The Performance Evolution of the Parallel Ocean Program on the Cray X1 *

P. H. Worley †
Oak Ridge National Laboratory

J. Levesque ‡
Cray Inc.

Abstract

We describe our experiences in repeated cycles of performance optimization, benchmarking, and performance analysis of the Parallel Ocean Program (POP) on the Cray X1 at Oak Ridge National Laboratory. We discuss the implementation and performance impact of Co-Array Fortran replacements for communication latency-sensitive routines. We also discuss the performance evolution of the system software from May 2003 to May 2004, and the impact that this had on POP performance.

1 Introduction

The X1 is the first of Cray's new scalable vector systems [6]. The X1 is characterized by high-speed custom vector processors, high memory bandwidth, and a high-bandwidth, low-latency interconnect linking the nodes. The performance of the processors in the Cray X1 is comparable or superior to that 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.

The Parallel Ocean Program (POP) is the ocean component of of the Community Climate System Model (CCSM) [2], the primary model for global climate simulation in the U.S. POP has proven amenable to vectorization on the NEC SX-6, and the expectation was that the same holds true on the Cray X1. This was demonstrated last year by Jones, et al [12] using the SX-6 port of POP on the X1. In this paper we describe in more detail the performance impact of the SX-6 inspired vectorization and MPI [13] optimizations on the Cray X1. We then describe the performance evolution of the Cray X1 over the past year (May 2003 to May 2004) as additional optimizations were introduced and as the system software on the Cray X1 matured.

2 POP Description

POP is an ocean circulation model derived from earlier models of Bryan [3], Cox [5], Semtner [1] and Chervin [4] in which depth is used as the vertical coordinate. The model solves the three-dimensional primitive equations for fluid motions on the sphere under hydrostatic and Boussinesq approximations. Spatial derivatives are computed using finite-difference discretizations which are formulated to handle any generalized orthogonal grid on a sphere, including dipole [17] and tripole [14] grids which shift the North Pole singularity into land masses to avoid time step constraints due to grid convergence.

Time integration of the model is split into two parts. The three-dimensional vertically-varying (baroclinic) tendencies are integrated explicitly using a leapfrog scheme. The very fast vertically-

*This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

†Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (worleyph@ornl.gov)

‡10703 Pickfair Drive Austin, TX 78750 (levesque@cray.com)

uniform (barotropic) modes are integrated using an implicit free surface formulation in which a preconditioned conjugate gradient solver is used to solve for the two-dimensional surface pressure.

A wide variety of physical parameterizations and other features are available in the model and are described in detail in a reference manual distributed with the code. Because POP is a public code, many improvements to its physical parameterizations have resulted from external collaborations with other ocean modeling groups and such development is very much a community effort. Detailed descriptions of the numerical discretizations and methods are described in the reference manual and in previous publications [7, 8, 11].

3 Cray X1 Description

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 GFlop/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 GFlop/sec for 64-bit operations. The four SSPs share a 2 MB "Ecache".

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. This design represents a compromise between non-vector-cache systems, like the SX-6, and cache-dependent systems, like the IBM p690, with memory bandwidths an order of magnitude less than the saturation bandwidth. The Cray X1's cache-based design deviates from the full-bandwidth design model only slightly. Each X1 processor is designed to have more single-stride bandwidth than an SX-6 processor; it is the higher peak performance that creates an imbalance. A relatively small amount of data reuse, which most modern scientific applications do exhibit, should enable a very high percentage of peak performance to be realized, and worst-case data access should still provide double-digit efficiencies.

The X1 compiler's primary strategies for using the eight vector units of a single MSP are parallelizing a (sufficiently long) vectorized loop or parallelizing an unvectorized outer loop. The effective vector length of the first strategy is 256 elements, like for the NEC SX-6. The second strategy, which attacks parallelism at a different level, allows a much shorter vector length of 64 elements for a vectorized inner loop. Cray also supports the option of treating each SSP as a separate processor.

Four MSPs and a flat, shared memory of 16 GB form a Cray X1 node. The memory banks of a node provide 200 GByte/sec 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 GByte/sec 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 shared memory processor (SMP) node, 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 [16], 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.

4 Experiment Details

In these experiments we used version 1.4.3 of POP as downloaded from the POP code repository at Los Alamos National Laboratory (http://climate.lanl.gov/Models/POP/index.htm).

We used a benchmark configuration (called `x1`) representing a relatively coarse resolution similar to that currently used in coupled climate models. The horizontal resolution is roughly one degree (320x384) and uses a displaced-pole grid with the pole of the grid shifted into Greenland and enhanced resolution in the equatorial regions. The vertical coordinate uses 40 vertical levels with a smaller grid spacing near the surface to better resolve the surface mixed layer. Because this configuration does not resolve eddies, it requires the use of computationally-intensive subgrid parameterizations. This configuration is set up to be identical to the actual production configuration of the Community Climate System Model with the exception that the coupling to full atmosphere, ice and land models has been replaced by analytic surface forcing.

The parallel algorithms in POP are based on a decomposition of the horizontal grid onto a two dimensional virtual processor grid. Thus, for example, when using 8 processors, POP can be run on a 1x8, 2x4, 4x2, or 8x1 grid. The choice of processor grid affects both load balance and comunication overhead. In all results presented in the paper, the optimal processor grid was first identified for each processor count, and only these results were used. In consequence, the processor grids used for a given processor count may vary between platforms.

The performance of POP is primarily determined by the performance of the baroclininc and barotropic processes described earlier. The baroclinic process is three dimensional with limited nearest-neighbor communication and typically scales well on all platforms. In contrast, the barotropic process involves a two-dimensional, implicit solver whose performance is very sensitive to network latency and typically scales poorly on all platforms.

As described later, version 1.4.3 was ported to the Earth Simulator by targeted vectorization and optimization of MPI communications. For the X1, we based our initial optimizations on this Earth Simulator version. Further optimizations came primarily from the introduction of Co-Array Fortran [15] implementations of the latency-sensitive communications in the barotropic process.

POP is instrumented with calls to MPI_WTIME to record the time spent in the basic logical units, such as the baroclinic and barotropic processes. For more detailed performance analyses we used the MPICL profiling and tracing package [9, 19], the Paragraph visualization tool [10], and the Cray Performance Analysis Tool (PAT).

