Early Performance Evaluation of the Cray X1 at Oak Ridge National Laboratory *

P. H. Worley †

T. H. Dunigan, Jr. ‡

Abstract

Oak Ridge National Laboratory recently installed a 32 processor Cray X1. In this paper, we describe our initial performance evaluation of the system, including microbenchmark data quantifying processor, memory, and network performance, and kernel data quantifying the impact of different approaches to using the system.

1 Introduction

The mainstream of parallel supercomputing in the U.S. has for some time been dominated by clusters of commodity shared-memory processors (SMPs). The Department of Energy's (DOE) Accelerated Strategic Computing Initiative (ASCI), the National Science Foundation's (NSF) Extensible Terascale Facility (ETF), and, with a few exceptions, the Department of Defense's (DOD) High Performance Computing Modernization Program are all following this path. This technology path provides excellent price/performance when measured from the theoretical peak speed, but many applications (e.g., climate, combustion) sustain only a small fraction of this speed. Indeed, over the past decade, the sustained fraction of peak performance achieved by these applications has been falling steadily. Many factors account for this, but foremost are the increasing disparity between the speed of processors and memory and the design of processors for commodity applications rather than high-end scientific simulations. Illustrative of an alternative path is the Japanese Earth Simulator, which recently attracted attention (including several Gordon Bell prizes at the SC2002 conference) by sustaining a large fraction of its 40 TFLOP/s peak performance on several scientific applications. This performance advantage was realized by employing relatively few (5,120) powerful (8 GFlops/sec) vector processors connected to high-bandwidth memory and coupled by a high-performance network.

The X1 is the first of Cray's new scalable vector systems. [1] The X1 is characterized by high-speed custom vector processors, high memory bandwidth, and a high-bandwidth, low-latency interconnect linking the nodes. The efficiency of the processors in the Cray X1 is anticipated to be comparable to the efficiency of the NEC SX-6 processors in the Earth Simulator on many computational science applications. A significant feature of the Cray X1 is that it attempts to combine the processor performance of traditional vector systems with the scalability of modern microprocessor-based architectures.

Oak Ridge National Laboratory (ORNL) recently procured a Cray X1 for evaluation. A 32 processor system was delivered at the end of March, 2003, and delivery of a 256 processor system will be complete by the end of September, 2003. Processor performance, system performance, and production computing readiness are all being evaluated. The evaluation approach focuses on scientific appli-

*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. 6012, Oak Ridge, TN 37831-6367 (WorleyPH@ornl.gov)

‡Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367 (DuniganTHJr@ornl.gov)

[1] See http://www.cray.com/products/systems/x1/ for more details.

cations and the computational needs of the science and engineering research communities. The primary tasks of this evaluation are to:

1. Evaluate benchmark and application performance and compare with systems from other high performance computing (HPC) vendors.
2. Determine the most effective approaches for using the Cray X1.
3. Evaluate system and system administration software reliability and performance.
4. Predict scalability, both in terms of problem size and number of processors.

In this paper, we describe initial results from the first two evaluation tasks.

2 Cray X1 Description

The Cray X1 is an attempt to incorporate the best aspects of previous Cray vector systems and massively-parallel-processing (MPP) 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, the design has a high-bandwidth, low-latency, scalable interconnect, and scalable system software. And, like the Cray SV1, the X1 design leverages commodity CMOS technology and incorporates non-traditional vector concepts, like vector caches and multi-streaming processors.

The Cray X1 is hierarchical in processor, memory, and network design. The basic building block is the multi-streaming processor (MSP), which is capable of 12.8 GFlops/sec for 64-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 super-scalar 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/sec for 64-bit operations. The four SSPs share a 2 MB "Ecache".

The primary strategy for utilizing the eight vector units of a single MSP is parallelism through outer loops and pipelined operations. However, Cray does support the option of treating each SSP as a separate processor. The Ecache has sufficient single-stride bandwidth to saturate the vector units of the MSP. The Ecache is needed because the bandwidth to main memory is not enough to saturate the vector units without data reuse - memory bandwidth is roughly half the saturation bandwidth.

Four MSPs and a flat, shared memory of 16 GB form a Cray X1 node. The memory banks of a node provide 200 GB/s of bandwidth, enough to saturate the paths to the local MSPs and service requests from remote MSPs. 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. This represents a byte per flop of interconnect bandwidth per computation rate, compared to 0.25 bytes per flop on the Earth Simulator and less than 0.1 bytes per flop expected on an IBM p690 with the maximum number of Federation connections. The collected nodes of an X1 have a single system image.

A single four-processor X1 node behaves like a traditional SMP, but each processor has the additional capability of directly addressing memory on any other node (like the T3E). Remote memory accesses go directly over the X1 interconnect to the requesting processor, bypassing the local cache. This mechanism is more scalable than traditional shared memory, but it is not appropriate for shared-memory programming models, like OpenMP [12], outside of a given four-processor node. This remote memory access mechanism is a good match for distributed-memory programming models, particularly those using one-sided put/get operations.

In large configurations, the Cray X1 nodes are connected in an enhanced 3D torus. This topology has relatively low bisection bandwidth compared to crossbar-style interconnects, such as those on the NEC SX-6 and IBM SP. 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 well to thousands of processors.

3 Evaluation Overview

In this paper we describe data obtained from microbenchmarking, application kernel optimization and benchmarking, and application benchmarking.

3.1 Microbenchmarking

The objective of the microbenchmarking is to characterize the performance of the underlying architectural components of the Cray X1. Both standard benchmarks and customized benchmarks are being used. The standard benchmarks allow component

performance to be compared with other computer architectures. The custom benchmarks permit the unique architectural features of the Cray X1 (distributed vector units, and cache and global memory) to be tested with respect to the target applications.

In the architectural-component evaluation we assess the following:

1. Scalar and vector arithmetic performance, including varying vector length and stride and identifying what limits peak computational performance.

2. Memory-hierarchy performance, including cache, local memory, shared memory, and global memory. These tests utilize both System V shared memory and the SHMEM primitives [5], as well as Unified Parallel C (UPC) [3] and Co-Array Fortran [11]. Of particular interest is the performance of the shared memory and global memory, and how remote accesses interact with local accesses.

3. Task and thread performance, including performance of thread creation, locks, semaphores, and barriers. Of particular interest is how explicit thread management compares with the implicit control provided by OpenMP, and how thread scheduling and memory/cache management (affinity) perform.

4. Message-passing performance, including intra-node and inter-node MPI [9] performance for one-way (ping-pong) messages, message exchanges, and aggregate operations (broadcast, all-to-all, reductions, barriers); message-passing hotspots and the effect of message passing on the memory subsystem are of particular interest.

5. System and I/O performance, including a set of tests to measure OS overhead (context switches), virtual memory management, low-level I/O (serial and parallel), and network (TCP/IP) performance.

3.2 Kernels and Performance Optimization

The Cray X1 has a number of unique architectural features, and coding style and parallel programming paradigm are anticipated to be as important as the compiler in achieving good performance. In particular, early indications are that coding styles that perform well on the NEC vector systems do not necessarily perform well on the X1.

Unlike traditional vector machines, the vector units are independent within the X1 MSP. As such, the compiler must be able to identify parallelism for "multistreaming" in addition to vectorization, and to exploit memory access locality in order to minimize cache misses and memory accesses. As described earlier, an alternative approach is to treat the four SSPs making up an MSP as four (3.2 GFlops/sec peak) vector processors, assigning separate processes to each. This takes the responsbility of assigning work to the vector units away from the compiler and gives it to the user, and may achieve higher performance for certain applications.

