Performance Evaluation of the Cray X1 Distributed Shared-Memory Architecture

THE CRAY X1 SUPERCOMPUTER'S DISTRIBUTED SHARED MEMORY PRESENTS A 64-BIT GLOBAL ADDRESS SPACE THAT IS DIRECTLY ADDRESSABLE FROM EVERY MSP WITH AN INTERCONNECT BANDWIDTH PER COMPUTATION RATE OF 1 BYTE/FLOP. OUR RESULTS SHOW THAT THIS HIGH BANDWIDTH AND LOW LATENCY FOR REMOTE MEMORY ACCESSES TRANSLATE INTO IMPROVED APPLICATION PERFORMANCE ON IMPORTANT APPLICATIONS.

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

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.[1,2] In this article, 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 nodes. This hardware support for global addressability naturally supports programming models such as the Cray Shmem API,[3] Unified Parallel C (UPC),[4] Co-Array Fortran,[5] and Global Arrays.[6]

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,[7] 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 nontraditional vector concepts, such as vector caches and multistreaming processors (MSPs).

Multistreaming 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-

Published by the IEEE Computer Society

0272-1732/05/$20.00 © 2005 IEEE

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*.

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 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 eight 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.

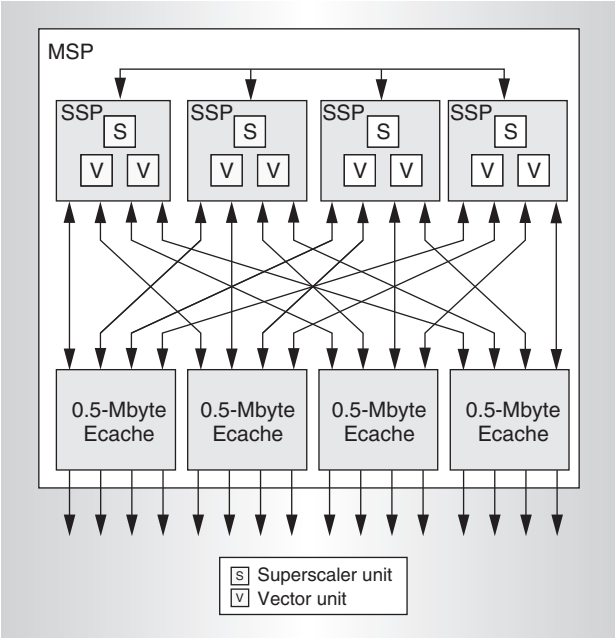


Figure 1. Cray MSP module.

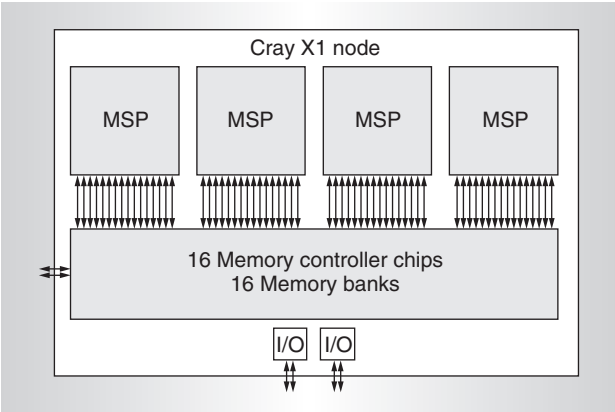


Figure 2. Cray X1 node.

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,[8] and less than 0.1 bytes per flop on an IBM p690 with the maximum number of High-Performance Switch (HPS) connections.[9]

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,[10] 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 (fast, submicrosecond, scalable locks and barriers) provide synchronization.[11] In particular, the X1 provides explicit memory ordering instructions for local ordering (Lsync), MSP ordering (Msync), and global ordering (Gsync). It also provides basic atomic memory operations such as fetch&op. Although these operations are efficient because they do not require a cache line of data, they are unordered with respect to other memory references and require synchronization using memory ordering instructions.

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 (http://www.openmp.org), 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 64 Kbytes to 4 Gbytes, selectable at execution time with different page sizes possible for text and data areas.

