Early Evaluation of the IBM BG/P *

P. H. Worley

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

Abstract. This paper describes early results of a performance evaluation of the IBM BG/P recently installed at Oak Ridge National Laboratory. We use microkernels to determine computation and communication performance, both with and without contention. We use the Parallel Ocean Program to evaluate application scalability and to distinguish BG/P performance from other High Performance Computing systems.

1 Introduction

In late October 2007, a two-rack IBM BlueGene/P system was installed at Oak Ridge National Laboratory (ORNL). Each rack consists of 1024 compute nodes, where each compute node contains four microprocessor cores and 2 GB of shared RAM. Each core is an 850 MHz PowerPC 450 32-bit microprocessor with a 64-bit dual-pipe floating point multiply-add unit (double FMA) that can deliver four floating point operations per cycle. Each compute node runs a lightweight kernel to execute user-mode applications only.

Compute nodes are connected via six networks, four of which are of importance to user applications.

- Three-dimensional torus network for point-to-point messaging between compute nodes. Each node has 6 connections to the torus network, where each link has a peak bidirectional bandwidth of 425 MB/s.
- Global collective network. Each node has 3 connections to the collective network, where each link has a peak bidirectional bandwidth of 850 MB/s.
- Global barrier and interrupt network. Each node has 4 connections to the interrupt network.

* This research was sponsored by the Climate Change Research Division of the Office of Biological and Environmental Research and by the Office of Mathematical, Information, and Computational Sciences, both in the 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.

– 10 Gigabit Ethernet. The Ethernet network consists of all I/O nodes connected via a standard 10 Gigabit Ethernet switch. Compute nodes are not connected to this network directly.

This paper describes a subset of the performance data being collected by performance researchers and application developers at ORNL. (The full evaluation will be described in an ORNL technical report. The current draft of the technical report is over 70 pages long, and so is not appropriate for this forum.) We use kernels to examine basic computation performance and to evaluate the impact of contention within the quad-core compute node. We use MPI [1] benchmarks to determine basic point-to-point messaging performance, both with and without contention. Finally, we use one application code, the Parallel Ocean Program, to examine application scalability and to illuminate some significant performance differences between the IBM BG/P and other High Performance Computing (HPC) systems such as the Cray XT series. As this is an early evaluation, performance will change as the software stack evolves, and we expect that performance for some of the benchmarks reported here will improve. In addition, there are many settings (e.g. process mappings and system environment variables) that affect performance and that we have not fully exercised in these initial studies. However, many of the basic performance characteristics are obvious from these experiments.

2 Terminology and Experiment Details

To exploit both pipes in the floating point unit requires issuing special SIMD instructions. If a code is compiled with `-qarch=450d` and a sufficiently high level of optimization (`-qhot` or `-O4` or `-O5`), the compiler will attempt to generate these instructions.

The compute node kernel supports three different modes of execution:

– `SMP`: assign only one MPI process to a node;
– `DUAL`: assign two MPI processes to a node;
– `VN`: assign four MPI processes to a node.

For `SMP` and `DUAL` modes, OpenMP [2] parallelism can be used to generate threads of execution for the unused cores. By default, processes are mapped to compute nodes in `XYZT` ordering, i.e., assigning one process to each node in the `X` direction of the torus, then the `Y`, then the `Z`, then returning to the first node and assigning a second process, etc. For most of our experiments we also used the `TXYZ` ordering, which assigns processes 0-3 to the first node, 4-7 to the second node (in the `X` direction), etc. Finally, for some experiments we mapped each process to a compute node and to a core in that node explicitly using a mapping file. Note that `VN` mode and an explicit mapping of processes to processors can also be used to emulate `SMP` and `DUAL` modes.

By default, some MPI collectives have been optimized to take advantage of the collective global and interrupt networks, or of special features of the torus

network. The optimized versions can be disabled by setting the environment variable `DCMF_COLLECTIVE` to 0, and reenabled by setting it to 1.

There are many MPI environment variables, including many that are specific to the BG/P, that can affect performance. In this paper the only MPI environment variable with which we have experimented is `DCMF_COLLECTIVE`.

Most BG/P performance data presented here were collected over a 3 month period (November 2007 through January 2008). A number of bad nodes were identified and replaced during this time. A number of software patches were also installed. After major changes we reran a subset of the experiments to determine whether performance was affected, and we present the latest results here. During January 2008, the BG/P system at ORNL was running Linux 2.6.16.27-193 on the compute nodes and using version mpich-1.2.7p1-15.4 of the MPI library, version 4.3.1-0 of the ESSL math library, version 11.01 of the XL Fortran compiler, and version 9.00 of the XL C/C++ compiler.