Each node in the Cray X1 has 4 MSPs in a shared-memory configuration with a network that supports globally addressable memory. How best to program for the hierarchical parallelism represented by clusters of SMP nodes is typically both system and application specific. The addition of both multistreaming and vectorization makes it even less clear which of the many shared- and distributed-memory programming paradigms are most appropriate for a given application. Cray currently supports the MPI programming paradigm, as well as the SHMEM one-sided communication library and the Co-Array Fortran and UPC parallel programming languages. OpenMP is expected to be supported later in the summer of 2003. System V shared-memory, Multi-Level Parallelism (MLP) [13], and Global Arrays [10] are also possible programming models for the Cray X1.

These usability and optimization issues are difficult to examine with either microbenchmarks, which do not measure the difficulty of implementing a given approach in an application code, and full application codes, in which it is not feasible to modify to try all of the different approaches. The approach we take is to identify representative kernels and use these to examine the performance impact of a variety of coding styles. The goal is not just to choose the best programming paradigm, assuming that one is clearly better than the others, but also to evaluate what can be gained by rewriting codes to exploit a given paradigm. Because of their complexity, many of the important scientific and engineering application codes will not be rewritten unless significant performance improvement is predicted. The evaluation data will also allow us to interpret the fairness of using a given benchmark code implemented using, for example, pure MPI-1 or OpenMP, when comparing results between platforms.

3.3 Application Evaluation and Benchmarking

A number of DOE Office of Science application codes from the areas of global climate, fusion, materials, chemistry, and astrophysics have been identified as being important to the DOE evaluation of the X1. In the paper and talk, we examine the performance of one of the codes - the Parallel Ocean Program (POP) [7].

The Parallel Ocean Program (POP) is the ocean component of the Community Climate System Model (CCSM) [2], the primary model for global climate simulation in the U.S. POP is developed and maintained at Los Alamos National Laboratory (LANL). The code is based on a finite-difference formulation of the three-dimensional flow equations on a shifted polar grid.

POP has proven amenable to vectorization on the SX-6, and the expectation is that the same will hold true on the X1. The two primary processes in POP tests the Cray X1 in different ways. The "baroclinic" process is three dimensional with limited nearest-neighbor communication and should scale well. The "barotropic" process, however, involves a two-dimensional, implicit solver and limits scalability. This two-dimensional process tests the latency of the X1 interconnect; early POP results from the NEC SX-6 show that latency is a significant bottleneck. The effectiveness of targeted communication optimizations to minimize latency, such as replacing MPI-1 calls with Co-Array Fortran, is being investigated.

4 Evaluation Results

All experiments were run using version 2.1.06 of the UNICOS/mp operating system, version 4.3.0.0 of the Cray Fortran and C compilers and math libraries, and version 2.1.1.0 of the MPI and SHMEM libraries. For comparison purposes, performance data is also presented for the following systems:

- Earth Simulator: 640 8-way vector SMP nodes and a 640x640 single-stage crossbar interconnect. Each processor has 8 64-bit floating point vector units running at 500 Mhz.
- HP/Compaq AlphaServer SC at Pittsburgh Supercomputing Center (PSC): 750 ES45 4-way SMP nodes and a Quadrics QsNet interconnect. Each node has two interconnect interfaces. The processors are the 1GHz Alpha 21264 (EV68).
- HP/Compaq AlphaServer SC at ORNL: 64 ES40 4-way SMP nodes and a Quadrics QsNet interconnect. Each node has one interconnect interface. The processors are the 667MHz Alpha 21264a (EV67).
- IBM p690 cluster at ORNL: 27 32-way p690 SMP nodes and an SP Switch2. Each node has 2 to 8 Corsair (Colony on PCI) interconnect interfaces. The processors are the 1.3 GHz POWER4.
- IBM SP at the National Energy Research Supercomputer Center (NERSC): 184 Nighthawk(NH) II 16-way SMP nodes and an SP Switch2. Each node has two interconnect interfaces. The processors are the 375MHz POWER3-II.
- IBM SP at ORNL: 176 Winterhawk (WH) II 4-way SMP nodes and an SP Switch. Each node has one interconnect interface. The processors are the 375MHz POWER3-II.
- NEC SX-6 at the Artic Region Supercomputing Center: 8-way SX-6 SMP node. Each processor has 8 64-bit floating point vector units running at 500 MHz.
- SGI Origin 3000 at Los Alamos National Laboratory: 512-way SMP node. Each processor is a 500 MHz MIPS R14000.

The evaluation team has had access to the X1 for approximately one month, and an update of the system software two weeks ago necessitated rerunning many of the experiments. In consequence, none of our evaluations are complete. However, data has been collected that illuminates a number of important performance issues, including:

- Range of serial performance. As in previous vector systems, the Cray X1 processor has a large performance range. Code that vectorizes and streams (uses multiple SSPs) well can achieve near peak performance. Code that runs on the scalar unit demonstrates poor performance.
- Memory performance. Can the memory hierarchy keep the vector units busy? Can processes on separate MSPs contend for memory bandwidth?

- SSP vs. MSP. What are the advantages or disadvantages to running in SSP mode (assigning separate processes to individual SSPs) compared to running in MSP mode?

- Page size. At run time, the user can specify whether to run using 64 KB, 1 MB, 4 MB, 16 MB, or 64 MB pages. The default within a 4-way SMP node is 64 KB pages, while the default when running across more than one node is 16 MB pages. Does this make a performance difference?

- Communication protocol. When using MPI, which of the many MPI commands should be used to implement point-to-point communication.

- Alternatives to MPI. What are the performance advantages of SHMEM and Co-Array Fortran?

While only preliminary results are presented here, additional data will be posted on the Evaluation of Early Systems project web pages at `http://www.csm.ornl.gov/evaluation`.

4.1 Standard Benchmarks

We are running a number of standard low-level and kernel benchmarks on the Cray X1 to identify performance potential and bottlenecks of the computational units and the memory subsystem. As standard benchmarks, we have not modified them in any way, except for linking in vendor libraries where appropriate.

4.1.1 Computation Benchmarks

The first three serial benchmarks indicate the range of performance that can be seen on the X1.

DGEMM. Figure 4.1 compares the performance of the vendor scientific libraries for a matrix multiply with DGEMM [4] on an X1 MSP with the performance on other IBM and HP systems.

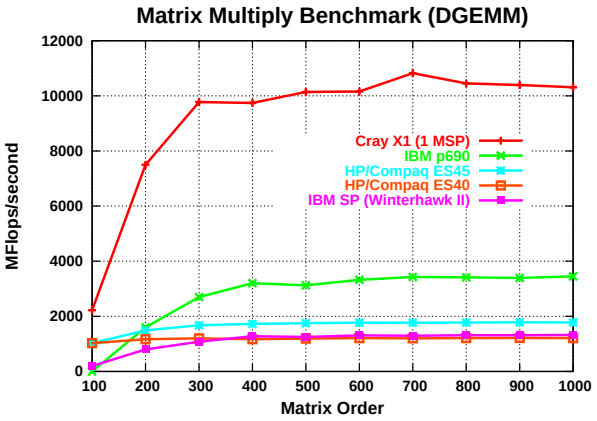

FIGURE 4.1: Single processor DGEMM performance

From these data it is clear that it is possible to achieve near peak ($> 80\%$) performance on computationally intensive kernels.