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

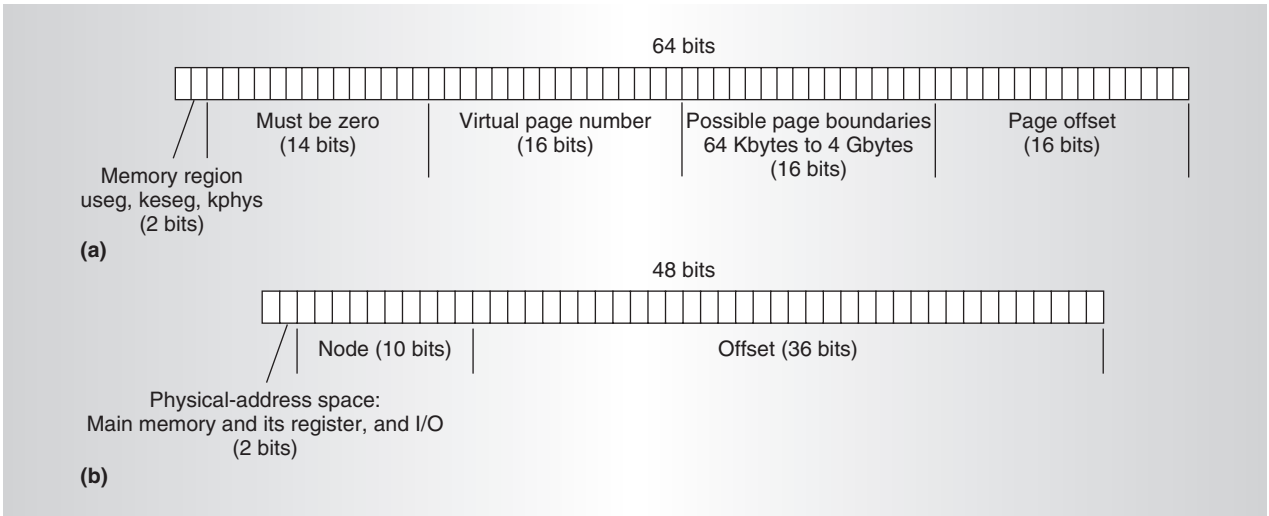


Figure 3. Cray X1 address translation.

Table 1. Platform configurations.

Characteristic	SGI Altix	Alpha SC	Earth Simulator	IBM SP4	Cray X1
Processor	Intel Itanium 2	Compaq Alpha EV67	NEC SX-6	IBM Power4	Cray X1
Interconnect	Numalink	Quadrics (Elan3)	Custom crossbar	HPS or SP Switch2	Cray X1
Processor speed (MHz)	1,500	667	500	1,300	800
Memory/node (Gbytes)	512	2	16	32	16
L1 cache size (Kbytes)	32	64	NA	32	16 (scalar)
L2 cache size (Mbytes)	0.256	8	NA	1.5	2 (per MSP)
L3 cache size (Mbytes)	6	NA	NA	128	NA
Processor peak performance (Mflops)	6,000	1,334	8,000	5,200	12,800
Peak memory bandwidth (Gbytes/s)	6.4	5.2	32 (per each processor)	51 (per mutichip modules)	26 (per each MSP)

configuration to 1,024 nodes (4,096 MSPs). The address translation scheme uses 256-entry table 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.

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.

Programming models

An X1 node (four MSPs) supports a cache-coherent shared memory, and Cray supports

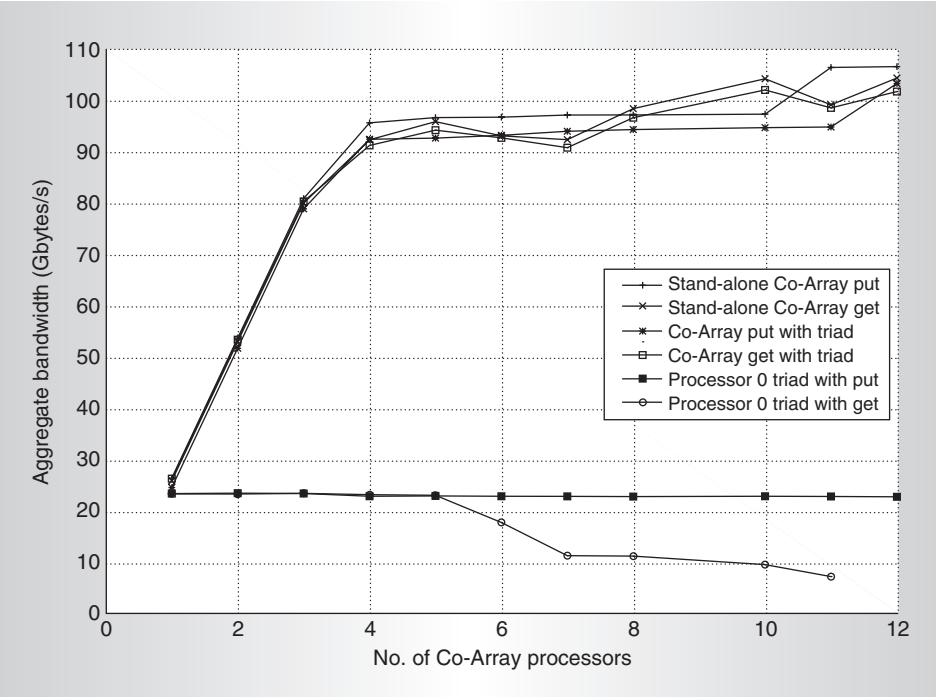


Figure 4. Stream triad with Co-Array traffic.