5 Platforms

POP performance results are presented for a Cray X1, the Earth Simulator, an HP AlphaServer SC, an IBM p690 cluster, an IBM SP, and an SGI Altix with the following processor and system specifications.

Processor	Proc Peak (GF/s)	Cache (MB)	Memory/ Proc (MB)
Cray X1			
Cray	12.8	2	4000
Earth Simulator			
ES	8.0	N/A	2000
HP AlphaServer SC			
EV68	2.0	8 (L2)	1000
IBM p690 cluster			
Power4	5.2	1.5 (L2)	1000
IBM SP			
Power3-II	1.5	8 (L2)	1000
SGI Altix			
Itanium2	6.0	6 (L3)	8000

SMP Size	Switch/Network
Cray X1	
4	Cray
Earth Simulator	
8	ES
HP AlphaServer SC	
4	Quadrics QsNet
IBM p690 cluster	
32	SP Switch2 (Corsair)
IBM SP	
16	SP Switch2 (Colony)
SGI Altix	
256	NUMAlink

The IBM p690 cluster, Cray X1, and the SGI Altix are located in the Center for Computational Sciences at Oak Ridge National Laboratory (ORNL). The HP AlphaServer SC is located at the Pittsburgh Supercomputer Center (PSC). The IBM SP is located in the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. The Earth Simulator is housed in the Earth Simulator Center in Yokohama, Japan.

The Cray X1 at ORNL has grown from the initial 32 processor (MSP) system delivered in March 2003, to a 128 MSP system in July 2003, to a 256 MSP system in October 2003, and to a 512 MSP system in June 2004. Results are presented throughout this period, ending with the 256 MSP system in May 2004.

6 Vectorization

Figure 6.1 compares the performance of POP for the fixed size `x1` benchmark on a number of different platforms. The graphs are plots of simulation years per day of computation as a function of the number of processors (MSPs for the Cray X1). The Earth Simulator results are a version of POP that was ported and optimized on the Earth Simulator by Dr. Y. Yoshida of the Central Research Institute of the Electric Power Industry (CRIEPI). The Altix results are for a version of POP that uses the same MPI optimizations used on the Earth Simulator. All other platforms used the stock POP version 1.4.3. (The MPI optimizations used on the Altix do not change the performance on the IBM or HP systems signficantly.) The performance of the stock version of POP is approximately the same on the X1 as on the HP and IBM systems, and much worse than on the Earth Simulator or the SGI Altix.

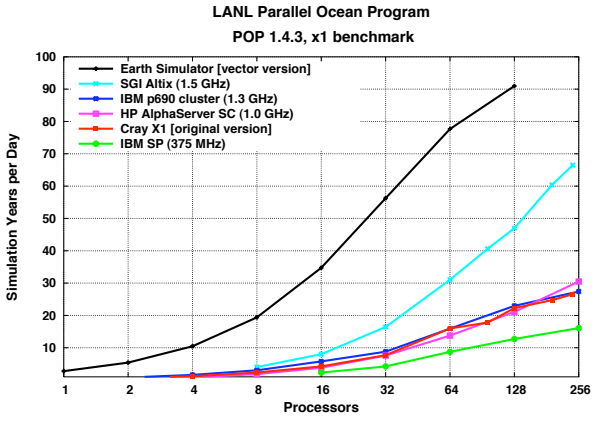


FIGURE 6.1: POP platform comparison: initial results

The first step we took in optimizing POP for the Cray X1 was to exploit the vectorization work already done by Yoshida. For the Earth Simulator port, Yoshida modified 701 lines of POP source code (out of 45000 lines) to improve vectorization. Over half of these modifications (approx. 400 lines) involved replacing Fortran 90 `where`, `merge`, `shift`, etc. constructs with their Fortran 77 equivalents. While these latter changes do not improve (or degrade) performance on the Cray X1, we adopted them anyway to minimize the differences between the two vector versions of the code. We then examined and changed some of the NEC compiler directives to Cray compiler directives. We also modified two routines (that were previously modified for the Earth Simulator) to improve performance on the Cray. Changes in one routine involved a few simple loop reorderings, to enable the Cray to stream over outer loops. Changes in the other routine involved undoing some of the promotion of scalar temporaries to vector temporaries, resulting in code that was closer to the unvectorized version of POP.

Figure 6.2 contains task Gannt charts for POP when using 128 MSPs on the X1. The top chart describes performance for the unmodified version of POP 1.4.3, while the bottom chart describes performance after making the modifications for vectorization. The X-axis is time, where each unit represents 2 milliseconds. The Y-axis is the process (MSP) id. The chart describes the task that each process is executing at a given point in time. The tasks are as follows:

Task	Task Description
0	everything else (primarily tracer updates)
1	baroclinc
2	baroclinic boundary update
3	barotropic (excluding the solver)
4	barotropic boundary update
5	barotropic solver

From these data, vectorization primarily decreased the time spent in the baroclinic process, approximately tripling the performance of this phase of the computation.

Figure 6.3 contains utilization count graphs for the same period of execution time for the original (top) and vectorized (bottom) versions of POP. The Y-axis is the number of processes in one of three states: `busy` (computing), `overhead` (actively communicating in an MPI command), `idle` (blocked waiting to receive a message). While subject to some error, `overhead` + `idle` does accurately reflect the time spent in MPI commands. From these data, most of the time in a 128 MSP run is spent in interprocessor communication. Also, vectorization improved performance of the computation in the baroclinic process by approximately a factor of 10. The performance of the baroclinic process as a whole increased by a factor of only three because the baroclinic communication overhead was unchanged.