MOD2D. Figure 4.2 illustrates the single processor performance of a dense eigenvalue benchmark (Euroben MOD2D [14]) using the vendor-supplied scientific libraries. On the X1, we used a single MSP. For this benchmark, we increased the problem size for the X1 to illustrate that its performance is still improving with larger matrices.

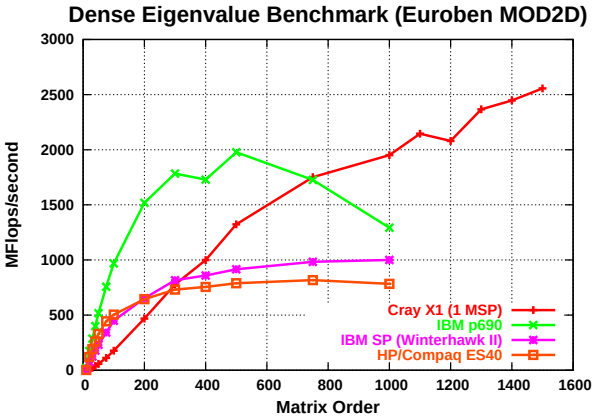

FIGURE 4.2: Single processor MOD2D performance

Performance is good for large problem sizes relative to the IBM and HP systems, but the percentage of peak is much lower than for DGEMM for these problem sizes. The eigenvalue routines in the Cray scientific library (`libsci`) may not yet have received the same attention as the matrix multiply routines, and MOD2D performance may improve over time.

MOD2E. Figure 4.3 describes the single processor performance for the Euroben MOD2E benchmark [14], eigenvalue analysis of a sparse system implemented in Fortran. Note that the performance metric here is iterations per second (of the iterative algorithm), and that larger is better.

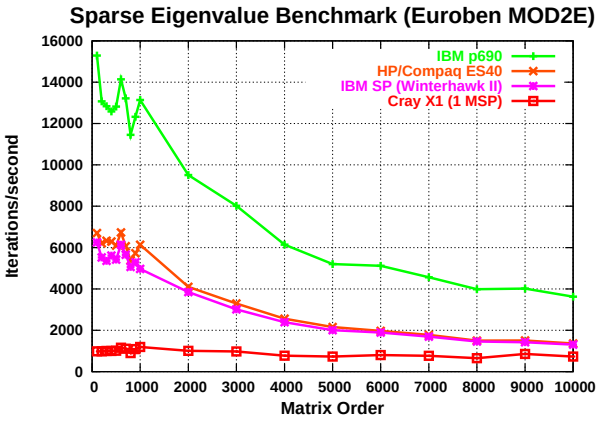

FIGURE 4.3: Single processor MOD2E performance.

The performance of MOD2E on the X1 is poor compared to the nonvector systems. While aggressive compiler optimizations were enabled, this benchmark was run unmodified, that is, none of the loops were modified nor were compiler directives inserted to enhance vectorization or streaming. Whether the performance problem is due to the nature of the benchmark problem or the nature of the code is not yet clear. Note that performance on the X1 improves for the larger problem sizes. The iterations per second metric decreases much slower than the matrix size, and computational complexity per iteration, increases, so some vectorization or streaming is being exploited. The lesson is clear however - it is important to exploit the vector units if good performance is to be achieved.

4.1.2 Memory Benchmarks

STREAMS. McCalpin's STREAMS benchmark [8] measures sustainable memory bandwidth. Figure 4.4 compares the triad bandwidth of an X1 MSP with other architectures.

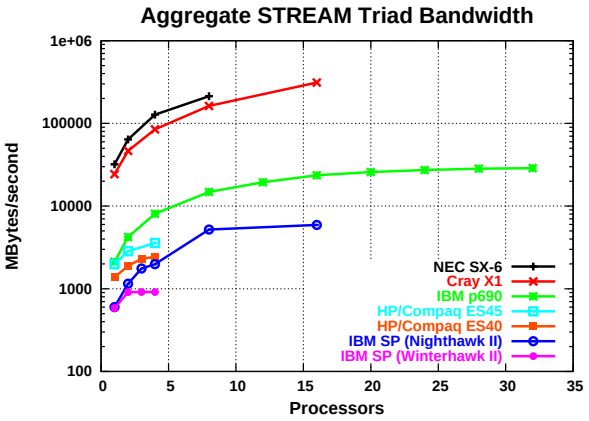

FIGURE 4.4: STREAMS triad bandwidth

From these data, the X1 has nearly the same memory bandwidth as the NEC SX-6, and more than the nonvector systems. Note that results are limited to processors within a single SMP node for all systems except the X1. When using more than 4 MSP processors, more than one X1 node is required.

4.1.3 Communication Benchmarks

Latency and bandwidth are often slippery concepts, in that they can be difficult to associate with performance observed in application codes. Using standard point-to-point comunication tests produces the following estimates for the Cray X1:

	latency (microseconds)	bandwidth MB/sec
MPI		
intranode	8.2	13914
internode	8.6	11928

In contrast, SHMEM and Co-Array latency are 3.8 and 3.0 microseconds, respectively. Maximum bandwidth is similar to that of MPI. The next benchmark attempts to measure communication performance in the context of a common application.

HALO. Wallcraft's HALO benchmark [15] 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.

In Figure 4.5, we compare the relative performance of a HALO exchange between 16 processors (MSPs) for SHMEM, MPI, and co-arrays.

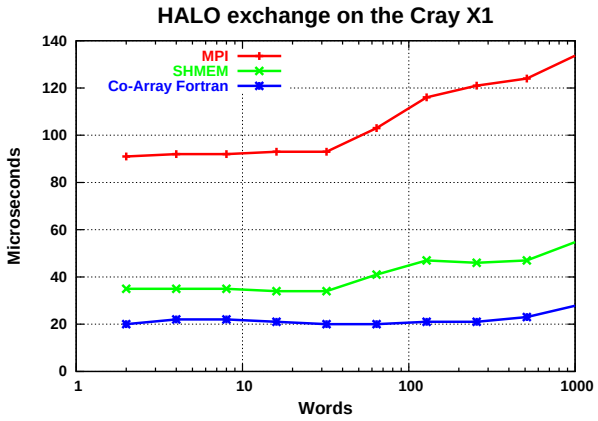

FIGURE 4.5: HALO: Comparison of Co-Array Fortran, MPI, and SHMEM performance.

From these results, there is a clear advantage from using SHMEM or Co-Array Fortran in such latency-dominated kernels.

The MPI data described in Figure 4.5 are the best timings from a number of different MPI implementations. Figure 4.6 describes the performance of all of the candidate MPI implementations.

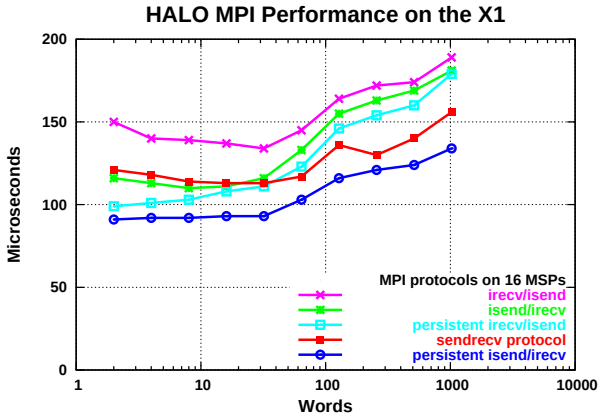

FIGURE 4.6: HALO: Performance comparison of different MPI protocols for exchange.