OpenMP System V shared memory, and Posix threads shared-memory programming (SMP) models. In addition, the compilers can treat the node processors as four streaming MSPs (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),[12] 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 microarchitecture 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 the management of external memory (that is, there is no caching for designated variables). Scientific libraries provide efficient management for 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.

Microbenchmarks

We use a collection of microbenchmarks to characterize the performance of the underlying hardware, compilers, and software libraries. The triad memory bandwidth for the Stream benchmark (http://www.cs.virginia.edu/stream) 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 of bandwidth for one processor of the modified NEC SX-6 in the Earth Simulator and the 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).

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/s. 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 the 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 and Figure 6 show the MPI intra- 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

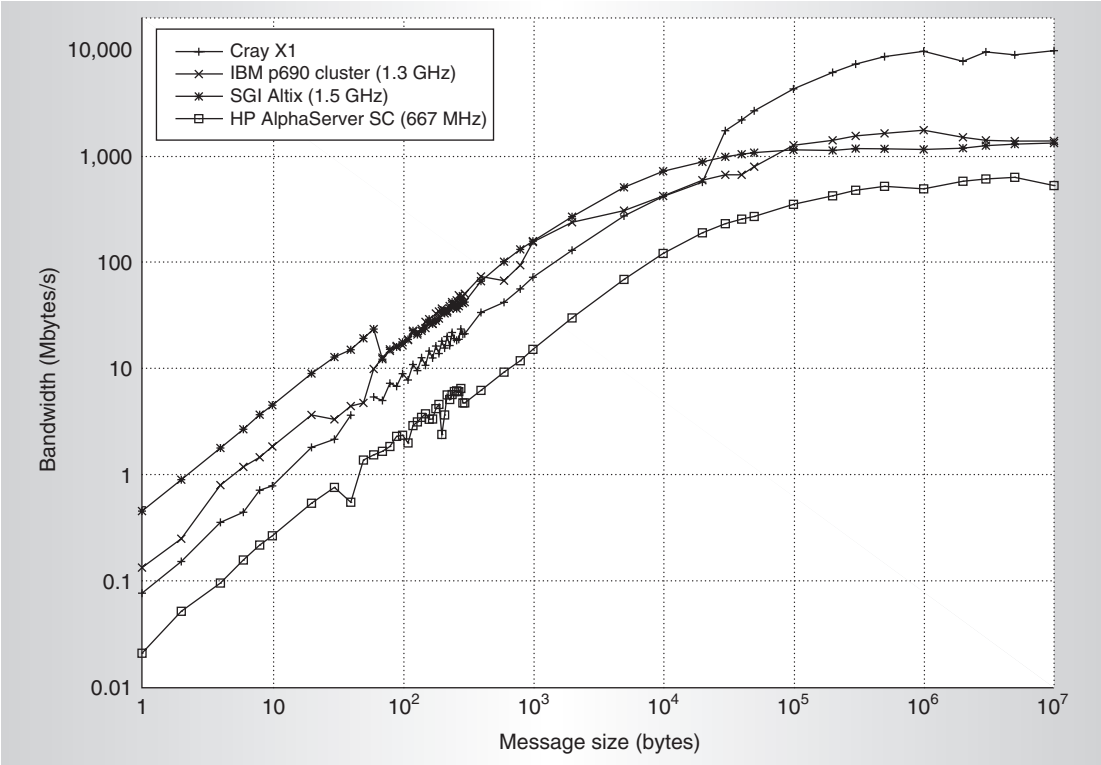


Figure 5. MPI intranode bandwidth.