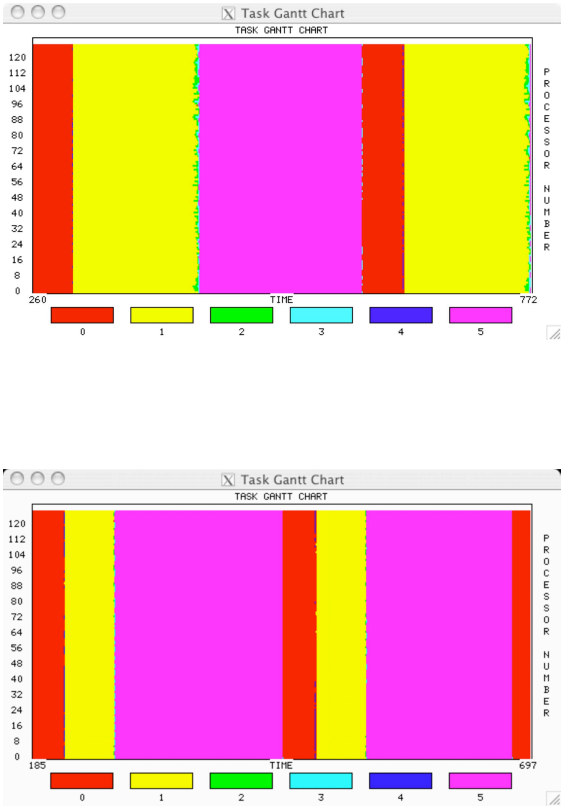


FIGURE 6.2: Task graph: without and with vectorization

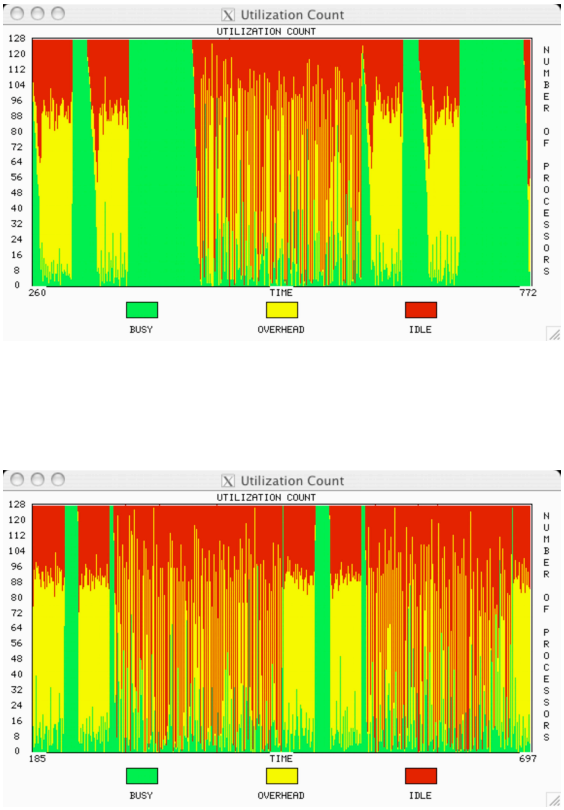


FIGURE 6.3: Utilization count: without and with vectorization

7 MPI Optimizations

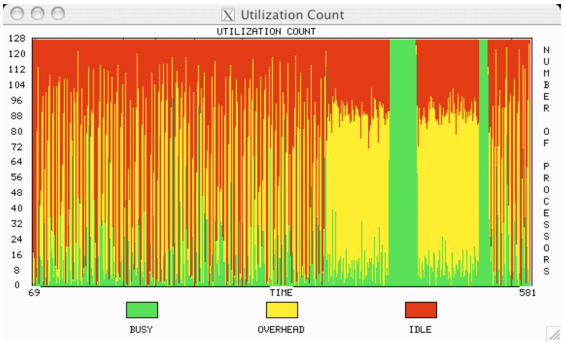
It is clear from Fig. 6.3 that communication costs dominate in the 128 MSP performance data. The next step in the Cray X1 optimization was to minimize these costs. While there was no reason to expect that the Earth Simulator-specific MPI optimizations to be useful on the X1, it was an easy experiment to make. In fact, these modifications improved performance on the Cray X1 significantly.

For the Earth Simulator, Dr. Yoshida modified 125 lines and added one new (140 line) routine to improve MPI performance. In the original version of POP, three routines in the barotropic process used MPI derived datatypes when updating halo regions in the domain decomposed data structures. This logic was replaced with explicitly packing and unpacking contiguous communication buffers and using standard datatypes in MPI calls. In the baroclinic process and in the `baroclinic_correct_adjust` routine, which updates tracers, a loop over a series of halo updates was replaced by a single routine that combined these into fewer communication calls involving larger messages. Finally, five routines were modified to replace communication logic using `MPI_IRECV` and `MPI_ISEND` with similar logic using `MPI_ISEND` and `MPI_RECV`. This latter modification does not improve (or degrade) performance on the X1, but was again adopted in order to minimize differences with the Earth Simulator version of POP.

Figure 7.1 contains the task Gannt chart for the vectorized version, without (top) and with (bottom) these MPI optimizations. The X-axis resolution is twice that used in in Figs. 6.2 and 6.3, with 1 millisecond units. Figure 7.2 contains the corresponding utilization count graphs. From these data, eliminating derived types triples the performance of the barotropic solver. Combining the halo updates almost eliminates the time spent doing tracer updates and more than halves the time spent in the baro-

clinic process. From the utilization count graphs, POP on the X1 is still communication bound when using 128 MSPs for this benchmark problem, but the performance is much improved.

Figure 7.3 demonstrates the performance impact of vectorization and MPI optimization on the X1. Using this modified version of the Earth Simulator port, performance on the X1 is similar to that on the Earth Simulator up to 96 processes. This version of the code is very similar to that used in the earlier paper [12]. Performance reported here is superior due to OS and other system software improvements over the past year (April 2003 to May 2004). This will be discussed in more detail in later sections.