The optimal implementation for all halo sizes uses a persistent MPI_ISEND/MPI_IRECV exchange protocol.

Figure 4.7 compares the HALO performance of the best MPI implementations on the IBM p690 and the Cray X1.

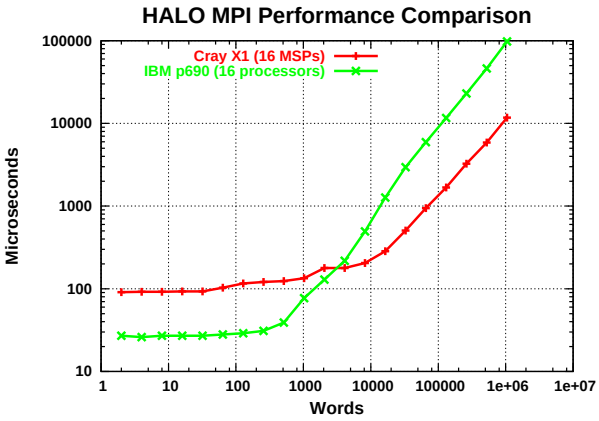

FIGURE 4.7: HALO: MPI performance intercomparison.

While the Cray X1 MPI performance is much better than that of the IBM p690 for large halos, the opposite is true for small halos. Note, however, that the IBM p690 performance for small halos is close to that for SHMEM on the X1. It is interesting that the X1 times are much higher than the 8-byte message latency. The performance loss may instead be due to the buffer copying.

4.1.4 Parallel Benchmarks

MG from the NAS Parallel Benchmark [1] (and from the ParkBench benchmark suite [6]) is a kernel that uses multigrid to solve a linear system. Figure 4.8 shows the aggregate MFlops/sec for a fixed grid of size $256 \times 256 \times 256$. Results are given for both an MPI implementation and a Co-Array Fortran implementation due to Alan Wallcraft. For the two IBM systems, results are given for both MPI and OpenMP implementations.

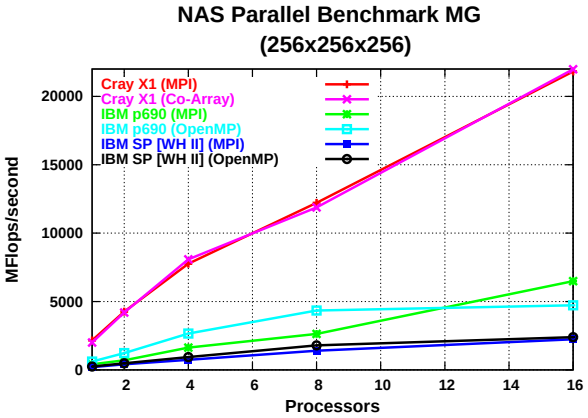

FIGURE 4.8: MG: Comparison of Co-Array Fortran, MPI, and OpenMP performance.

Due to the nature of the memory access patterns in the hierarchical multigrid algorithm, MG is typically very sensitive to latency, both interprocess and memory. From these data, the X1 performance scales reasonably well, with the only significant problem at the transition from 4 to 8 processors, i.e. the transition from using only one SMP node to using multiple nodes. Co-Array Fortran is not a significant performance enhancer for this code, problem size, and number of processors. While percentage of peak is only 10-16% on the X1, this code was not modified to improve vectorization or streaming on the X1.

4.2 Custom Benchmarks

The following custom benchmarks were developed at ORNL, and have been used in a number of performance studies over the past ten years. As these were developed on-site, we are able to experiment with different implementations and to modify them to examine new performance issues. In the first kernel, PSTSWM, we describe our experiences in porting and optimization, look for memory contention, and compare SSP and MSP performance. The second kernel, COMMTEST, looks at MPI and SHMEM message-passing performance in more detail.

PSTSWM. The Parallel Spectral Transform Shallow Water Model (PSTSWM) [16, 17] represents an important computational kernel in spectral global atmospheric models. As 99% of the floating-point operations are multiply or add, it runs well on systems optimized for these operations. PSTSWM also exhibits little reuse of operands as it sweeps through the field arrays, making significant demands on the memory subsystem. By these characteristics, the code should run well on vector systems. However, PSTSWM is also a parallel code that supports a large number of different problem sizes and parallel algorithms, and all array dimensions and loop bounds are determined at runtime. This makes it difficult for the compiler to identify which loops to vectorize or stream.

When porting and optimizing PSTSWM on the X1, five routines were identified as needing modification:

- Forward complex FFT
- Inverse complex FFT
- Forward Legendre transform
- Inverse Legendre transform
- Nonlinear interactions in gridpoint space

All code modifications were local to the specified subroutines. In particular, no global data structures were modified. After modification, the new versions of the Legendre transforms and the nonlinear interactions subroutines still perform reasonably well on nonvector systems. The same is not true for the FFT routines, but on most nonvector systems we use optimized math library routines for these functions. We also examined fixing the vertical problem dimension at compile-time. In each the above mentioned routines, two of the three problem dimensions (longitude, latitude, vertical) are available for vectorization and streaming, one of which is always the vertical. Specifying the vertical dimension at compile-time gives the compiler additional information for optimization. While this restricts the choice of domain decomposition and parallel algorithms in a parallel run, it is not an unreasonable restriction in many situations.

In a previous evaluation, we also ported PSTSWM to the NEC SX-6. Optimizing on the SX-6 led us to modify the same routines. While similar "vector-friendly" versions of the FFT and nonlinear interactions routines were used on the SX-6 and the Cray X1, the code modifications for the Legendre transform routines were very different for the X1 and the SX-6. On the SX-6, long vectors are the goal of any restructuring, even if indirection arrays are then needed due to loop collapsing. Similar techniques are not productive on the X1. Vectors need not be as long on the X1 as long as there is a loop to stream over. Also, the scatter-gather hardware on the X-1 is not efficient for stride-zero or stride-one scatter-gathers, limiting the utility of collapsing loops. The X1 modifications were much closer to the original version of the code.

In the following experiments we examined a number of different alternatives. The code could be run in MSP mode (using both vectorization and streaming) or in SSP mode (running on a single SSP and only exploiting vectorization). As with most compilers, there are a number of different optimization options, as well as compiler directives to provide additional control. We added a small number of directives to the above mentioned routines to indicate to the compiler which loops could be vectorized or streamed. We also looked at two different optimization levels, the default (`def. opt.`) and an aggressive setting (`agg. opt.`):

```
-Oaggress,scalar3,vector3,stream3
```

Finally, at runtime you can specify whether to use 64KB pages or 16MB pages. The default is 64KB pages when running within a single SMP node, and 16MB pages when using multiple nodes. However, unless noted otherwise, all of the following experiments were run using 16MB pages.

Figures 4.9-4.13 display performance for a number of different problem sizes: T5L18, T10L18, T21L18, T42L18, T85L18, and T170L18. The "T" number defines the horizontal (longitude, latitude) resolution, while the "L" number refers to the number of vertical levels. So the problem set varies in horizontal resolution, but all with 18 vertical levels. Each problem size requires approximately 4 times as much memory as the next smaller problem size. For example, T5 requires approximately 60KB of data space for each vertical level, while T85 requires approximately 18MB for each vertical level. The performance is described in terms of MFlops/second. The floating point operation count was calculated using an operation count model (which was validated on an IBM p690), and the computation rate should be viewed as a consistently weighted inverse time metric.

