

Performance Evaluation of the Cray X1 Distributed Shared Memory Architecture¹

Thomas H. Dunigan, Jr., Jeffrey S. Vetter, James B. White III, Patrick H. Worley
Oak Ridge National Laboratory

Abstract—The Cray X1 supercomputer is a distributed shared memory vector multiprocessor, scalable to 4096 processors and up to 65 terabytes of memory. The X1’s hierarchical design uses the basic building block of the multi-streaming processor (MSP), which is capable of 12.8 GF/s for 64-bit operations. The distributed shared memory (DSM) of the X1 presents a 64-bit global address space that is directly addressable from every MSP with an interconnect bandwidth per computation rate of one byte per floating point operation. Our results show that this high bandwidth and low latency for remote memory accesses translates into improved application performance on important applications. Furthermore, this architecture naturally supports programming models like the Cray SHMEM API, Unified Parallel C, and Co-Array Fortran. It can be important to select the appropriate models to exploit these features, as our benchmarks demonstrate.

I. INTRODUCTION

The Cray X1 supercomputer, introduced in 2002, has several interesting architectural features. Two key features are the X1’s distributed shared memory and its vector multiprocessors. Recent studies of the X1’s vector multiprocessors have shown significant performance improvements on several applications [1, 7]. In this paper, we characterize the performance of the X1’s distributed shared-memory system and its interconnection network using microbenchmarks and applications.

The X1’s distributed shared-memory architecture presents a 64-bit global address space, which is directly addressable from every processor using traditional load and store instructions. From the application perspective, this memory system behaves like a nonuniform memory access (NUMA) architecture; however, this memory system does not cache accesses between symmetric multiprocessor (SMP) nodes. This hardware support for global addressability naturally supports programming models such as the Cray SHMEM API [2], Unified Parallel C (UPC) [4], Co-Array Fortran [12], and Global Arrays [11].

II. CRAY X1 OVERVIEW

The Cray X1 is an attempt to incorporate the best aspects of previous Cray vector systems and massively parallel processing systems into one design. Like the Cray T90, the X1 has high memory bandwidth, which is key to realizing a high percentage of theoretical peak performance. Like the Cray T3E [14], the X1 has a high-bandwidth, low-latency, scalable

interconnect, and scalable system software. And, like the Cray SV1, the X1 leverages commodity CMOS technology and incorporates non-traditional vector concepts, such as vector caches and multi-streaming processors (MSPs).

A. Multi-streaming Processor

The X1 has a hierarchical design with an MSP basic building block capable of 12.8 Gflops/s for 64-bit operations (or 25.6 Gflops/s for 32-bit operations). As Figure 1 illustrates, each MSP consists 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 Gflops/s for 64-bit operations. The four SSPs share a 2 Mbyte *Ecache*.

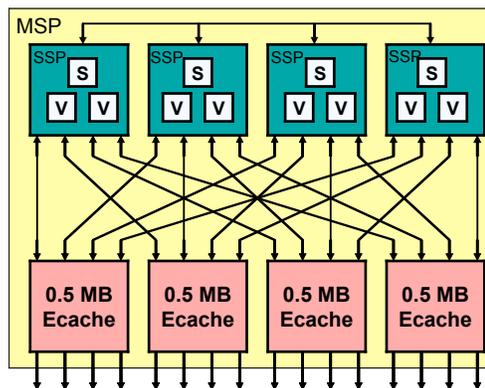


Figure 1: Cray MSP module.

Although the *Ecache* has sufficient single-stride bandwidth (accessing consecutive memory locations) to saturate the vector units of the MSP, the *Ecache* is necessary because the bandwidth to main memory is insufficient to saturate the vector units without data reuse. That is, memory bandwidth is roughly half the saturation bandwidth. This design represents a compromise between non-vector-cache systems, such as the NEC SX-6, and cache-dependent systems, such as the IBM p690, which has memory bandwidths that are an order of magnitude less than the saturation bandwidth. The X1, because of its short cache lines and extra cache bandwidth, has a random-stride scatter/gather memory access that is just a factor of two slower than stride-one access, not the factor of eight or more typical of cache-based systems like those based on the IBM Power4, Compaq Alpha, or Intel Itanium. The X1’s cache-based design only deviates slightly from the full-bandwidth design model. Each X1 MSP has the single-stride

¹ This paper was accepted for publication and the final (edited, revised and typeset) version of the paper was published in IEEE Micro, Volume 25, No. 1, January/February 2005 by the IEEE Computer Society.

bandwidth of an SX-6 processor; it is the X1's higher peak performance that creates an imbalance. A relatively small amount of data reuse, which most modern scientific applications do exhibit, can enable the X1 to realize a very high percentage of peak performance, and even during worst-case data access, data reuse can still provide double-digit efficiencies.