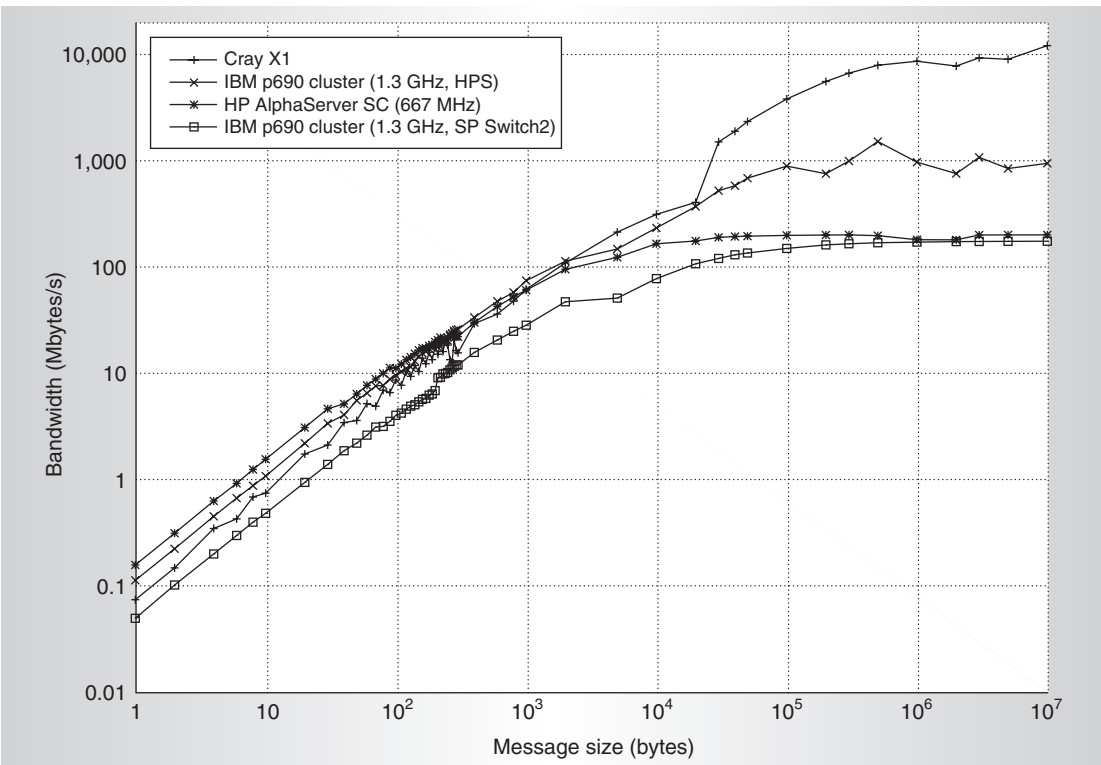


Figure 6. MPI internode bandwidth.