Figure 4.9 compares the serial performance when run on a single MSP.

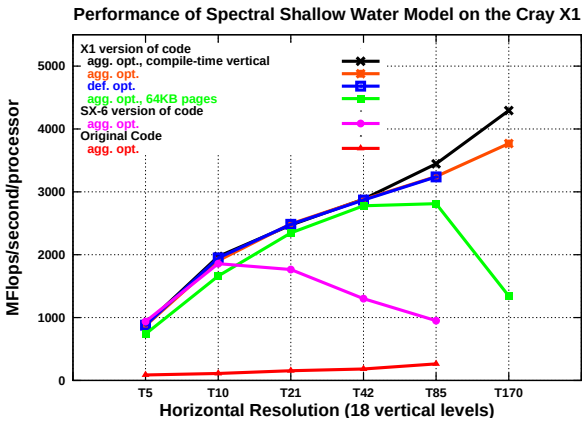

FIGURE 4.9: PSTSWM: comparing compiler options and code modification on the Cray X1

From these results, code modification was crucial for performance, and the SX-6 modifications were not appropriate on the X1. The second most important performance optimization was the use of 16MB pages. The default compiler optimizations were as good as the more aggressive optimizations for this code. Stipulating the vertical dimension at compile improved performance for the largest problem sizes. The advantage of compile-time over runtime specification is even greater when the number of vertical levels is smaller.

Figure 4.10 compares the performance when running on an MSP with running on an SSP, and when running simultaneously on all processors in an SMP node. For the MSP experiments, this denotes running 4 instances of the same problem simultaneously, while for the SSP experiments, 16 serial instances are running. We also include performance data for running 4 serial SSP instances simultaneously. In this situation, all 4 instances are scheduled on the same MSP, and this is way of examining contention between SSPs within the same MSP.

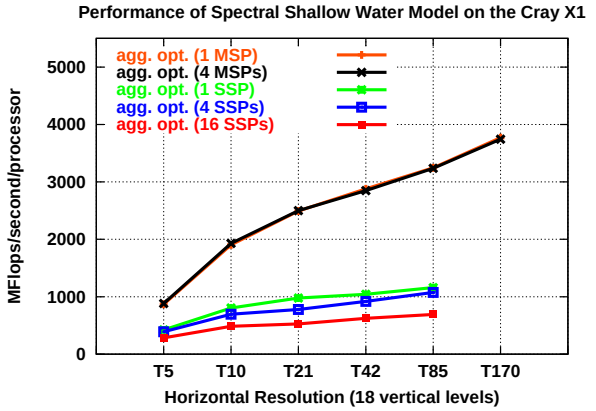

FIGURE 4.10: PSTSWM: comparing SSP and MSP performance on the Cray X1

From these results, there is no contention when running multiple instances of PSTSWM in MSP mode on an X1 node. (The curves for 1 MSP and 4 MSPs overlap, and are difficult to distinguish in this figure.) This is an excellent result, differing from all other platforms on which we have run the same experiment. Comparing performance between runs on single SSPs and MSPs is complicated. MSP performance can suffer from lack of independent work to exploit with streaming and from contention for memory bandwidth or cache conflicts between the 4 SSPs. In contrast, in the single SSP experiments the other SSPs are idle, and the active SSP has sole possession of all shared resources. In these experiments, MSP performance is twice as fast as SSP performance for the smallest problem size, growing to 2.8 times faster for the largest problem size. When looking at the simultaneous runs, the 16 simultaneous SSP instances, which do share resources, show contention. The performance when running 4 MSP instances is 3.1 to 4.7 times faster than when running 16 SSP instances. Thus, simply from a throughput metric, running simultaneous MSP jobs is more efficient than running simultaneous SSP jobs (for this

code) for all but the smallest problem instances. The 4-instance SSP experiment has performance closer to that for the single SSP, but there is still some performance degradation. However, over half of the performance degradation in the 16-instance SSP experiment appears to be due to simple memory contention within the SMP node, not contention within the MSP.

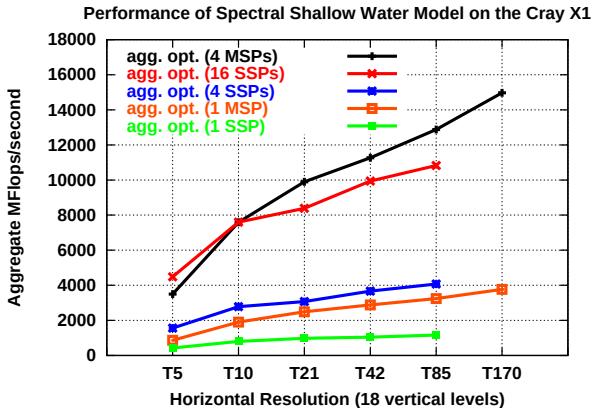

FIGURE 4.11: PSTSWM: comparing SSP and MSP aggregate performance on the Cray X1

Figure 4.11 is an alternative presentation of the same data, graphing the aggregate computational rate. Using this view, it is clear that running 4 serial SSP instances uses the MSP more effectively than running a single MSP instance (for this code, problem sizes, and experiment). Despite the memory intensive nature of the code, the lack of any need to exploit streaming results in higher efficiency, and higher throughput. The same is not true when comparing running on 4 MSPs with running on 16 SSPs, i.e. using the entire SMP node. The locality in the MSP implementation decreases demands on memory, avoiding contention that limits the SSP performance. The comparison of SSP and MSP modes will always be application and problem size dependent, and these experiments do not address important issues that arise when comparing parallel MSP and SSP codes. But these data do indicate some of the memory contention issues.

Figure 4.12 compares PSTSWM single processor performance across a number of platforms, using the best identified compiler optimizations, math libraries (when available), and versions of the code, excepting that all loop bounds are specified at runtime.

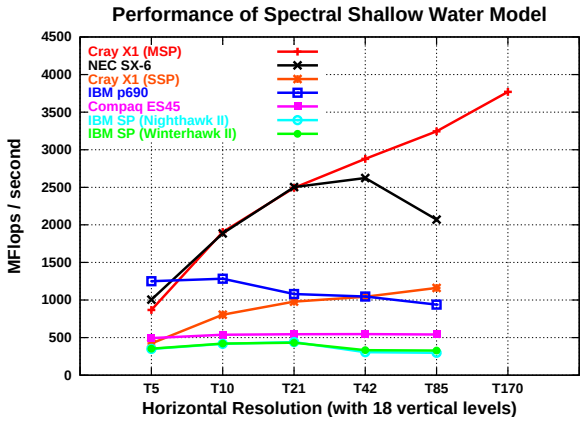

FIGURE 4.12: PSTSWM: single processor cross-platform comparison

The p690 processor (5.2 GFlops/sec peak performance) performs reasonably well on this code for small problem sizes, with performance tailing off as problem size (and the memory traffic) increases. As the problem size increases, the vector performance increases, and even the performance on a single SSP (peak performance 3.2 GFlops/sec) exceeds that of the nonvector systems for problem sizes larger than T42L18. The SX-6 performance peaks at 2.6 GFlops/sec and begins decreasing for problem sizes larger than T42. This occurs because the Legendre transforms dominate the performance for the large problem sizes, and they are only achieving approximately 1.5 GFlops/sec on the SX-6. The FFT routines, which determine performance for the small problem sizes, achieve over 4 Gflops/sec on the SX-6. Just the opposite occurs on the X1, where the Fortran implementations of the FFT routines are only achieving 2 GFlops/sec while the Legendre transform performance ranges as high as 4.5 GFlops/sec.