The X1 compiler has two options for using the eight vector units of a single MSP. First, it can use all 8 when vectorizing a single loop. Second, it can split up (or multistream) the work in an unvectorized outer loop and assign it to the four SSPs, each with two vector units and one scalar unit. (The compiler can also vectorize a "long" outer loop and multistream a shorter inner loop if the dependency analysis allows it.)

The effective vector length of the first option is 256 elements, the vector length of the NEC SX-6. The second option, which attacks parallelism at a different level, allows a shorter vector length of 64 elements for a vectorized loop. Cray also supports the option of treating each SSP as a separate processor.

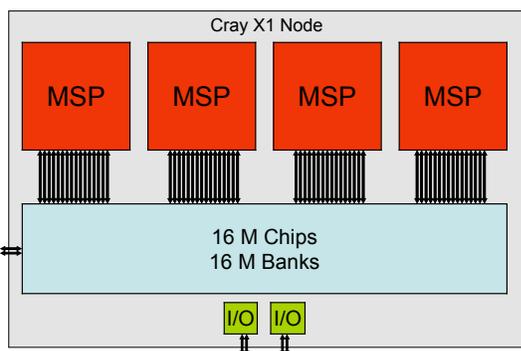


Figure 2: Cray X1 node.

As Figure 2 illustrates, four MSPs, 16 memory controller chips (M-chips), and 32 memory daughter cards form a Cray X1 node. A node's memory banks provide 204 Gbytes/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. These banks have error-correcting-code memories, which provide reliability by correcting single-bit errors, detecting multiple-bit errors, and providing chip-kill error detection.

Each bank of shared memory connects to several banks on remote nodes, with an aggregate bandwidth of roughly 50 Gbytes/s between nodes. This balance represents one byte per floating-point operation (flop) of interconnect bandwidth per computation rate, compared to 0.25 bytes per flop on the Japanese Earth Simulator [15], and less than 0.1 bytes per flop on an IBM p690 with the maximum number of High Performance Switch (HPS) connections [9].

B. Interconnect Overview

X1 routing modules connect the Cray X1 nodes. Each node has 32 1.6 Gbytes/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 in this way form a processor *stack*. The local memory bandwidth per node is 204 Gbytes/s, enough to service both local and remote memory requests.

An X1 cabinet consists of 16 node boards and four routing boards (or two processor stacks). Each routing board has eight routing modules. The routing module ASIC is an eight-way nonblocking crossbar switch supporting worm hole routing. The routing module supports prioritization based on credits or aging. Ports connect to the node boards or other router ports with 96-pin cables with a maximum length of 4 meters. Data packets carry a cyclic redundancy code (CRC), and if the receiver detects a CRC error, the sending node resends the packet. Communication latency increases by about 500 ns per router hop. The X1 routing module uses software-loaded configuration tables for data flow mapping across the interconnection network. At system boot, these tables are initialized, but are reloadable, providing a means to reconfigure the network around hardware failures.

Interstack connectivity allows several options. First, a four-node X1 can interconnect directly via the memory modules links. Second, with eight or fewer cabinets (up to 128 nodes or 512 MSPs), the interconnect topology is a 4D hypercube. Larger configurations use an enhanced 3D torus, where one dimension of the torus, the processor stack, is fully connected.

The 3D torus topology has relatively low bisection bandwidth compared to crossbar-style interconnects [6], such as those on the IBM SP and the Earth Simulator. Whereas bisection bandwidth scales as the number of nodes, $O(n)$, for crossbar-style interconnects, it scales as the $2/3$ root of the number of nodes, $O(n^{2/3})$, for a 3D torus. Despite this theoretical limitation, mesh-based systems - such as the Intel Paragon, the Cray T3E, and ASCI Red - have scaled to thousands of processors.