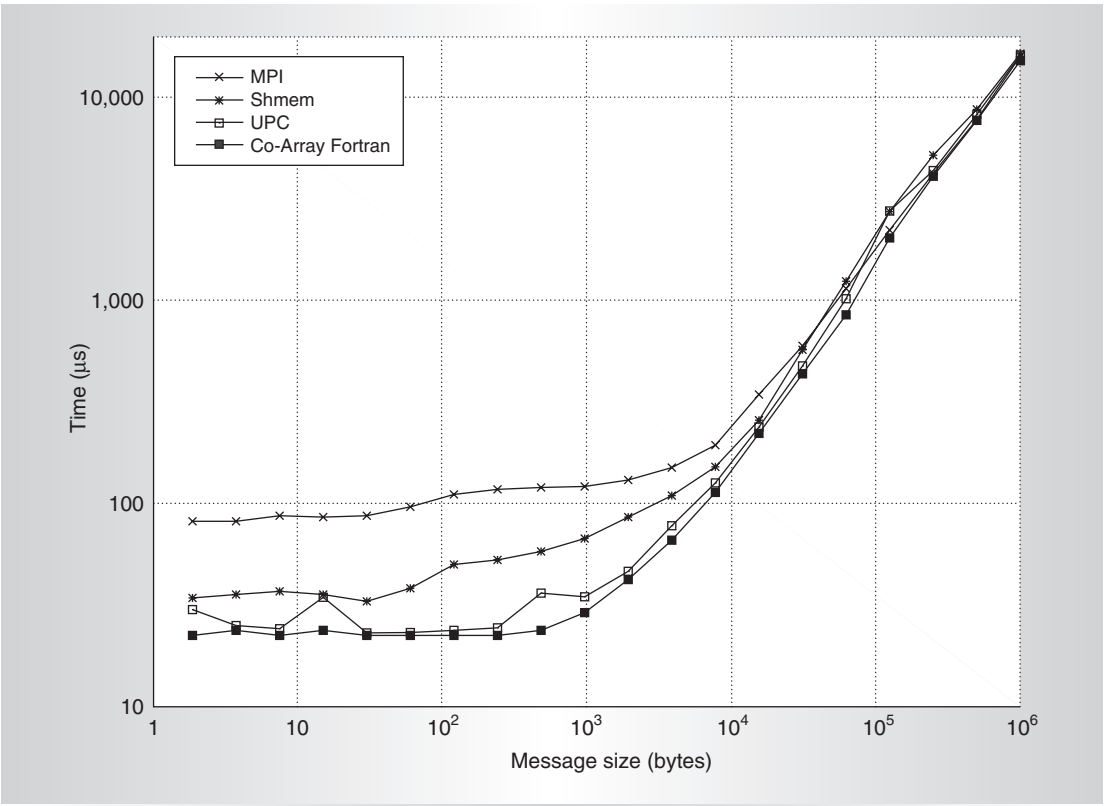


Figure 7. Halo exchange timings.

different nodes. MPI latency was 7.3 µs (one-way) for an 8-byte message between X1 nodes. Each additional hop in the torus network requires less than 0.5 µs. 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 8 Kbytes.

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[13] on 16 MSPs. The Halo benchmark simulates the nearest neighbor exchange of a 1 to 2 row or column "halo" from a 2D array. This is a common operation in 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 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 can 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 use the underlying hardware. However, barriers for the various programming models use the same underlying hardware and average about 5 µs, essentially independent of the number of participating processors at the current scale (up to 512 MSPs).

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.

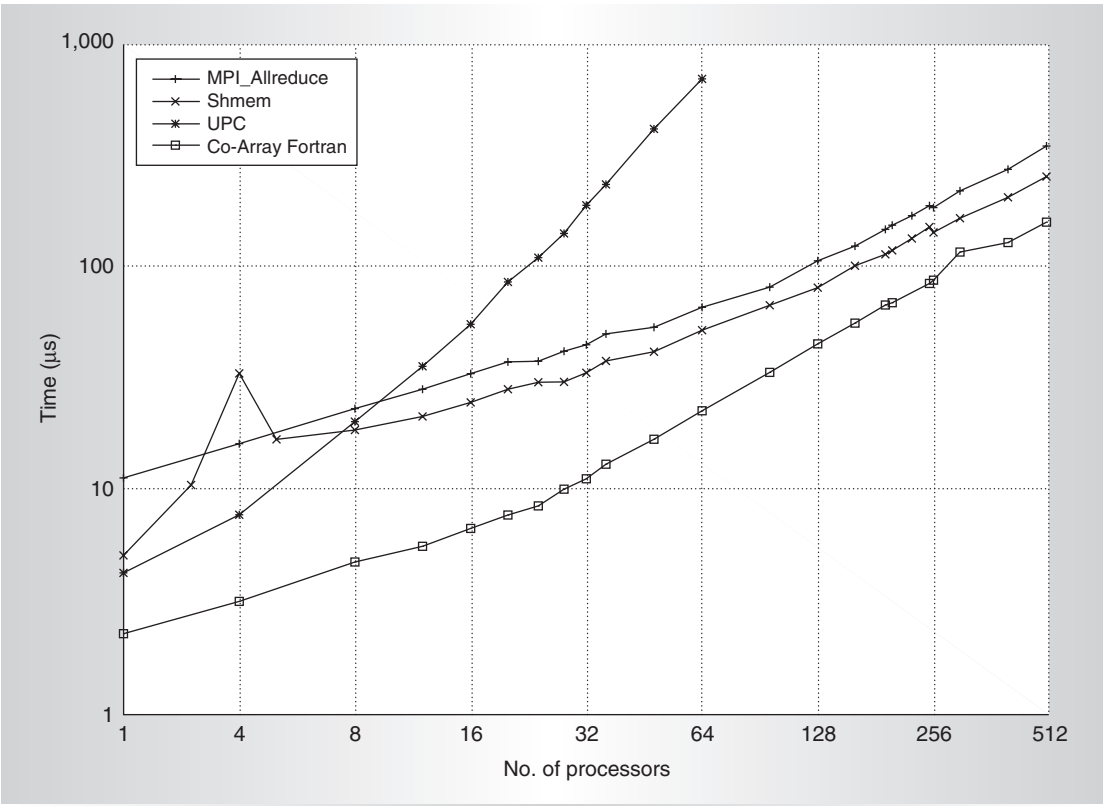


Figure 8. Allreduce latency.

Climate modeling

The Parallel Ocean Program (POP)[14] is an ocean modeling code developed at Los Alamos National Laboratory (LANL); it serves as the ocean component in the Community System Climate 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 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 is an Eulerian, gyrokinetic Maxwell solver developed by R.E. Waltz and J. Candy at General Atomics.[15] 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.