Figure 4.13 compares PSTSWM SMP node performance across a number of platforms. As before, the same problem is solved on all processors in the SMP node simultaneously. Performance degradation compared with the single processor performance indicates contention for shared resources and is one metric of throughput capability.

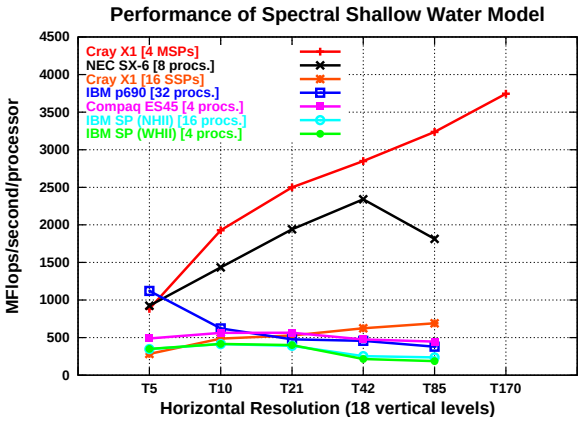

FIGURE 4.13: PSTSWM: single SMP node cross-platform comparison

As noted earlier, the lack of contention in the MSP runs is unique. All of the other platforms (and the SSP version of PSTSWM on the X1) demonstrate some performance degradation when running the larger problem sizes.

COMMTEST. COMMTEST is a suite of codes that measure the performance of MPI interprocessor communication. COMMTEST differs somewhat from other MPI benchmark suites in its focus on determining the performance impact of communication protocol and packet size in the context of "common usage". However, the performance we report here should be similar to that measured using other interprocessor communication benchmarks. SHMEM performance is also measured in the context of implementing a simple version of the SWAP and SENDRECV operators, where message selection is by source only.

Using COMMTEST, we measured the peak bidirectional SWAP (ping-ping) bandwidth for five experiments:

1. MSP 0 swaps data with MSP 1 (within the same node)
2. MSP 0 swaps data with MSP 4 (between two neighboring nodes)
3. MSP 0 swaps data with MSP 8 (between two more distant nodes)
4. MSP i swaps data with MSP $i+1$ for $i = 0, 2$, i.e. 2 pairs of MSPs (0-1, 2-3) in the same SMP node swap data simultaneously.
5. MSP i swaps data with MSP $i + P/2$ for $i = 0, \ldots, P/2 - 1$ for $P = 16$, i.e. 8 pairs of MSPs (0-8, 1-9, ..., 7-16) across 4 SMP nodes swap data simultaneously.

We measured SWAP bandwidth because the swap operator is more commonly used in performance critical interprocessor communication in our application codes than is the unidirectional send (receive).

The COMMTEST MPI results are displayed in Figures 4.14-4.15, using log-log and log-linear plots, respectively. For these experiments, we ran using 16MB pages. We measured performance using approximately 40 different implementations of SWAP using MPI point-to-point commands, including block, nonblocking, and synchronous commands, but not persistent commands. Only the best performance is reported in the graphs. For most of the experiments, the implementation using MPI_SENDRECV was among the best performers.

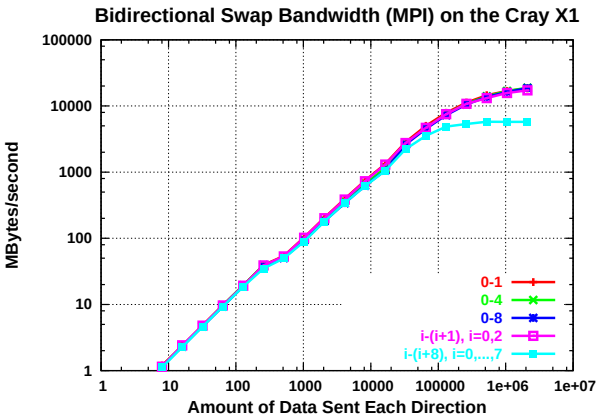

FIGURE 4.14: Log-log plot of MPI bandwidth

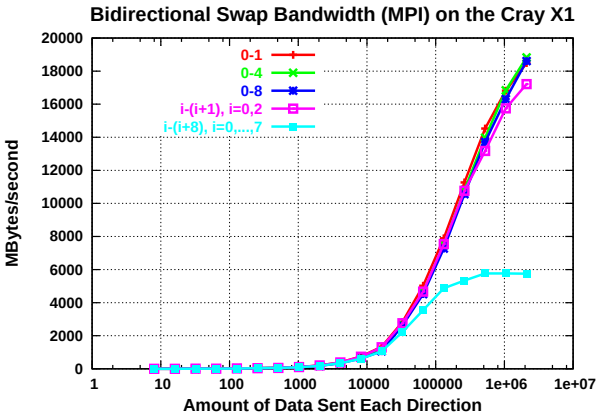

FIGURE 4.15: Log-linear plot of MPI bandwidth

Performance for small messages (length less than 4 KBytes) is similar across all experiments. For larger messages, performance differs significantly only for

the 16 processor simultaneous exchange experiment, where the internode bandwidth must be shared by all pairs. This saturation test reaches a maximum bidirectional bandwidth of 6 GB/sec for exchanging messages of size greater than 256 KB. The other experiments have not achieved maximum bandwidth by 2MB messages. These same experiments have also been run with explicitly invalidating the cache before each timing, and with multiple iterations of each swap. Neither variant affects the achieved bandwidth significantly.

Figures 4.16-4.18 graph MPI and SHMEM SWAP bidirectional bandwidth and MPI and SHMEM ECHO (ping-pong) unidirectional bandwidth for experiment (1), using log-log and log-linear plots.

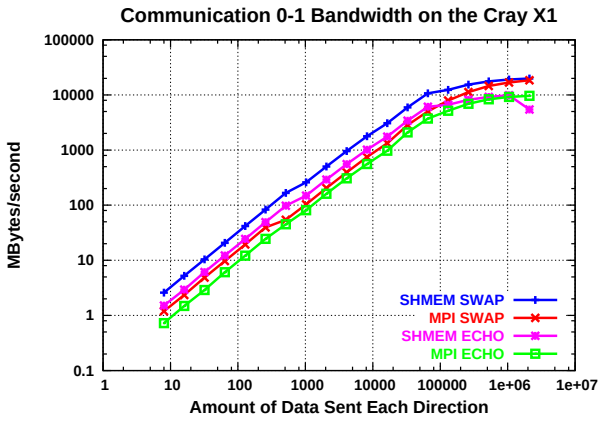

FIGURE 4.16: Log-log intranode bandwidth for 0-1 experiment.

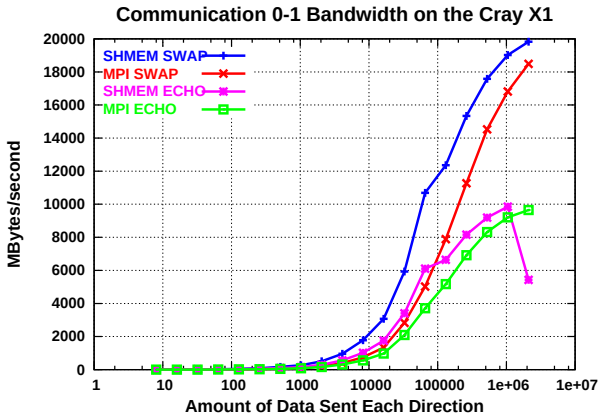

FIGURE 4.17: Log-linear intranode bandwidth for 0-1 experiment.