Atomic in-memory operations (submicrosecond scalable locks and barriers) provide synchronization [5]. In particular, the X1 provides explicit memory ordering instructions for local ordering (LSYNC), MSP ordering (MSYNC), and global ordering (GSYNC). It also provides the basic atomic memory operations like `fetch&op`. Although these operations are efficient because they do not require a cache-line of data, they are not ordered with respect to other memory references and require synchronization using memory ordering instructions.

C. Local and Remote Memory Accesses

A single four-MSP X1 node behaves like a traditional SMP. Like the T3E, each processor has the additional capability of directly addressing memory on any other node. Different, however, is the fact that the processors directly issue these remote memory accesses as load and store instructions, which go transparently over the X1 interconnect to the target processor, bypassing the local cache. This mechanism is more scalable than traditional shared memory, but it is not appropriate for shared-memory programming models, such as OpenMP [13], outside of a given four-MSP node. This remote-memory access mechanism is a natural match for distributed-memory programming models, particularly those using one-sided put/get operations.

As Figure 3 shows, the X1 64-bit global virtual address decomposes into two parts: two bits to select the memory region and 48 bits for a virtual page number, page boundaries, and page offset. The page size can range from 64Kbytes to 4 Gbytes, selectable at execution time with different page sizes possible for text and data areas.

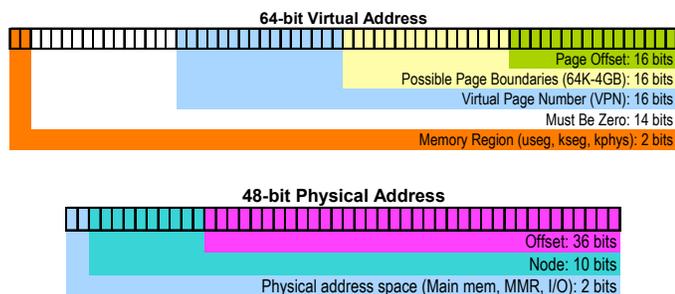


Figure 3: Cray X1 address translation.

The 48-bit physical address decomposes into a 2-bit physical-address region marker, a 10-bit node number, and a 36-bit offset. The 10-bit node number limits the maximum X1 configuration to 1,024 nodes (4,096 MSPs). The address translation scheme uses 256-entry translation look-aside buffers (TLBs) on each node and allows noncontiguous multinode jobs (though this mode typically degrades performance). When a job uses contiguously numbered nodes, it is possible to remotely translate page offsets, so the TLB needs to hold translations for just one node. This design scheme allows the system to scale with the number of nodes with no additional TLB misses. Such a design can hide memory latency with the compiler’s help; the hardware dynamically unrolls loops, performs scalar and vector renaming, and issues scalar and vector loads early. Vector load buffers permit 2,048 outstanding loads for each MSP. Nonallocating references can bypass the cache for remote communication to avoid cache pollution and to provide efficient large-stride (or scatter/gather) support.

III. PERFORMANCE

This section describes some of our results in evaluating the Cray X1 and its memory hierarchy. We conducted these tests on the eight-cabinet, 512-MSP X1 located at Oak Ridge National Laboratory (ORNL). Our evaluation uses both standard and custom benchmarks as well as application kernels and full applications. Table 1 provides the basic configurations of each platform used in this experimental evaluation.

A. Programming models

An X1 node (four MSPs) supports a cache-coherent shared memory, and Cray supports OpenMP, System V shared memory, and POSIX threads shared-memory programming models. In addition, the compilers can treat the node processors as four streaming MSP’s (in MSP mode) or 16 individual SSPs (in SSP mode). Each node can have from 8 to 32 Gbytes of local memory.

Cray supports several distributed-memory programming

models for the X1, including the Message Passing Interface (MPI) [16], SHMEM, Co-Array Fortran, and UPC. For MPI message passing, the minimum addressable unit is an MSP (or an SSP if the job is compiled in SSP mode). For UPC and Co-Array Fortran, the compiler can overlap computation with remote memory requests because the decoupled micro-architecture allows the scalar unit to prepare operands and addresses for the vector unit.