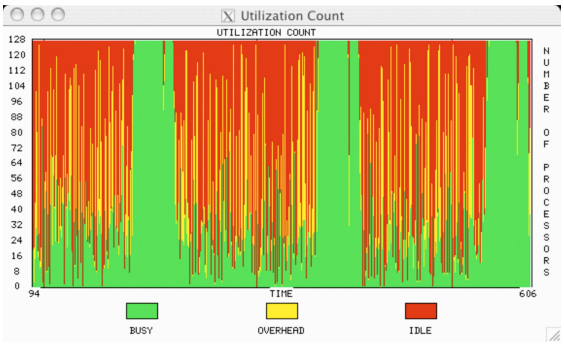
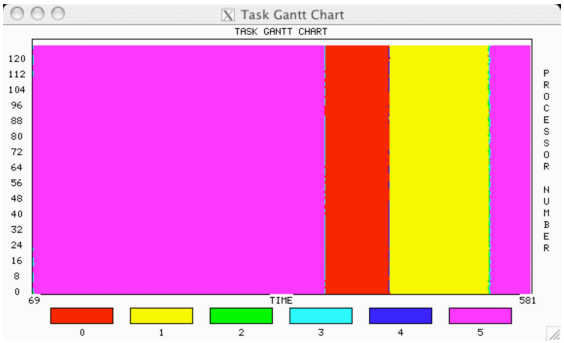


FIGURE 7.2: Utilization count: without and with MPI optimization



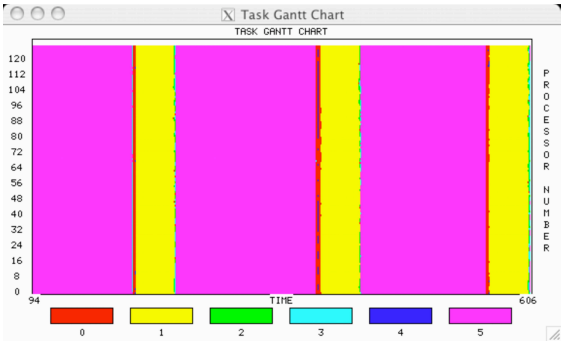


FIGURE 7.1: Task graph: without and with MPI optimization

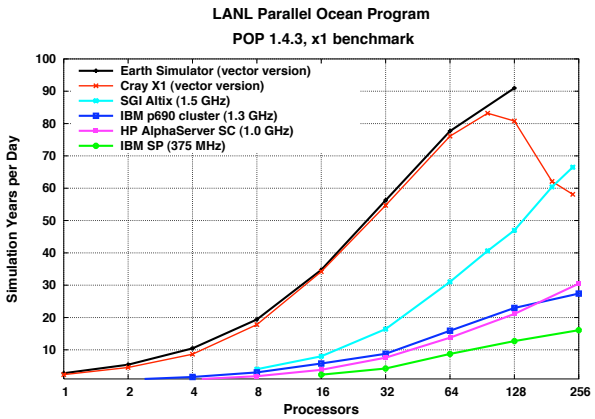


FIGURE 7.3: Platform comparison: with vectorization and MPI optimization

8 Co-Array Fortran Optimizations

Process scaling for a fixed size problem will necessarily show diminishing returns for large process counts. However, the feeling was that the *rollover* in X1 performance in Fig. 7.3 occurred too soon. Performance before the rollover is also lower than expected compared to the Earth Simulator given the peak processor and network rates on the Cray X1 and the Earth Simulator. From Fig. 7.2, communication overhead is the dominant constraint on performance. Profiling indicates that dominant communication overhead in this port of POP to the X1 is from

- `GLOBAL_SUM`: global sum of the "physical domain" of a two dimensional array. This is used to compute the inner product in the conjugate gradient solver in the barotropic process. The MPI version uses MPI_Allreduce.

- `NINEPT_4`: weighted nearest neighbor sum for 9 point stencil, requiring a halo update. This is used to compute residuals in the conjugate gradient solver in the barotropic process.

The performance of both of these is sensitive to MPI latency. MPI performance for small messages is a known problem on the X1, and the current solution is to use Co-Array Fortran to implement latency-sensitive communications. Co-Array Fortran more efficiently exploits the globally addressable memory on the Cray than is currently possible with MPI two-sided messaging semantics. For example, Fig. 8.1 is a graph of performance of a halo update (using Wallcraft's HALO benchmark [18]) for 16 MSPs as a function the number of four byte words in the halo. For small halos, the Co-Array Fortran implementation is over 5 times faster.

MPI and Co-Array Fortran messaging can coexist in the same program. For POP we replaced the existing inner product and halo update subroutines used in barotropic solver with Co-Array Fortran implementations. While performance improved with our initial implementations, it was not as good as we expected. Over the next 8 months (May 2004 to December 2004) we developed and tested a number of different implementations:

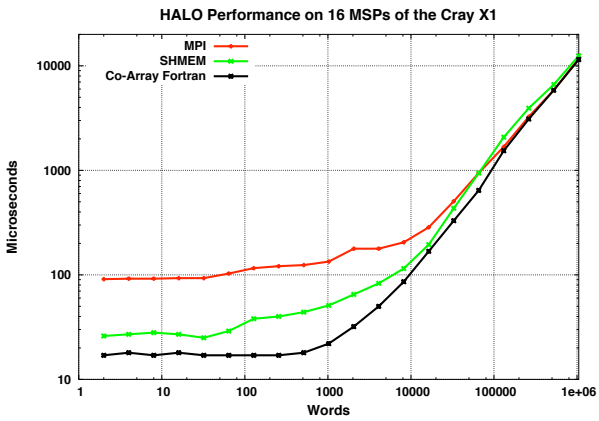


FIGURE 8.1: Paradigm Comparison for Halo Update

May 7

- `GLOBAL_SUM`: Master reads partial sums from remote memory, completes sum, and writes results back to other processes' memory. Global barriers are used for synchronization.

- `NINEPT_4`: Each process reads remote memory to update the local halo. East-west updates occur first, followed by north-south updates. Global barriers are used for synchronization.

August 12

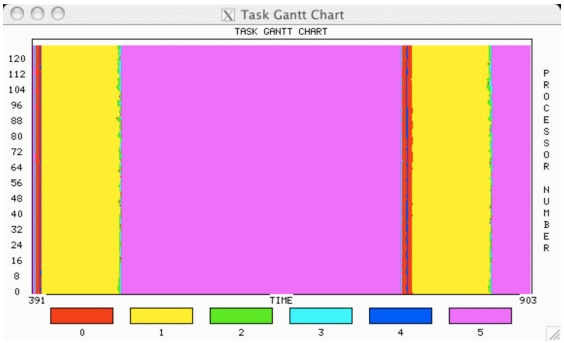
- `GLOBAL_SUM`: Processes write partial sums to master's memory. Flags are used to implement pairwise synchronization, eliminating the need for global barriers.