Our experiments show that the high bandwidth and 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 benchmark results also demonstrate that it can be important to select the appropriate programming models to exploit these benefits. For the most recent

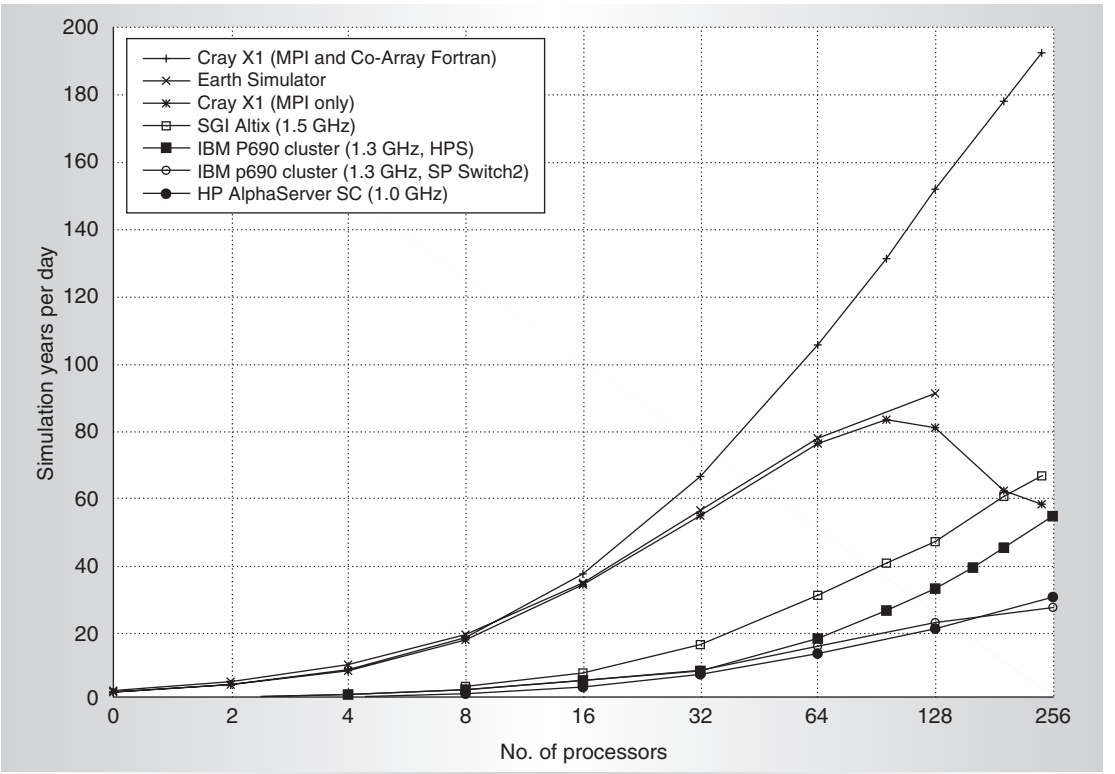


Figure 9. Performance of the LANL Parallel Ocean Program (POP) version 1.4.3.