The programmer can mix node-level SMP with both MPI and direct access (SHMEM, UPC, or Co-Array Fortran) to remote memory. Hardware handles synchronization (locks and barriers). Exploiting this diverse set of programming models is one of the X1’s opportunities.

The compilers also provide directives to assist in parallelization and external memory management (e.g., no caching for designated variables). Scientific libraries provide efficient management of the Ecache and vector pipes. The user can specify page size for text and data areas when initiating an executable. The resource management system provides processor allocation, job migration, and batch scheduling.

B. Microbenchmarks

We use a collection of microbenchmarks to characterize the performance of the underlying hardware, compilers, and software libraries. The STREAM [10] triad memory bandwidth is 24 Gbytes/s for a streaming MSP or 40 Gbytes/s (aggregate) for 4 SSPs. The aggregate STREAM triad memory bandwidth for an X1 SMP node is 84 Gbytes/s for 4 MSPs and 90 Gbytes/s for 16 SSPs. This compares favorably with the 30 Gbytes/s bandwidth for one processor of the modified NEC SX-6 used in the Earth Simulator, and 213 Gbytes/s for an eight-processor SMP. Remote memory access bandwidth peaks at about 30 Gbytes/s for the X1 (using Co-Array Fortran).

Table 1: Platform Configurations.

	SGI Altix	Alpha SC	Earth Simulator	IBM SP4	Cray X1
Proc	Itanium 2	Alpha EV67	NEC SX-6	POWER4	Cray X1
Interconnect	Numalink	Quadrics (Elan3)	custom crossbar	HPS or SP Switch2	Cray X1
MHz	1500	667	500	1300	800
Mem/Node	512GB	2GB	16GB	32GB	16GB
L1	32K	64K	n/a	32K	16K (scalar)
L2	256K	8MB	n/a	1.5MB	2MB (per MSP)
L3	6MB	n/a	n/a	128MB	n/a
Proc Peak Mflops	6000	1334	8000	5200	12800
Peak mem BW	6.4 GB/s	5.2GB/s	32GB/s/processor	51GB/s/MCM	26GB/s/MSP

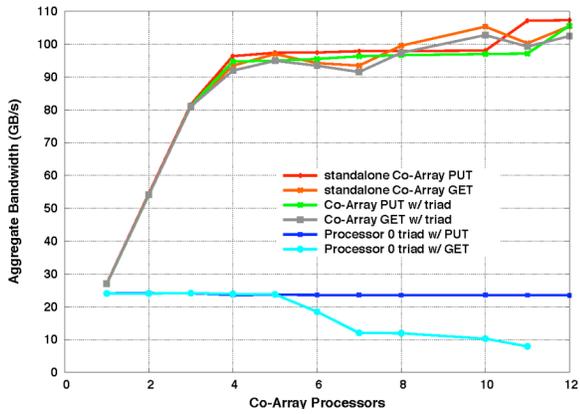


Figure 4: Stream triad with Co-Array Traffic.

Figure 4 illustrates the effect of remote accesses on local-memory performance. Processor 0 is executing a STREAM triad. With no memory interference, the triad runs at 24 Gbytes/second. The figure shows the effect of an increasing number of processors doing Co-Array FORTRAN gets from or puts to processor 0. If more than five processors are executing gets, it reduces triad performance, but puts have no effect on triad performance. The local-memory activity (triad) has little effect on the aggregate throughput of the gets and puts.

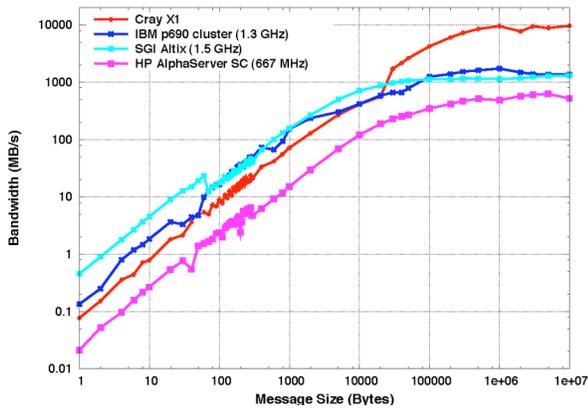


Figure 5: MPI intra-node bandwidth.