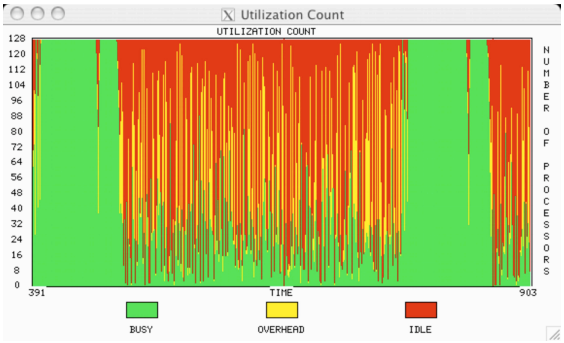
- `NINEPT_4`: Each process writes remote memory to fill remote halos. East-west updates occur first, followed by north-south updates. Flags are used to implement pairwise synchronization, eliminating the need for global barriers.

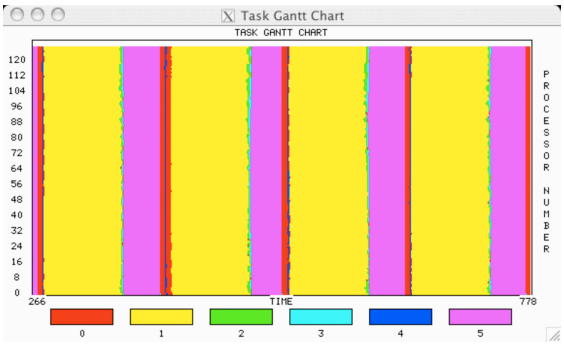
September 1

- `GLOBAL_SUM`: Added padding to co-arrays in August 12 version to eliminate contention in the memory controller.

- `NINEPT_4`: Added padding to co-arrays in August 12 version to eliminate contention in the memory controller.







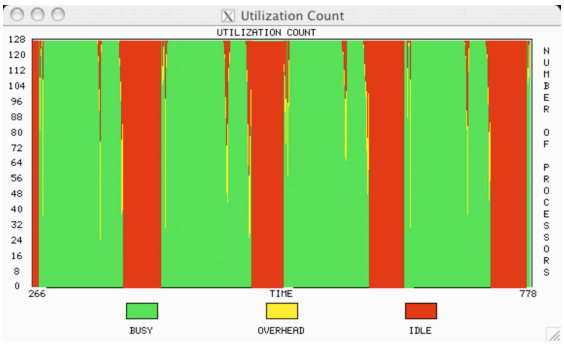


FIGURE 8.2: Task graph: without and with Co-Array Fortran optimization

FIGURE 8.3: Utilization count: without and with Co-Array Fortran optimizations

October 25

- `GLOBAL_SUM`: Used special values in data co-arrays to implement pairwise synchronization in September 1 version, eliminating flags.
- `NINEPT_4`: Used September 1 algorithm.

December 13a

- `GLOBAL_SUM`: Used a variant of the May 7 version in which partial sums are written to the master's memory. Global barriers are again used for synchronization.
- `NINEPT_4`: Used August 12 algorithm.

December 13b

- `GLOBAL_SUM`: Used a generalization of the September 1 algorithm that implements a tree sum and tree broadcast. The branching factor is specified at compile time. For the results described here, each parent has up to 16 children.
- `NINEPT_4`: Used August 12 algorithm.

Figures 8.2-8.4 compare the performance of the optimized MPI-only version with the optimized version using the December 13b Co-Array Fortran modifications. Figures 8.2 and 8.3 are the usual task Gannt chart and utilization count graphs. The X-axis resolution is again doubled from the previous figures, with .5 millisecond units. The Co-Array Fortran modifications were used only in the barotropic solver, and this is the only task that shows any performance change. However, when using 128 MSPs, the barotropic solver is 5 times faster when using the Co-Array Fortran modifications, and is no longer communication bound. (In Fig. 8.3, the time spent in `GLOBAL_SUM` and `NINEPT_4` is defined to be idle, as this is closer to reality than defining it to be

busy. Fine grain tracing of the Co-Array Fortran would perturb performance too much to be useful.) Figure 8.4 is task Gannt chart focusing on the barotropic solver. The X-axis unit is 1 microsecond. The three tasks are 1: `GLOBAL_SUM`, 2:`NINEPT_4`, and 0: everything else. From this, the Co-Array Fortran implementation is 8 times faster than the MPI implementation for `GLOBAL_SUM` and 4 times faster for `NINEPT_4`.

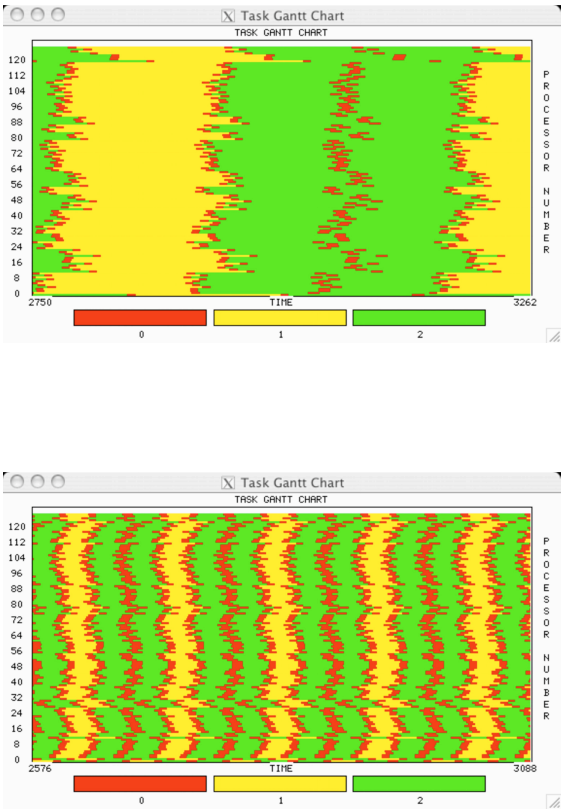


FIGURE 8.4: Task graph of barotropic solver: without and with Co-Array Fortran optimization

9 System Software Optimizations

As mentioned earlier, the development and evaluation of the Co-Array Fortran implementations took place over 8 months. Figure 9.1 is a graph of the performance evolution that we have observed over the past year using the best performing version of POP at each point in time. While the performance of POP motivated the algorithm experimentation, some of the performance improvement did not arise from algorithm improvements. This can be seen in the performance improvement in the December 13b version between March 4 and May 11, 2004. The source of each non-algorithmic performance improvement varies with the particular update to the system software, but one of the largest changes was made in a September 1 update to the operating system. This is displayed in Fig. 9.2. On August 29, the August 12 version of POP achieved a maximum performance of 85 years per day, but the average performance was much lower. The variability of POP timings was very high. This behavior was used to document a problem in the way that the operating system scheduled interrupts. By using a global clock to schedule interrupts on all processors, the September 29 performance was achieved. The best performance increased from 85 years per day to 125 years per day, and the performance variability decreased significantly.

