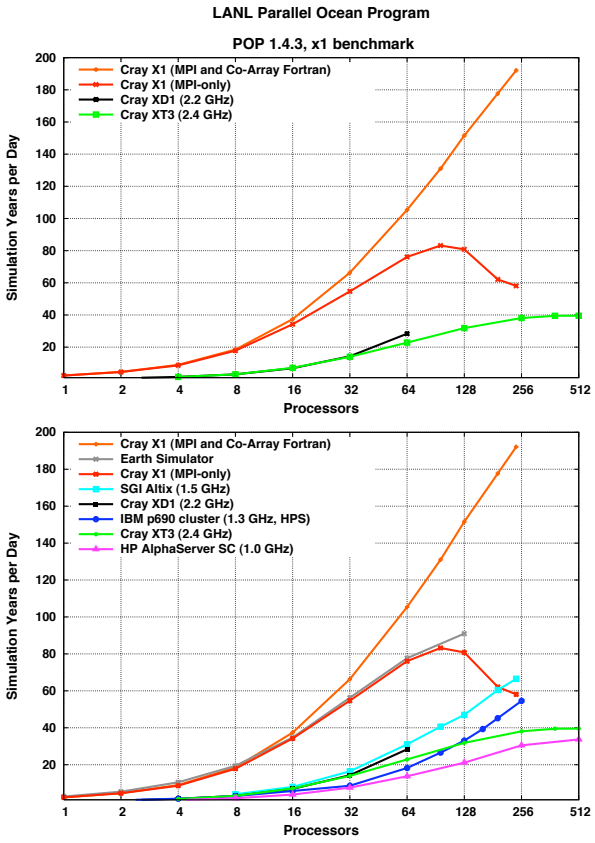


FIGURE 5.1: POP Performance Platform Intercomparisons

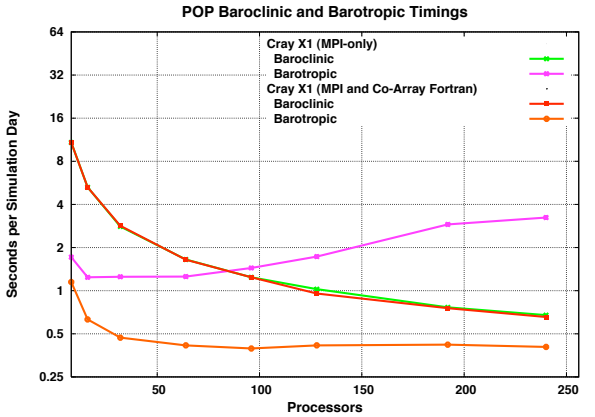


FIGURE 5.2: X1 POP Phase Analysis: With and Without Co-Array Fortran

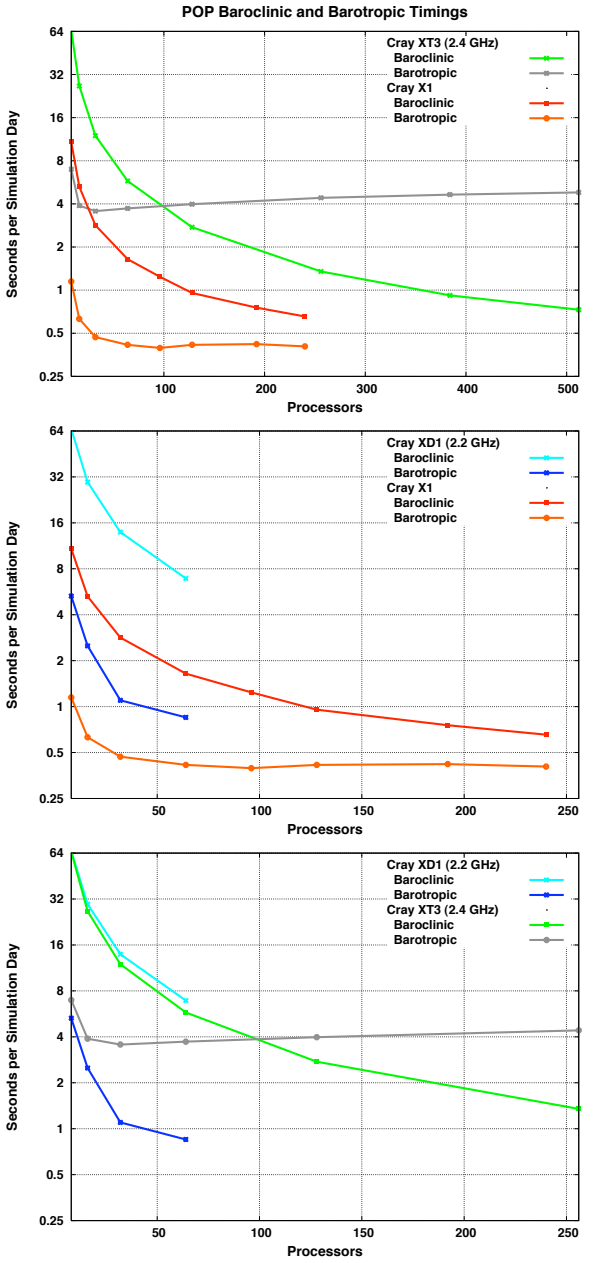


FIGURE 5.3: POP Phase Analysis: X1 vs. XD1 vs. XT3

In summary, it is clear that a small communication latency is required to achieve good scalability for the POP "x1" benchmark. Good performance on the X1 was achieved by using Co-Array Fortran to implement the two collectives: allreduce and halo update. (In future work we will examine the extent to which the new implementation of MPI_Allreduce on the X1 decreases the importance of Co-Array Fortran in achieving good performance for this benchmark.) Good performance on the XT3 will not be possible until MPI (or SHMEM) latency

is decreased. While performance of the barotropic phase on the XD1 is good, the performance of the allreduce and halo update need to be examined more closely to see whether it could be improved further.

5.2 GYRO

GYRO [4] is an Eulerian gyrokinetic-Maxwell solver developed by J. Candy and R.E. Waltz at General Atomics. It is used by researchers worldwide to study plasma microinstabilities and turbulence relevant to controlled fusion research. GYRO comes with ports to a number of different platforms. The port and optimization on the X1 is primarily due to Mark Fahey of ORNL. For X1 experiments, GYRO is run with one process per MSP.