Figure 5 and Figure 6 show the MPI intra-node and internode bandwidths. We used the ParkBench *comms1* benchmark code to measure MPI communication performance between two processors on the same node and then two different nodes. MPI latency was 7.3 microseconds (one-way) for an 8-byte message between X1 nodes. Each additional hop in the torus network requires less than 0.5 microseconds. MPI bandwidth for ping-pong reaches 12 Gbytes/s between nodes. The X1 demonstrates a significant advantage over the other platforms when message sizes rise above 8Kbytes.

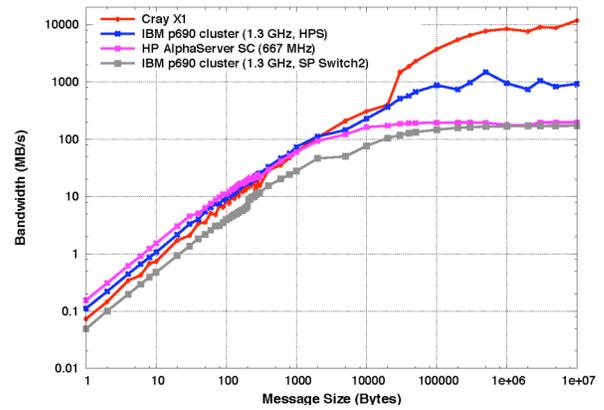


Figure 6: MPI inter-node bandwidth.

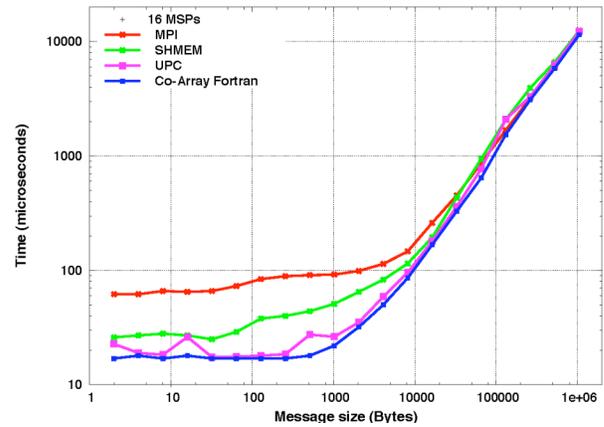


Figure 7: HALO exchange timings.

MPI is not yet fully optimized for the X1, and SHMEM and Co-Array Fortran usually perform better for small message sizes. Figure 7 shows how the various X1 programming paradigms perform a HALO operation [17] on 16 MSPs. The HALO benchmark simulates the nearest neighbor exchange of a 1 to 2 row/column “halo” from a 2D array. This is a common operation when using domain decomposition. Latency dominates small message performance, whereas bandwidth limits the performance for larger messages. The Co-Array paradigm performs the best, partially because the compiler can hide some of the latency.

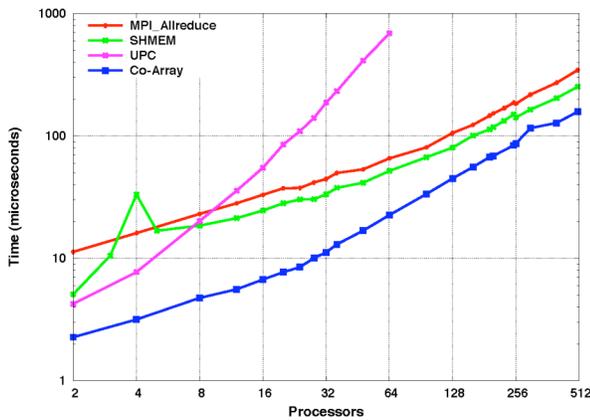


Figure 8: Allreduce Latency.

Figure 8 illustrates the time to perform an allreduce—a common operation in scientific applications—using a double-word sum operator implemented in various programming paradigms. For the Co-Array Fortran, SHMEM, and UPC implementations, the algorithm gathered data to a single process, summed it, then broadcasted it. MPI_Allreduce may use a different algorithm. As with the HALO operation, the Co-Array Fortran implementation performed the best, and Cray has not yet optimized the UPC performance. Viewed in this light, it is clear that choosing the appropriate programming paradigm can be important to efficiently using the underlying hardware. However, barriers for the various programming models use the same underlying hardware and average about 5 microseconds, essentially independent of the number of participating processors at the current scale (up to 512 MSPs).