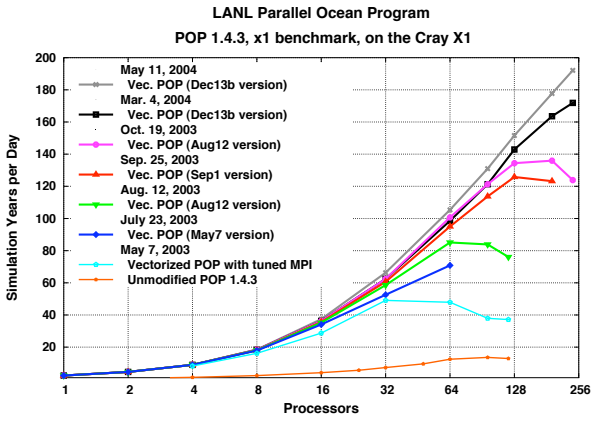


FIGURE 9.1: Performance evolution

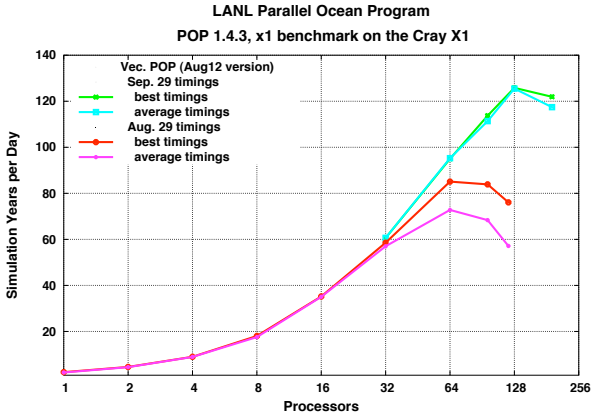


FIGURE 9.2: Performance Impact of Sept. 1 OS Update

Figure 9.3 is a graph of the performance of the August 12 implementation up to May 11, 2004. Figure 9.4 is a graph of the performance of the optimized MPI-only implementation for the same period. In both cases, it is apparent that the performance of the operating system and system software has continued to improve. In particular, while the MPI-only version is still not competitive with the version using Co-Array Fortran optimizations, MPI performance has improved significantly.

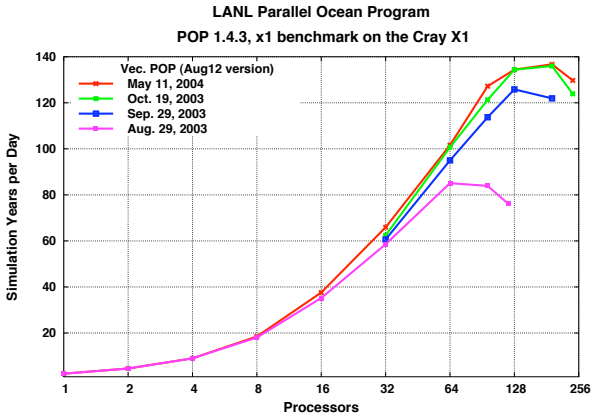


FIGURE 9.3: Performance Impact of OS Updates on Aug. 12 version

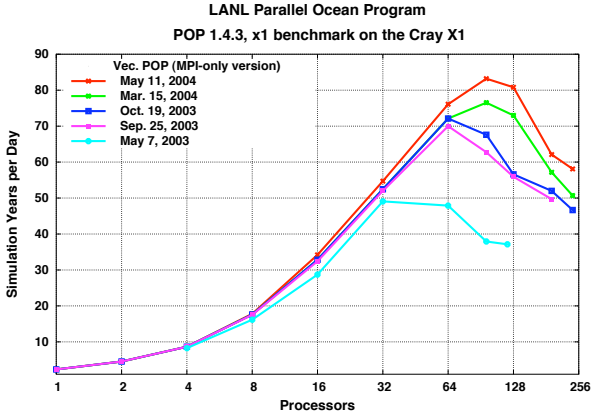


FIGURE 9.4: Performance Impact of OS Updates on MPI-only version

Figure 9.5 is a graph of the performance of all of the POP implementation described so far as of May 11, 2004. Notice that the Co-Array Fortran optimizations prior to December 13 have very similar performance. The attempt to solve what were ultimately performance bottlenecks in the operating system by algorithm optimization was not very effective. It was not until a number of operating system problems were resolved that significant progress could be made. For example, a tree-based algorithm similar to that used in the December 13b version was tried in September. It was not competitive at that time, and was discarded.

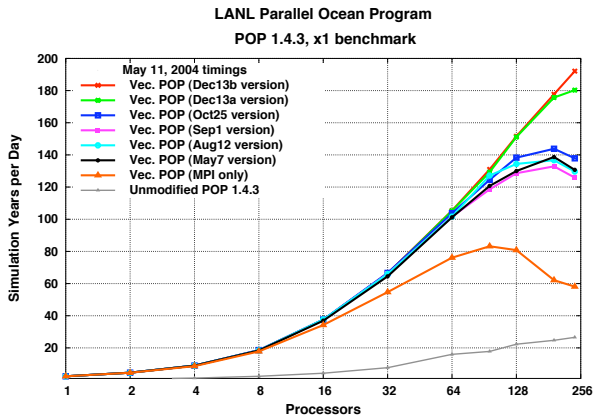


FIGURE 9.5: Implementation Comparison

Figure 9.6 is a graph of the performance of just the barotropic process for all of the POP implementations (in terms of wallclock seconds per simulation day). The advantage of the December 13 versions of POP is their scalability. The time spent in the barotropic is beginning to increase for large process counts for all of the other versions of POP. For the December 13 versions, the time is relatively constant for 64 to 240 processes. Figure 9.7 is a graph of the wallclock seconds per simulation day for both the barotropic and baroclinic processes, for both the optimized MPI-only and December 13b versions. The performance (and code) for the baroclinc process is identical for these two implementations, and is scaling well out to 240 processes. For the MPI-only version, the time spent in the barotropic process starts growing at 16 processes and dominates that of the baroclinc when using more than 92 processes. In contrast, for the December 13b version, time spent in barotropic process stops decreasing at 92 processes, but has not yet started growing at 240 processes. It is also only one third the time spent in the baroclinic process when using 240 processes. From this data, performance should continue to scale to 256 processes and beyond.