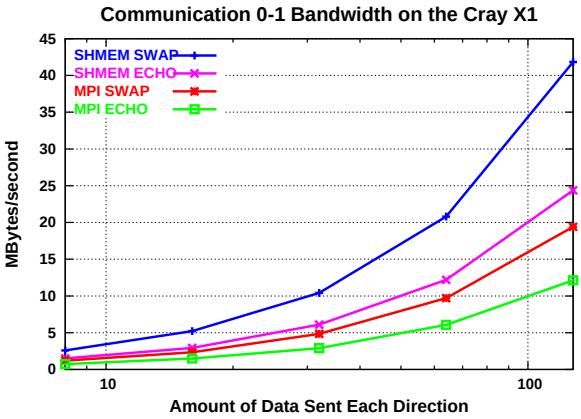

FIGURE 4.18: Log-linear intranode bandwidth for 0-1 experiment.

From these data, SHMEM SWAP achieves the highest bandwidth, and SWAP bidirectional bandwidth is close to twice the ECHO unidirectional bandwidth for all but the very largest message sizes. So, nearly full bidirectional bandwidth is achieveable. SHMEM SWAP is only 5% better than MPI SWAP for the largest message sizes, but is over twice as fast for small and medium message sizes. Note that the SHMEM echo test suffers a performance problem for the largest message size. This is consistent across the other experiments and is repeatable, with this version of the system software. It did not occur in experiments run with the previous version of system software.

Figures 4.19-4.21 graph MPI and SHMEM SWAP bidirectional bandwidth and MPI and SHMEM ECHO (ping-pong) unidirectional bandwidth for experiment (5), again using both log-log and log-linear plots.

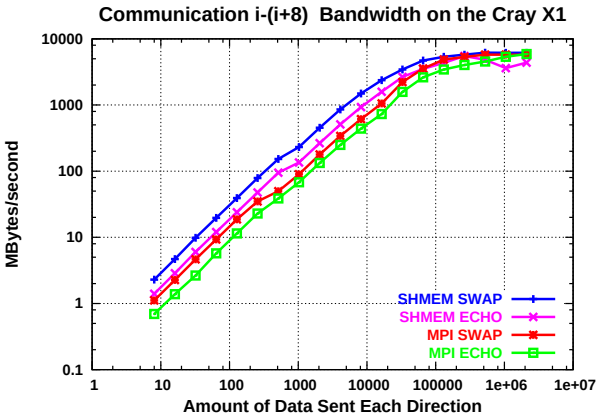

FIGURE 4.19: Log-log internode bandwidth for 8 simultaneous pairs experiment.

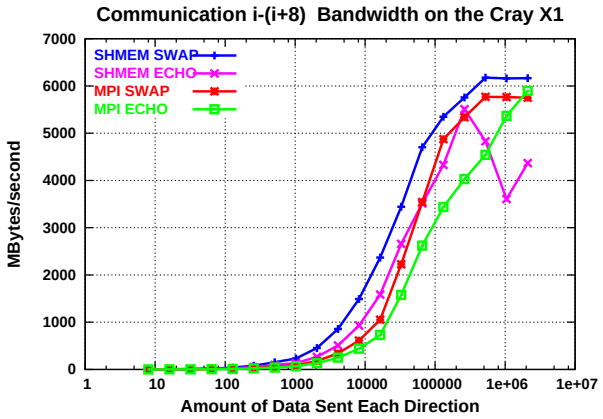

FIGURE 4.20: Log-linear internode bandwidth for 8 simultaneous pairs experiment.

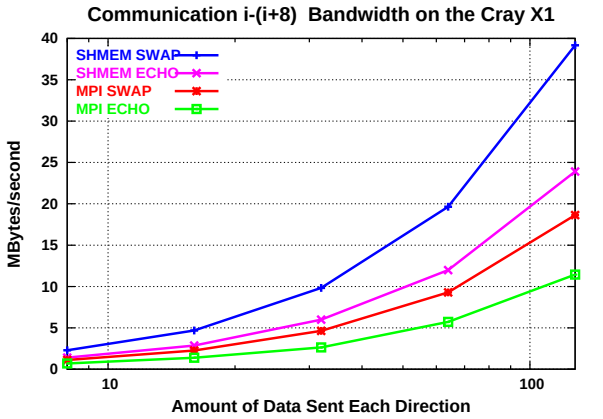

FIGURE 4.21: Log-linear internode bandwidth for 8 simultaneous pairs experiment.

These data are qualitatively very similar to those from the previous experiment. SHMEM SWAP is again the best performer, and SHMEM ECHO again has a performance problem for large messages. For small messages, the SWAP bandwidth is nearly twice that of the ECHO bandwidth. However, once SWAP bandwidth saturates the network, ECHO bandwidth continues to grow until the two are nearly equal. For the most part, MPI and SHMEM message-passing performance for the optimal protocols are smooth functions of message size.

4.3 Application Performance

POP. POP has previously been ported to the Earth Simulator, where restructuring improved the vectorization of the baroclinic process. Cray also has a port of POP, one that uses co-array Fortran to minimize the interprocessor communication overhead of the barotropic process. These two codes have similar performance, but represent orthogonal optimizations. A merger is currently being investigated, and preliminary results for the merged code are described here.

Figure 4.22 compares performance when using 64KB and 64MB pages, as well as performance data for the unmodified Earth Simulator versions and one timing using the original version of POP. The metric here is the number of years of simulation that can be computed in one day of compute time, i.e., a weighting of inverse time. The problem is a benchmark specified by the authors of the code that uses a grid with one degree horizontal resolution, referred to as the "by one", or `x1`, benchmark. The code was compiled with the default optimization level, which, as with PSTSWM, was as effective as more aggressive compiler optimizations.

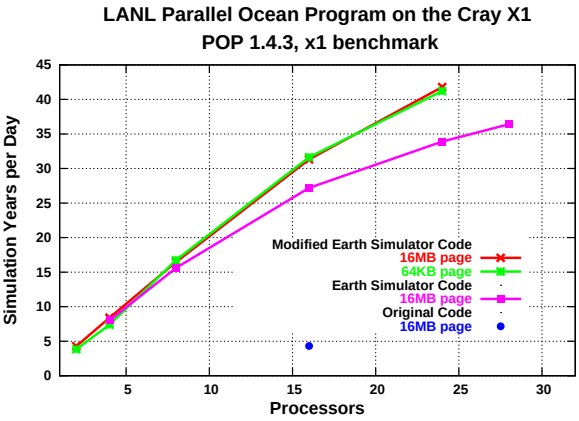

FIGURE 4.22: POP: X1 performance results

These data indicate that page size does not impact performance for POP. (The curves for using 16MB and 64KB pages overlap, and are difficult to distinguish in this figure.) Given the earlier PSTSWM results, it is safest to use 16MB pages sizes. The original version of the code again shows poor performance. The Earth Simulator version of the code performs reasonably well. The primary difference between the merged code and the Earth Simulator version is the use of Co-Array Fortran to implement the conjugate gradient algorithm, an algorithm that is very sensitive to message-passing latency.

Figures 4.23-4.25 compare the performance of POP on the X1 with performance on other systems. The first figure again plots the years per day metric. The other figures plot the seconds per simulation day spent in the baroclinic and barotropic processes, respectively.