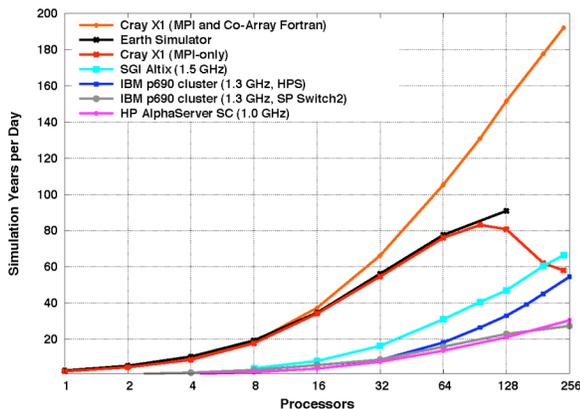


Figure 9: Performance of the LANL Parallel Ocean Program (POP 1.4.3).

C. Applications

These impressive performance results for microbenchmarks on the X1 are uninteresting unless they also translate into performance improvements in applications. Two such application areas at ORNL are climate modeling and fusion simulations.

Climate Modeling

The Parallel Ocean Program (POP) [8] is an ocean

modeling code developed at Los Alamos National Laboratory (LANL); it serves as the ocean component in the Community Climate System Model coupled climate model. Figure 9 compares the performance of this code on the X1 when using a pure MPI implementation and when using Co-Array Fortran for two routines: a halo update and an allreduce. Both routines are used in a conjugate gradient linear system solver: the halo update in calculating residuals and the allreduce in calculating inner products. Figure 9 also shows performance on a Hewlett-Packard AlphaServer SC, an IBM p690 cluster, the Earth Simulator, and an SGI Altix. POP’s performance scalability is very sensitive to latency, and MPI latency limits performance on the Cray X1 compared to that achievable using Co-Array Fortran.

Fusion Simulation

GYRO [3] is an Eulerian gyrokinetic-Maxwell solver developed by R.E. Waltz and J. Candy at General Atomics. It is used to study plasma microturbulence in fusion research. Figure 10 compares the performance of GYRO on the X1, the SGI Altix, and an IBM p690 cluster using both SP Switch2 and High Performance Switch (HPS) interconnects. GYRO uses the MPI_ALLTOALL command to transpose the distributed data structures; it is more sensitive to bandwidth than to latency. As Figure 11 shows, the IBM results indicate the sensitivity of performance to bandwidth, because the primary difference in performance between the SP Switch2 and HPS results is in message-passing performance. For this benchmark, MPI bandwidth on the X1 does not limit scalability.

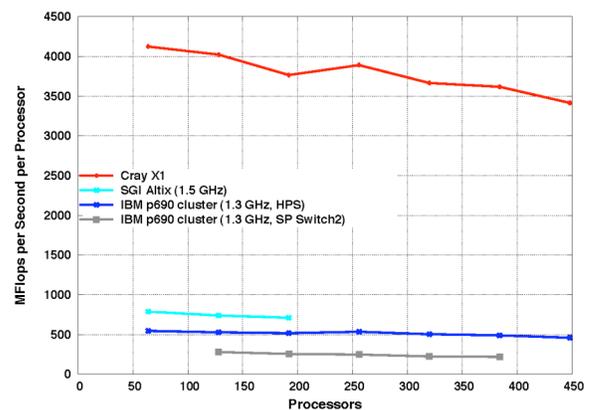


Figure 10: Performance of the GYRO Eulerian Gyrokinetic-Maxwell Solver (64-mode GTC benchmark).

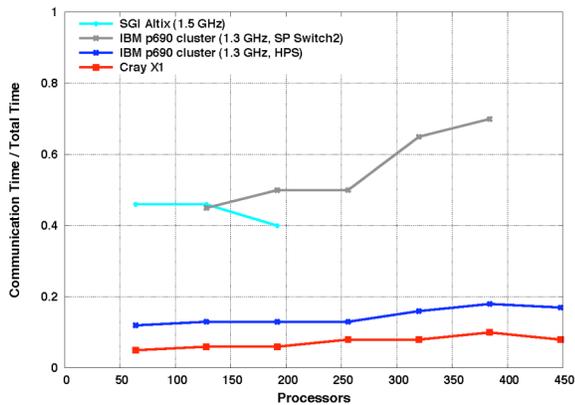


Figure 11: Communication Fraction for the GYRO Eulerian Gyrokinetic-Maxwell Solver (64-mode GTC benchmark).