GYRO uses a five-dimensional grid (three spatial and two velocity coordinates) and advances the system in time using a second-order, implicit-explicit Runge-Kutta integrator. Parallelism is based on a domain decomposition of the computational grid. MPI is used for interprocess communication. A simplified view of the GYRO control flow is as follows.

```
do step=1,nstep

   1) domain redistribution
   2) collision term evaluation
   3) domain redistribution

   4) implicit solves

   5) domain redistribution
   6) nonlinear term evaluation
   7) domain reverse redistribution

   8) evaluation of linear terms
   9) implicit solve

   10) domain redistribution
   11) nonlinear term evaluation
   12) domain redistribution

   13) evaluation of linear terms
   14) time-advance

   15) diagnostics and I/O

enddo
```

We refer to the domain redistributions as *transposes*. These are implemented in GYRO using a series of calls to the MPI collective MPI_Alltoall.

In these experiments we examined two different benchmark problems, both time dependent:

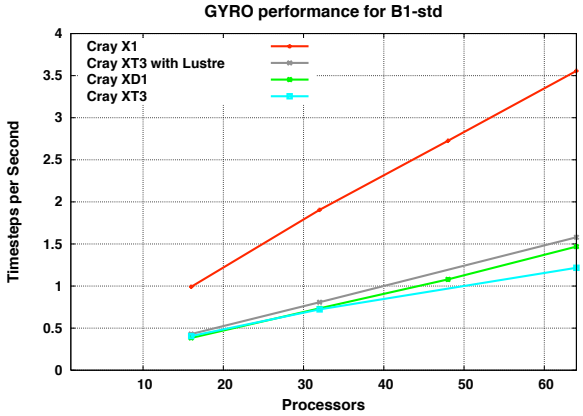
- B1-std. B1-std is the Waltz standard case benchmark [18]. This is a simulation of electrostatic turbulence using parameters characteristic of the DIII-D tokamak at mid-radius. The grid is $16 \times 140 \times 8 \times 8 \times 20$. Since 16 toroidal modes are used, a multiple of 16 processors must be used to run the simulation.