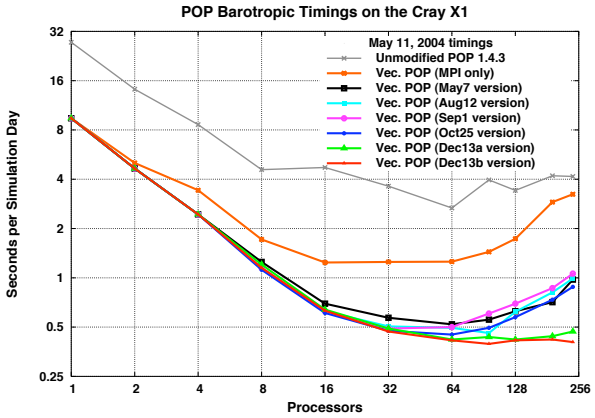


FIGURE 9.6: Implementation Comparison: Barotropic Process

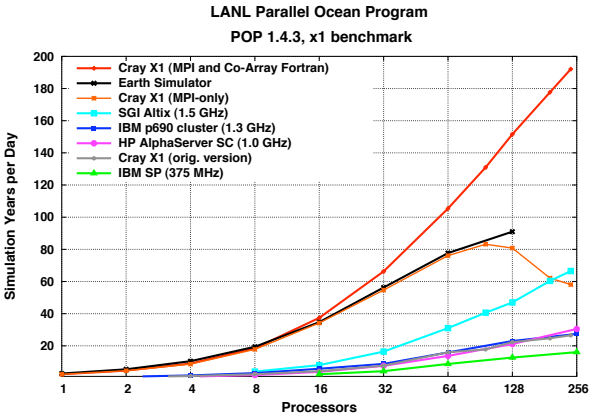


FIGURE 10.1: Platform comparison: current results

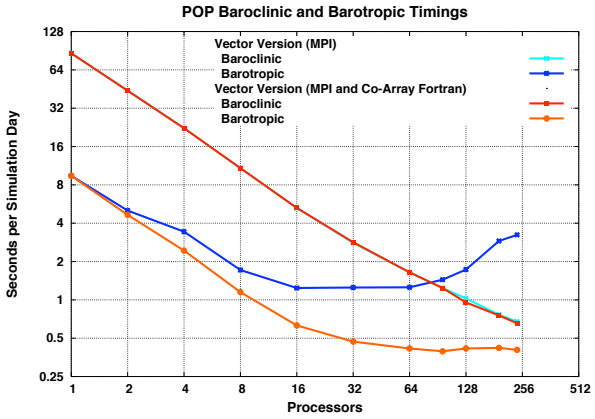


FIGURE 9.7: Implementation Comparison: MPI-only vs. Dec13b

10 Platform Comparisons

Figure 10.1 is the current platform comparison using the `x1` benchmark problem and version 1.4.3 of POP. It includes performance data for the original version, the optimized MPI-only version, and the December 13b version of POP on the Cray X1. Performance on the X1 is clearly better than that on the IBM and HP systems. In particular, no matter how many processors are used on the IBM and HP systems (for this fixed size benchmark), performance will never approach that on the X1. Only performance on the Earth Simulator and the SGI Altix among these platforms approach the performance on the X1.

Figure 10.2 contains graphs of the performance of the barotropic and baroclinic processes on the X1 and the Earth Simulator. From these data, Earth Simulator performance is better for the baroclinic for the smallest process counts, and worse for the largest process counts. This (roughly) indicates a performance advantage on the Earth Simulator for long vectors and an advantage on the X1 for short vectors. As the vectorization in POP is basically that developed for the Earth Simulator, this may also indicate that more Cray-specific optimizations are needed. For the barotropic process, the X1 performance is superior when using more than 8 processes. The Earth Simulator has 8-processor SMP nodes, and messaging between SMP nodes is required when using more than 8 processors. The performance of the barotropic process on the Earth Simulator is relatively constant out to 128 processes, but it also begins dominating the baroclinc process at this point. Using more than 128 processors on the Earth Simulator is unlikely to improve performance for this benchmark problem. Note that the Earth Simulator performance data are from March, 2003. The Earth Simulator implementation could also have continued to evolve, perhaps by using MPI-2 1-sided messaging in the barotropic solver, but we are not aware of more recent performance data.

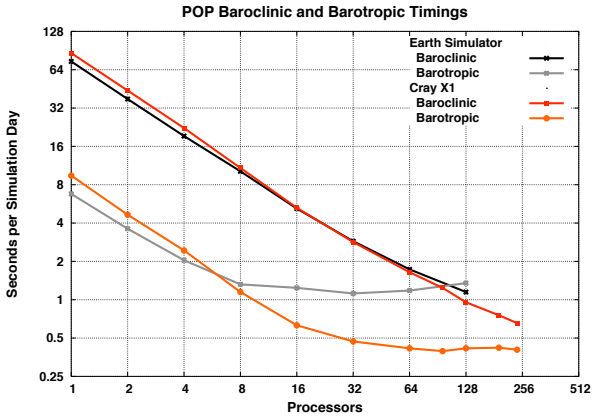


FIGURE 10.2: Platform comparison: X1 vs. ES

Figure 10.3 contains graphs of the performance the barotropic and baroclinic processes on the X1 and the SGI Altix. The X1 is approximately 5 times faster than the Altix for the baroclinic process when using 8 processors, and 3 times faster when using 240 processors. The shorter vector lengths are decreasing the performance advantage of the vector processor over that of the Itanium2. The X1 is also approximately 3 times faster than the Altix for the barotropic process when using 240 processors. Time spent in the barotropic is relatively flat for 64 to 240 procssors. So, while both systems will likely be able to use more processors effectively, the SGI does not appear to be able to approach closer than one third of the performance of the X1. Note that POP version 1.4.3 comes with an implementation that uses SHMEM for interprocessor communication. This version does not improve POP performance on the SGI. Other optimizations may also be possible, but we are already using the Yoshida MPI optimizations on the Altix.