IV. CONCLUSION

Our experiments show that the high bandwidth and the low latency for X1 interconnect translates into improved application performance on diverse applications, such as the POP ocean model and the GYRO gyrokinetic-Maxwell solver. Our benchmarks also demonstrate that it can be important to select the appropriate programming models to exploit these benefits. For the most recent results and additional performance data comparing the X1 with other systems, see www.ccs.ornl.gov/evaluation.

We plan to continue our investigations of other core technologies for high-performance computing, which will include future generations of Cray systems, including the X1E and Black Widow. Most importantly, we plan to investigate next-generation interconnects, such as Infiniband, and the proprietary interconnects of the Cray XD1, the Cray XT3, and the Cray Rainier architectures.

ACKNOWLEDGEMENTS

We gratefully acknowledge Cray Inc. for its ongoing cooperation, and, in particular, Steve Scott, James Schwarzmeier, and Nathan Wichmann. The work described in this paper 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-Battelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

REFERENCES

[1] P.A. Agarwal, R.A. Alexander *et al.*, “Cray X1 Evaluation Status Report,” ORNL, Oak Ridge, TN, Technical Report ORNL/TM-2004/13, 2004, <http://www.csm.ornl.gov/evaluation/PHOENIX/PDF/CRAYEvaluationTM2004-15.pdf>.

[2] R. Barriuso and A. Knies, “SHMEM Users Guide,” Cray Research, Inc. 1994.

[3] J. Candy and R. Waltz, “An Eulerian gyrokinetic-Maxwell solver,” *J. Comput. Phys.*, 186(545), 2003.

[4] W.W. Carlson, J.M. Draper *et al.*, “Introduction to UPC and language specification,” Center for Computing Sciences, IDA, Bowie, MD, Technical Report CCS-TR-99-157, 1999.

[5] Cray Incorporated, “Cray X1 System Overview,” Cray Incorporated, Mendota Heights, MN, Technical Manual S-2346-22, 2002.

[6] W.J. Dally and B. Towles, *Principles and practices of interconnection networks*. San Francisco: Morgan Kaufmann Publishers, 2003.

[7] T.H. Dunigan, Jr., M.R. Fahey *et al.*, “Early Evaluation of the Cray X1,” Proc. ACM/IEEE Conference on High Performance Networking and Computing (SC03), 2003.

[8] P.W. Jones, P.H. Worley *et al.*, “Practical performance portability in the Parallel Ocean Program (POP),” *Concurrency and Computation: Experience and Practice*(in press), 2004.

[9] O. Lascu, Z. Borgosz *et al.*, “An Introduction to the New IBM pSeries High Performance Switch,” IBM Corporation, IBM Redbook SG24-6978-00, 2004.

[10] J.D. McCalpin, *Stream Benchmarks*, <http://www.cs.virginia.edu/stream>, 2002.

[11] J. Nieplocha, R.J. Harrison, and R.J. Littlefield, “Global Arrays: A portable shared memory model for distributed memory computers,” Proc. Supercomputing 94, 1994, pp. 340-9.

[12] R.W. Numrich and J. Reid, “Co-Array Fortran for parallel programming,” *ACM SIGPLAN FORTRAN Forum*, 17(1998):1-31, 1998.

[13] OpenMP, *OpenMP Reference*, <http://www.openmp.org>, 1999.

[14] S.L. Scott, “Synchronization and Communication in the T3E Multiprocessor,” Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS), 1996, pp. 26-36.

[15] S. Shingu, Y. Tsuda *et al.*, “A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator,” Proc. SC2002, 2002.

[16] M. Snir, S. Otto *et al.*, Eds., *MPI--the complete reference*, 2nd ed. Cambridge, MA: MIT Press, 1998.

[17] A.J. Wallcraft, “The NRL Layered Ocean Model Users Guide,” Naval Research Laboratory, Stennis Space Center, MS NOARL Report 35, 1991.