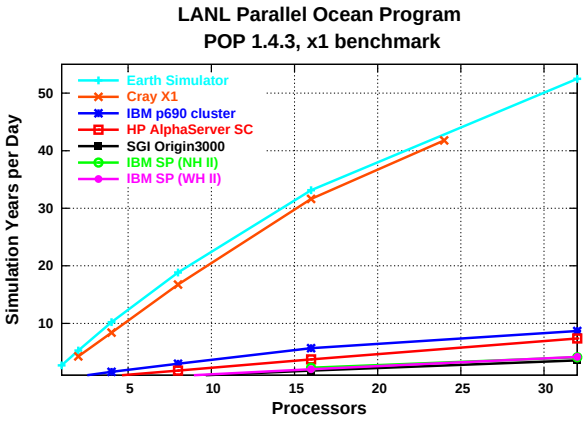

FIGURE 4.23: POP: cross-platform performance comparison

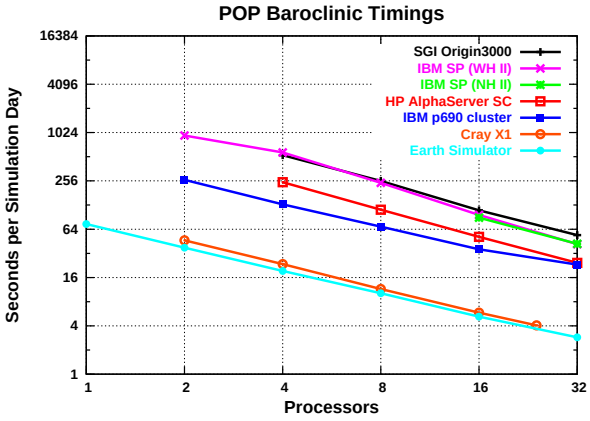

FIGURE 4.24: POP: baroclinic performance

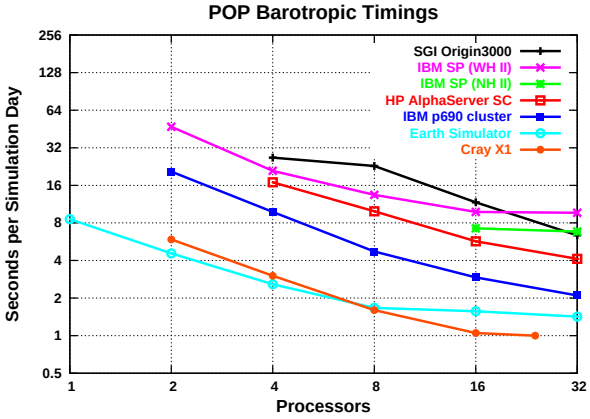

FIGURE 4.25: POP: barotropic performance

From these data, POP on the X1 performs well compared to the nonvector systems, but still lags slightly behind the performance on the Earth Simulator. The time spent in the baroclinic process is nearly identical on the Earth Simulator and the X1. Performance is better in the barotropic process on the X1 than on the Earth Simulator, primarily due to the use of Co-Array Fortran in the conjugate gradient solver. Much of the X1 version of POP is still identical to the Earth Simulator version, there is every indication that X1 performance can be improved.

5 Conclusions

In this very early evaluation of the X1, excellent performance was observed with some of the standard computational benchmarks. Very poor performance was also observed, and it is clear that some tuning will be required of any code that is ported to the system, including vector codes that were designed for traditional vector systems. However, when care is taken to optimize the codes, performance has been excellent when compared to the nonvector systems.

The memory bandwidth in the SMP node is excellent, and both SHMEM and Co-Array Fortran provide significant performance enhancements for codes whose performance is sensitive to latency. It is unfortunate that this is required, and we hope that MPI small-message performance can yet be improved. However, MPI large message performance is very good. With regard to SSP and MSP mode, both approaches work. Which is best for a given application is too sensitive to the application, problem size, and number of processors to make any definitive statements. For a fixed size problem, it is typically best to use fewer, more powerful processors (MSPs), in order to minimize parallel overhead. On the Cray X1, this can be offset by inefficiencies in using the SSPs in an MSP, i.e., streaming overhead or inefficiency. In our limited experience, the compilers are quite effective at identifying and exploiting multistreaming, and coding for vectorization and streaming is often easier than for vectorization alone.

6 Acknowledgements

The NEC SX-6 version of POP and POP performance data on the Earth Simulator were provided by Dr. Yoshikatsu Yoshida of the Central Research Institute of Electric Power Industry. The modification to POP using Co-Array Fortran was provided by John Levesque of Cray. POP performance data for the SGI Origin3000 were provided by Dr. Phillip Jones of Los Alamos National Laboratory. POP performance data for the IBM SP at the National Energy Research Scientific Computing Center

(NERSC) were provided by Dr. Tushar Mohan of NERSC. We also gratefully acknowledge the following institutions for access to their HPC systems:

- Pittsburgh Supercomputing Center for access to the Compaq AlphaServer SC at PSC
- Center for Computational Sciences at ORNL for access to the IBM p690, IBM SP, and Cray X1 at ORNL
- National Energy Research Scientific Computing Center for access to the IBM SP at NERSC

7 About the Authors

Tom Dunigan is a Senior Research Scientist in the Computer Science and Mathematics Division at Oak Ridge National Laboratory. He has been conducting early evaluations of advanced computer architectures since the mid 80's. When not testing new computer systems, he is investigating protocols for high-speed networks. He can be reached at the above US mail address or E-mail: thd@ornl.gov.

Pat Worley is a Senior Research Scientist in the Computer Science and Mathematics Division at Oak Ridge National Laboratory. He has been conducting early evaluations of advanced computer architectures since the early 90's. He also does research in performance evaluation tools and methodologies, and designs and implements parallel algorithms in climate and weather models. He can be reached at the above US mail address or E-mail: worleyph@ornl.gov.

References

[1] D. H. Bailey, T. Harris, R. F. V. der Wijngaart, W. Saphir, A. Woo, and M. Yarrow, *The NAS Parallel Benchmarks 2.0*, Tech. Rep. NAS-95-010, NASA Ames Research Center, Moffett Field, CA, 1995.

[2] 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.

[3] W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, E. Brooks, and K. Warren, *Introduction to UPC and language specification*, Technical Report CCS-TR-99-157, Center for Computing Sciences, 17100 Science Dr., Bowie, MD 20715, May 1999.

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

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

[6] R. Hockney and M. B. (Eds.), *Public international benchmarks for parallel computers, parkbench committee report-1*, Scientific Programming, 3 (1994), pp. 101–146.

[7] 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.

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

[9] MPI Committee, *MPI: a message-passing interface standard*, Internat. J. Supercomputer Applications, 8 (1994), pp. 165–416.

[10] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, *Global Arrays: A portable 'shared-memory' programming model for distributed memory computers*, in Proceedings of Supercomputing '94, Los Alamitos, CA, 1994, IEEE Computer Society Press, pp. 340–349.

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

[12] OpenMP Architecture Review Board, *OpenMP: A proposed standard api for shared memory programming.* (available from http://www.openmp.org/openmp/mp-documents/paper/paper.ps), October 1997.

[13] J. R. Taft, *Achieving 60 GFLOP/s on the production CFD code OVERFLOW-MLP*, Parallel Computing, 27 (2001), pp. 521–536.

[14] A. J. van der Steen, *The benchmark of the EuroBen group*, Parallel Computing, 17 (1991), pp. 1211–1221.

[15] 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.

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

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