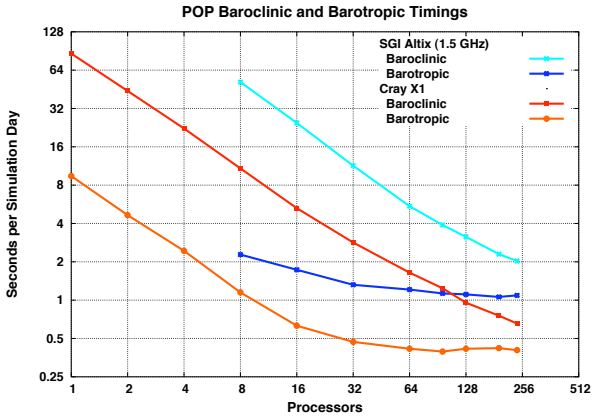


FIGURE 10.3: Platform comparison: X1 vs. Altix

11 Conclusions

We learned a number of important lessons from optimizing and tracking POP performance on the Cray X1. First, vector and MPI optimizations developed for the Earth Simulator were reasonably efficient on the X1, and were a good starting point when optimizing for the Cray. While some of these optimizations did not enhance performance, they also did not degrade performance. However, the advantage of the Earth Simulator for long vector performance indicates that we need to revisit Cray-specific vectorization, especially for larger problem sizes. Second, explicit packing and unpacking of message buffers and using standard data types performs much better than using MPI derived datatypes to do this implicitly (currently). Third, the performance of latency-sensitive MPI collective and point-to-point commands limits scalability of latency-sensitive codes (again, currently). Co-Array Fortran can be used to work around this deficiency in MPI performance and achieve excellent scalability. Fourth, performance aspects of the operating system and other system software improved significantly from May 2003 to May 2004. Finally, we found tracking the performance evolution of POP to be an effective way of identifying performance problems in the operating system and other system software.

Future work includes optimization and benchmarking for even higher processor counts and for larger benchmark problems. In particular, POP will be used for additional investigations into Co-Array Fortran algorithms and performance, operating system scalability, and MPI performance as the system software continues to evolve.

12 Acknowledgements

We gratefully acknowledge

- Dr. Phil Jones of Los Alamos National Laboratory for help in obtaining and porting POP and for defining benchmark problems;
- Dr. Tushar Mohan of Lawrence Berkeley National Laboratory for providing performance data from the IBM SP at NERSC;
- Howard Pritchard and James Schwarzmeier of Cray Inc. and James B. White III of ORNL for their contributions to the development of the Co-Array Fortran algorithms used in POP;

- Dr. Yoshikatsu Yoshida of CRIEPI for his excellent port of POP to the Earth Simulator and for providing the performance data from the Earth Simulator.

We also thank the Pittsburgh Supercomputer Center for access to the HP AlphaServer SC and the ORNL Center for Computational Science for access to the Cray X1, IBM p690 cluster, and SGI Altix.

13 About the Authors

John Levesque is a 30 year veteran of High Performance Computing. Levesque is Technical Staff to the VP of Cray Development and is responsible for demonstrating performance capabilities of Cray's HPC systems. In addition to porting and optimizing applications for Cray platforms, Levesque gives training workshops on the effective utilization of Cray Hardware. He can be reached at the US mail address listed on the title page or via E-mail: levesque@cray.com.

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 US mail address listed on the title page or via E-mail: worleyph@ornl.gov.

References

[1] J. A. Semtner Jr., *Finite-difference formulation of a world ocean model*, in Advanced Physical Oceanographic Numerical Modeling, J. J. O'Brien, ed., Reidel Publishing Company, Norwell, MA, 1986, pp. 187–202. NATO ASI Series.

[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] K. Bryan, *A numerical method for the study of the circulation of the world ocean*, J. Comp. Phys., 4 (1969), pp. 347–376.

[4] R. M. Chervin and J. A. Semtner Jr., *An ocean modeling system for supercomputer architectures of the 1990s*, in Proceedings of the NATO Advanced Research Workshop on Climate-Ocean Interaction, M. Schlesinger, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1988, pp. 565–568.

[5] M. D. Cox, *A primitive equation, 3-dimensional model of the ocean*, Tech. Rep. GFDL Ocean Group Technical Report No. 1, GFDL/NOAA, Princeton, NJ, 1984.

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

[7] D. K. Dukowicz and R. D. Smith, *Implicit free-surface method for the bryan-cox-semtner ocean model*, J. Geophys. Res., 99 (1994), pp. 7991–8014.

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

[9] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, *A users' guide to PICL: a portable instrumented communication library*, Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.

[10] M. T. Heath and J. A. Etheridge, *Visualizing performance of parallel programs*, Tech. Rep. ORNL/TM-11813, Oak Ridge National Laboratory, Oak Ridge, TN, May 1991.

[11] 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.

[12] P. W. Jones, P. H. Worley, Y. Yoshida, J. B. W. III, and J. Levesque, *Practical performance portability in the Parallel Ocean Program (POP)*, Concurrency and Computation: Experience and Practice, ((*in press*)).

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

[14] R. J. Murray, *Explicit generation of orthogonal grids for ocean models*, J. Comp. Phys., 126 (1996), pp. 251–273.

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

[16] 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.

[17] R. D. Smith, S. Kortas, and B. Meltz, *Curvilinear coordinates for global ocean models*, Tech. Rep. LAUR-95-1146, Los Alamos National Laboratory, Los Alamos, NM, 1995.

[18] 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.

[19] P. H. Worley, *MPICL.* http://www.csm.ornl.gov/picl/.