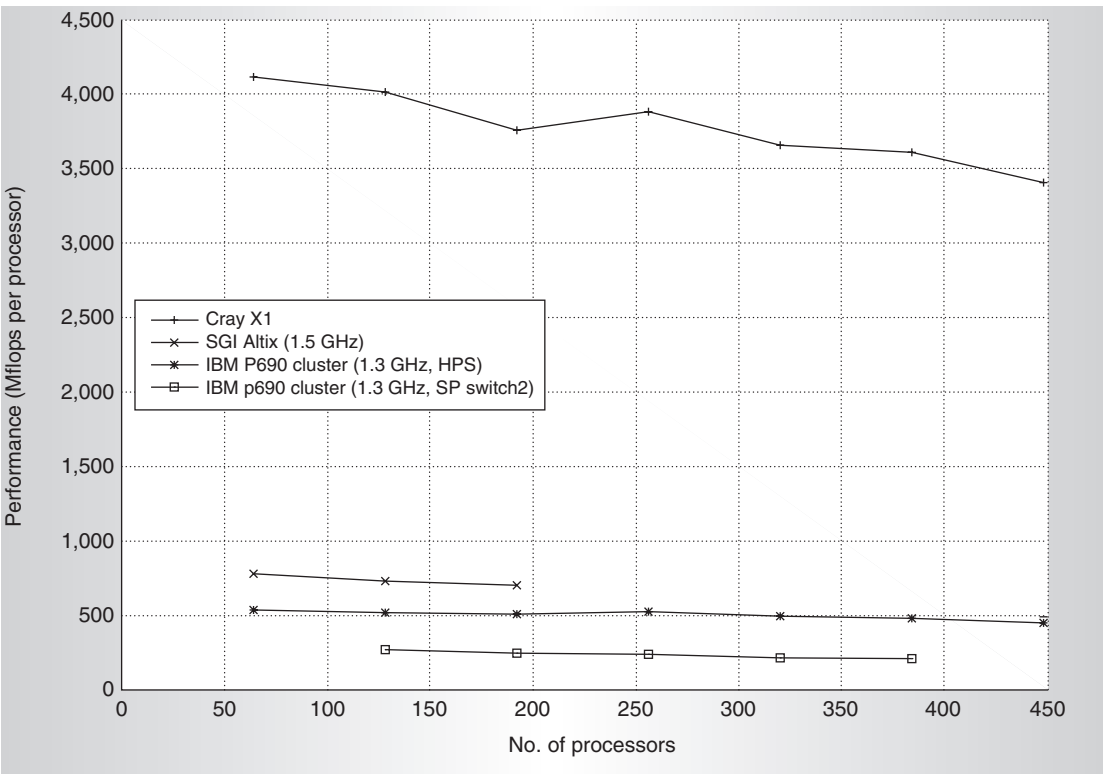


Figure 10. Performance of the Gyro Eulerian Gyrokinetic-Maxwell Solver (64-mode GTC benchmark).

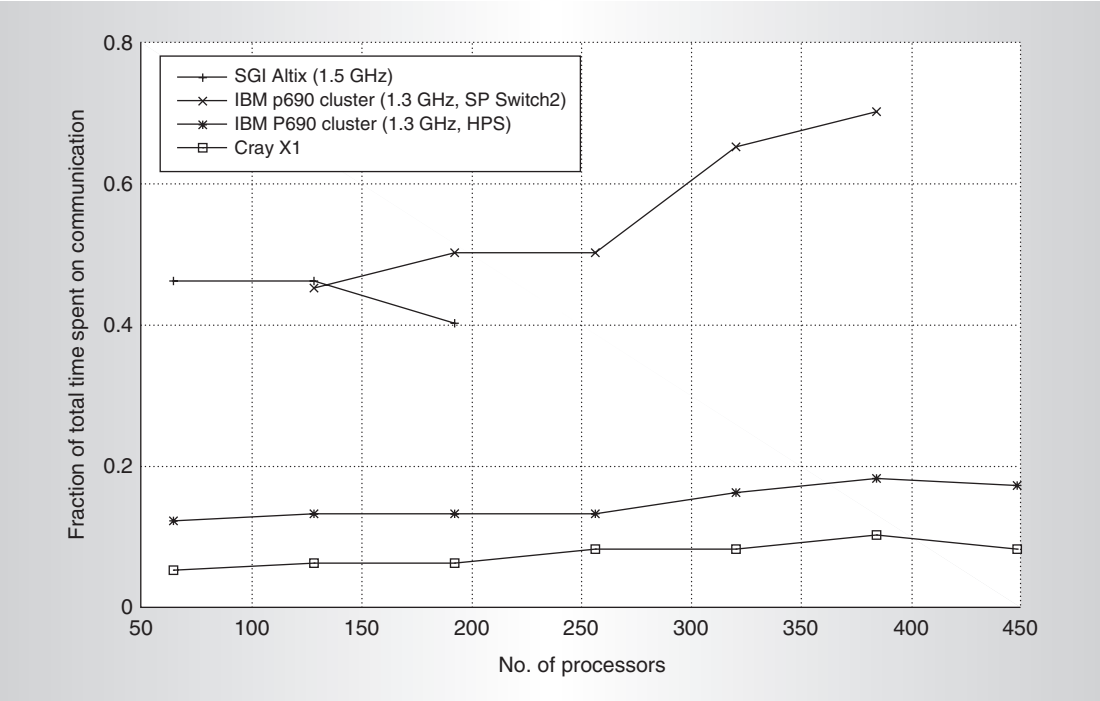


Figure 11. Communication fraction for Gyro (64-mode GTC benchmark).

results and additional performance data comparing the X1 with other systems, see http://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. MICRO