- B3-gtc. B3-gtc is a high-toroidal-resolution electrostatic simulation with simplified electron dynamics. The grid is $64 \times 400 \times 8 \times 8 \times 20$. This case uses 64 toroidal modes, and so must be run on multiples of 64 processors.

Figure 5.4 compares the performance of GYRO for the B1-std problem on the Cray systems, the SGI Altix, and the IBM p690 cluster. The first graph compares performance of the Cray systems only on up to 64 processors. The second graph includes the other systems as well and results up to 512 processors. I/O overhead for these benchmarks is insignificant on all of the platforms except the XT3, where I/O performance is very poor currently. An experimental implementation of the Lustre file system is available on a 96 processor test system. We used results from this system to estimate the performance we would see on the large XT3 system if it had a Lustre file system. (The code is instrumented, and the time spent in I/O is measured, making it simple to reduce to the predicted value.) Figure 5.5 contains the data for the B3-gtc benchmark.

From these data scalability is good on all platforms except the SGI Altix. The X1 continues to show the best performance, especially for the large benchmark where the vector processors provides a clear advantage over the nonvector systems. However, the XT3, after elimination of the I/O performance bottleneck, is the next best performer for both benchmark problems.

Figure 5.6 contains graphs of the fraction of the runtime spent in the transposes used to restructure the domain decomposition during each timestep. From these data it is clear that the SGI performance problem is related to the time spent in these calls to MPI_Alltoall. It is also clear that the performance advantage of the X1 is not just from vectorization. The interconnect performance is also very good on the XT3, especially for the B3-gtc benchmark, which is primarily bandwidth-sensitive. The scalability of the XD1 appears to be good for the B1-std problem, but we need to run the benchmarks on the full 144 processor system in order to understand the XD1 performance scaling behavior.



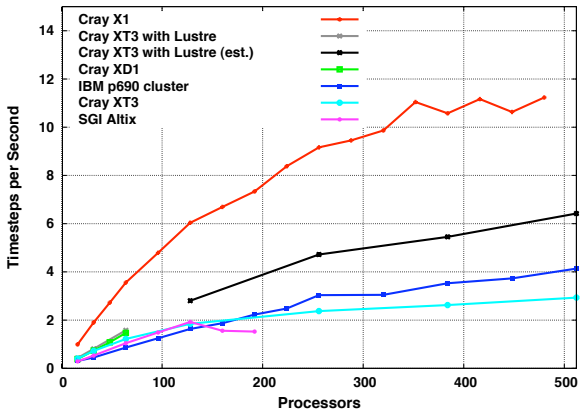


FIGURE 5.4: GYRO B1-std Performance Platform Intercomparisons

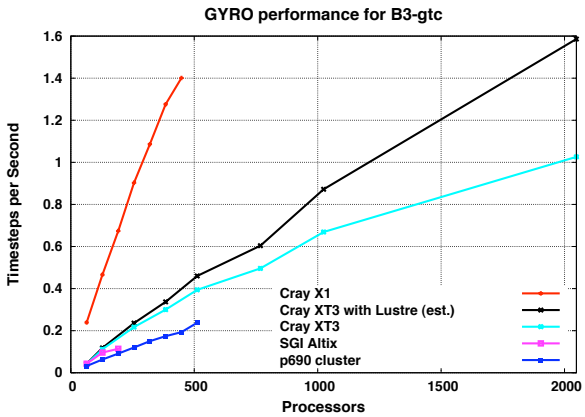
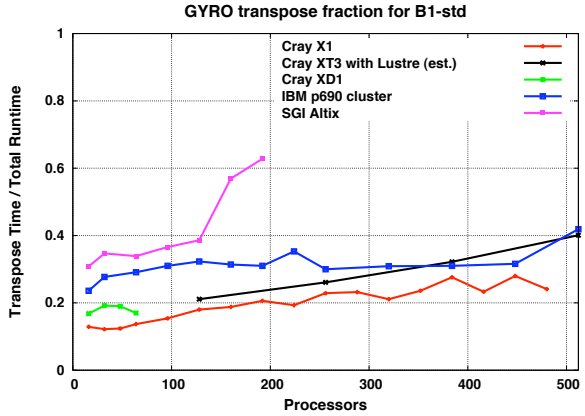