For comparison purposes, we also present performance data collected on a Cray XT4, a Cray X1E, and a Xeon cluster sited at ORNL and on an IBM p575 cluster at the National Energy Research Scientific Computing Center (NERSC). The Cray XT4 data were collected in the Spring of 2007 on a system with 6296 compute nodes running the Catamount light weight kernel. Each compute node is a 2.6 GHz dual-core Opteron processor with 4 GB of memory, where each processor core is capable of 5.2 GFlop/s for 64-bit operations. The XT4 interconnect is a custom, three-dimensional toroidal network utilizing the Cray SeaStar network interface controller to connect the compute node, via HyperTransport, with the network. Further details on the XT4 and its performance can be found in [3] and [4]. The Cray X1E data were collected in the Summer and Fall of 2005 on a system consisting of 1024 Multi-Streaming Processors (MSP), each capable of 18 GFlop/s for 64-bit operations. MSPs are fully connected within 32-MSP subsets, and are connected via a 2-D torus between subsets. Additional details on the X1E and its performance can be found in [5]. The Xeon cluster data were collected in December 2005 on a system with 80 nodes interconnected with Gigabit Ethernet and running Linux. Each node consists of a 3.4 GHz dual-core Intel Xeon processor and 4 GB of memory, where each processor core is capable of 6.8 GFlop/s for 64-bit operations. The IBM p575 cluster data were collected in the Fall of 2006 on a system with 122 8-way p575 SMP nodes (1.9 GHz POWER5 processors) and an HPS interconnect with 1 two-link adapter per node. Each processor is capable of 7.6 GFlop/s for 64-bit operations.

3 Computation Kernels

3.1 Matrix Multiply

Figure 1 describes the double-precision floating point performance of a matrix multiply using the DGEMM [6] routine from the IBM ESSL library. Matrix multiply has a high ratio of floating point operations to operands and good register and cache locality, when implemented carefully. A DGEMM benchmark

is often used to define the achievable peak performance of a processor. In these experiments the matrix multiply was executed on a single core, on two cores within the same compute node, and on all 4 cores. In the multi-core experiments each core executed identical instances of the benchmark simultaneously. The per-core performance is graphed in Fig. 1 as a function of matrix size, indicating no performance degradation from contention between the simultaneous executions. The peak performance observed was approximately 2.7 GFlop/sec, or 79% of the peak when using both pipes on the floating point unit. (This performance is greater than peak when using only one pipe, so the ESSL library is clearly able to exploit both floating point pipes.)

Figure 2 compares the BG/P results with similar experiments on a dual-core Cray XT4 node and on an 8-way IBM p575 node. Here the Opteron processor core in the XT4 node achieves 90% of peak on this benchmark and the POWER5 processor in the p575 node achieves 95% of peak. Figure 3 compares the performance of the ESSL DGEMM routine with that of a simple three loop Fortran implementation of the matrix multiply. The original Fortran version was also modified to reorder the two outer loops and to remove a scalar temporary. These results show the gap between user code performance and a good library implementation, and also indicate an inability in the current version of the compiler to permute loops and other scalar optimizations despite using aggressive levels of compiler optimization.