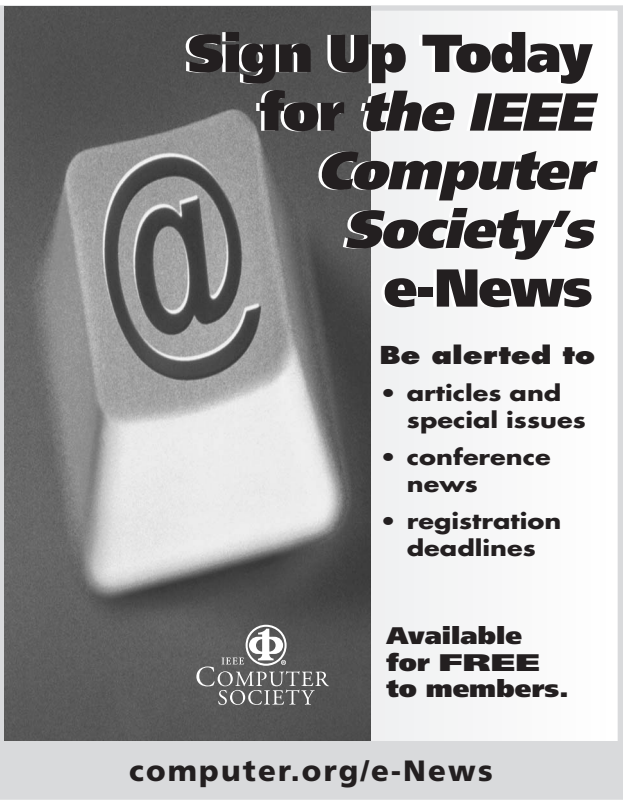
Acknowledgments

We gratefully acknowledge Cray Inc. for its ongoing cooperation and, in particular, Steve Scott, James Schwarzmeier, and Nathan Wichmann. The work described in this article was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, US Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. Accordingly, the US government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for US Government purposes.

References

1. P.A. Agarwal et al., *Cray X1 Evaluation Status Report*, tech. report ORNL/TM-2004/13, Oak Ridge National Lab., 2004; http://www.csm.ornl.gov/evaluation/PHOENIX/PDF/CRAYEvaluationTM2004-15.pdf.
2. T.H. Dunigan Jr. et al., "Early Evaluation of the Cray X1," *Proc. ACM/IEEE SC2003 Conf.* (SC 03), IEEE CS Press, 2003, p. 18.
3. R. Barriuso and A. Knies, *SHMEM Users Guide*, Cray Research, Inc., 1994.
4. W.W. Carlson et al., "Introduction to UPC and Language Specification," tech. report CCS-TR-99-157, Center for Computing Sciences, Institute for Defense Analyses, 1999.
5. R.W. Numrich and J. Reid, "Co-Array Fortran for Parallel Programming," *ACM SIGPLAN FORTRAN Forum*, vol. 17, no. 2, Aug. 1998, pp. 1-31.
6. J. Nieplocha, R.J. Harrison, and R.J. Littlefield, "Global Arrays: A Portable Shared Memory Model for Distributed Memory Computers," *Proc. Supercomputing 94* (SC 94), IEEE CS Press, 1994, pp. 340-349.
7. S.L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Proc. Architectural Support for Programming Languages and Operating Systems* (ASPLOS

96), ACM Press, 1996, pp. 26-36.
8. S. Shingu et al., "A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator," *Proc. SC2002 Conf.* (SC 02), IEEE CS Press, 2002, p. 52.
9. O. Lascu et al., *An Introduction to the New IBM pSeries High Performance Switch*, IBM Redbook SG24-6978-00, IBM, 2004.
10. W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers, 2003.
11. *Cray X1 System Overview*, tech. manual S-2346-22, Cray Inc., 2002.
12. *MPI: The Complete Reference*, 2nd edition, M. Snir et al., eds., MIT Press, 1998.
13. A.J. Wallcraft, "The NRL Layered Ocean Model Users Guide," Naval Research Lab., Stennis Space Center, 1991.
14. P.W. Jones et al., "Practical Performance Portability in the Parallel Ocean Program (POP)," to appear in *Concurrency and Computation: Experience and Practice*, Wiley, 2004.
15. J. Candy and R.E. Waltz, "An Eulerian Gyrokinetic-Maxwell Solver," *J. Computational Physics*, vol. 186, no. 2, Apr. 2003, pp. 545-581.



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.

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.

James B. White III (Trey) is an R&D staff member and a technical lead of the Scientific Application Support group in the Center for Computational Sciences at Oak Ridge National Laboratory. His research interests include enabling new science through high-end computation and improving the productivity of computational scientists. White has an MS in physics from the Ohio State University.

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.

Direct questions and comments about this article to Jeffrey S. Vetter, Oak Ridge National Laboratory, MS-6173, Oak Ridge, TN 37831-6173; vetter@computer.org.