FIGURE 5.5: GYRO B3-gtc Performance Platform Intercomparisons



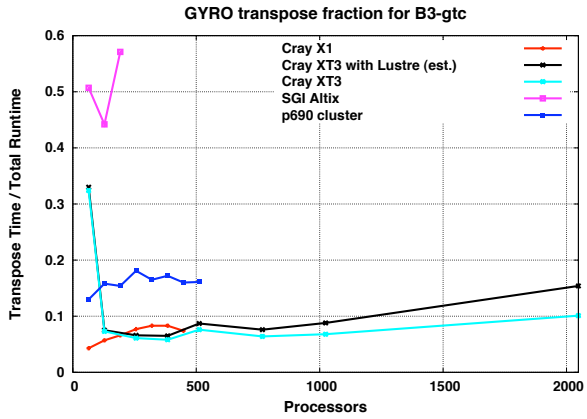


FIGURE 5.6: GYRO Communication Fraction

6 Conclusions

This paper is a snapshot of the performance of three systems in transition. Performance on the XD1 and XT3 should only improve over the next year. In contrast, communication performance as seen by applications is expected to decrease when moving from the X1 to the X1E. However, the X1 communication bandwidth is significantly better than that of the other systems examined in this study, and that advantage will continue on the X1E, though it may not be as large. MPI latency continues to be unimpressive on the X1, but SHMEM and Co-Array Fortran continue to provide excellent workarounds for latency-senstive applications. The performance of at least one of the MPI collectives (MPI_Allreduce) has improved significantly on the X1 over the past year.

Performance results for the XD1 and the XT3 were promising, though performance could be better. MPI on the XD1 achieves very low latency. However, aggregate bandwidth with the direct connect topology is not as good as that on the XT3, and the expansion fabric did not improve performance in our experiments. Initial studies (not described

here) indicate that performance scales poorly when using more than 72 processors with the direct connect topology. A fat tree topology may provide a workaround for this issue.

The latency on the XT3 is extremely high, and point-to-point bandwidth is not better than that on the XD1. However, aggregate bandwidth and scalability (as measured in the contention tests and the collective operator benchmarks with large message sizes and in the large application benchmarks) are very good on the XT3. At least some of the MPI collectives could use additional of optimization on the XD1 and the XT3.

Interestingly, the results from the microbenchmarks, collective operator benchmarks, and application benchmarks were reasonably consistent, and the microbenchmarks results provide some performance prediction capability. This reflects careful engineering on the part of the vendor in the system design. Note, however, the importance in customizing the microbenchmark experiments to examine the unique features of the systems.

7 Acknowledgements

We gratefully acknowledge Nina Hathaway of ORNL and David Londo of Cray for help in using the XD1, Don Maxwell of ORNL and Jeff Becklehimer of Cray for help in using the XT3, Cathy Willis of Cray for help in using the X1, and Phil Roth of ORNL for porting GYRO and collecting data on the SGI Altix. We also thank the Pittsburgh Supercomputer Center for access to the HP AlphaServer SC, the National Energy Research Scientific Computing Center for access to the IBM SP, and the ORNL Center for Computational Sciences (CCS) for access to the Cray X1, Cray XD1, Cray XT3, IBM p690 cluster, and SGI Altix. The CCS is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

8 About the Authors

Sadaf R. Alam is a post-doctoral research associate in the Future Technologies group, Computer Science and Mathematics Division (CSMD) at the Oak Ridge National Laboratory. She has a PhD in Informatics from the University of Edinburgh, United Kingdom. Email: alamsr@ornl.gov.

Thomas H. Dunigan Jr. is a senior R&D staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory. His research interests include the performance characterization and analysis of parallel computers and their communication subsystems; and computer and network security. Dunigan has a PhD in computer science from the University of North Carolina at Chapel Hill. E-mail: dunigan@ornl.gov.

Mark R. Fahey is a senior Scientific Application Analyst in the Center for Computational Sciences (CCS) at Oak Ridge National Laboratory. He is the current CUG X1-Users SIG chair. Mark has a PhD in mathematics from the University of Kentucky. E-Mail: faheymr@ornl.gov.

Jeffrey S. Vetter is a senior R&D staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory, where he leads the Future Technologies Group. His research interests include experimental software systems and architectures for high-end computing. Vetter has a PhD in computer science from the Georgia Institute of Technology. He is a member of IEEE and the ACM. Also, Vetter is an Adjunct Professor in the College of Computing at Georgia Tech. E-mail: vetter@computer.org.

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 ACM and the Society for Industrial and Applied Mathematics. E-mail: worleyph@ornl.gov.

References

[1] P. K. Agarwal, R. A. Alexander, E. Apra, S. Balay, A. S. Bland, J. Colgan, E. F. D'Azevedo, J. J. Dongarra, T. H. Dunigan Jr., M. R. Fahey, R. A. Fahey, A. Geist, M. Gordon, R. J. Harrison, D. Kaushik, M. Krishnakumar, P. Lusczek, A. Mezzacappa, J. A. Nichols, J. Nieplocha, L. Oliker, T. Packwood, M. S. Pindzola, T. C. Shulthess, J. S. Vetter, J. B. White III, T. L. Windus, P. H. Worley, and T. Zacharia, *ORNL Cray X1 evaluation status report*, Tech. Rep. ORNL/TM-2004/13, Oak Ridge National Laboratory, Oak Ridge, TN, January 2004.

[2] AMD, *Software optimization guide for AMD Athlon 64 and AMD Opteron processors*. Technical Manual 25112, 2004.

[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] J. Candy and R. Waltz, *An Eulerian Gyrokinetic-Maxwell solver*, J. Comput. Phys., 186 (2003), p. 545.

[5] Cray Inc., *Cray X1*. http://www.cray.com/products/x1/.

[6] ——, *Cray XD1*. http://www.cray.com/products/xd1/.

[7] ——, *Cray XT3*. http://www.cray.com/products/xt3/.

[8] D. K. Dukowicz and R. D. Smith, *Implicit free-surface method for the Bryan-Cox-Semtner ocean model*, J. Geophys. Res., 99 (1994), pp. 7991–8014.

[9] D. K. Dukowicz, R. D. Smith, and R. C. Malone, *A reformulation and implementation of the Bryan-Cox-Semtner ocean model on the connection machine*, J. Atmos. Ocean. Tech., 10 (1993), pp. 195–208.

[10] 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 SC2003 Conference, November 15-21, 2003, D. Lora, ed., New York, NY, 2003, Association for Computing Machinery.

[11] T. H. Dunigan Jr., J. S. Vetter, J. B. White III, and P. H. Worley, *Performance evaluation of the Cray X1 distributed shared memory architecture*, IEEE Micro, 25 (2005), pp. 30–40.

[12] M. R. Fahey, S. R. 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., Cray User Group, Inc.

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

[14] P. W. Jones, *The Los Alamos Parallel Ocean Program (POP) and coupled model on MPP and clustered SMP computers*, in Making its Mark – The Use of Parallel Processors in Meteorology: Proceedings of the Seventh ECMWF Workshop on Use of Parallel Processors in Meteorology, G.-R. Hoffman and N. Kreitz, eds., World Scientific Publishing Co. Pte. Ltd., Singapore, 1999.

[15] R. W. Numrich and J. K. Reid, *Co-Array Fortran for parallel programming*, ACM Fortran Forum, 17 (1998), pp. 1–31.

[16] 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., Cray User Group, Inc.

[17] 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.

[18] R. Waltz, G. Kerbel, and J. Milovich, *Toroidal gyro-landau fluid model turbulence simulations in a nonlinear ballooning mode representation with radial modes*, Phys. Plasmas, 1 (1994), p. 2229.

[19] 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., Cray User Group, Inc.