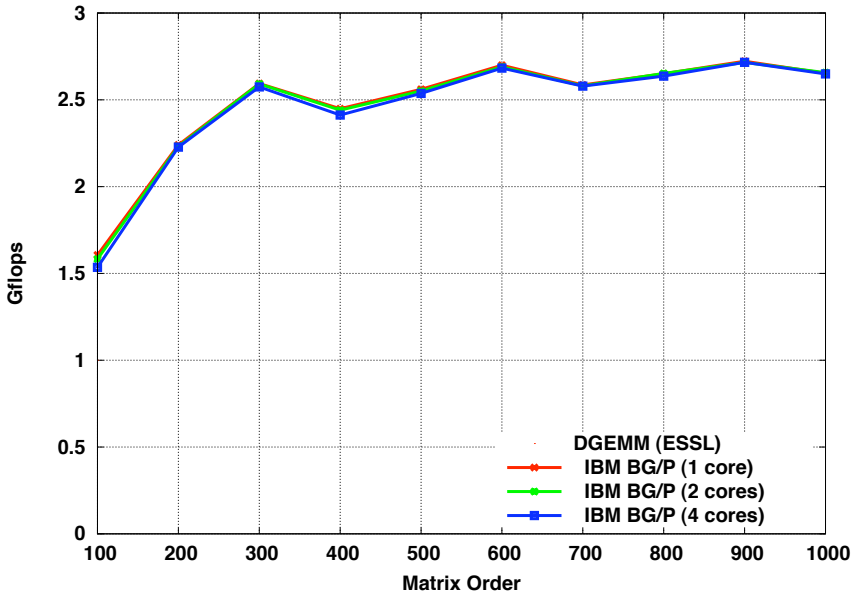

Fig. 1. Matrix Multiply (DGEMM) Performance

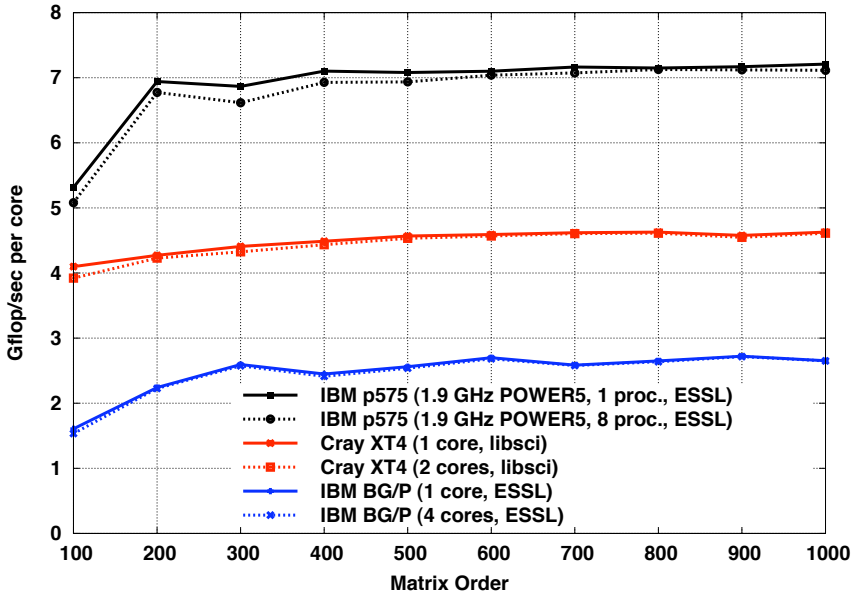

Fig. 2. Matrix Multiply (DGEMM) Platform Comparison

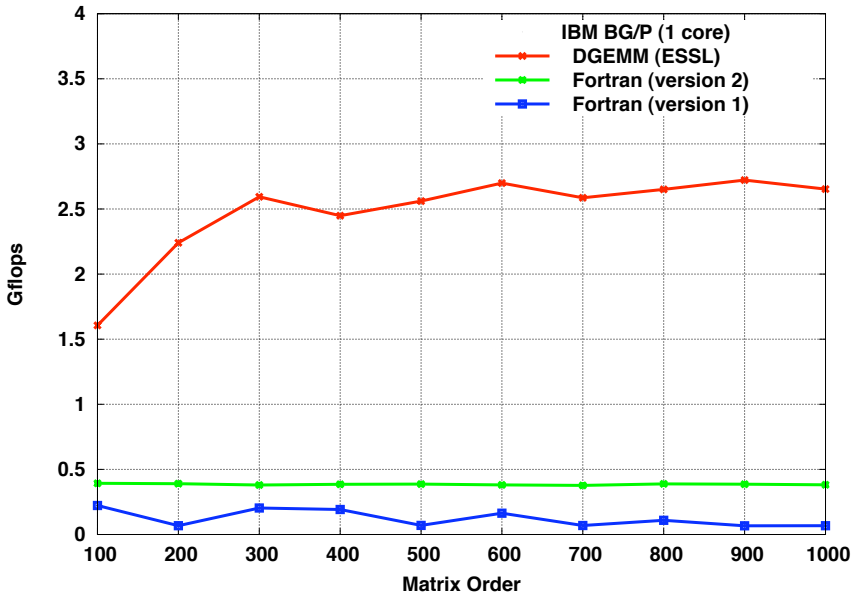

Fig. 3. Matrix Multiply (DGEMM and Fortran) Performance

3.2 PSTSWM

The Parallel Spectral Transform Shallow Water Model (PSTSWM) [7, 8] represents an important computational kernel in spectral global atmosphere models. PSTSWM exhibits little reuse of operands as it sweeps through the field arrays; thus it exercises the memory subsystem as the problem size is scaled and can be used to evaluate the impact of memory contention in multi-core nodes. All array sizes and loop bounds in PSTSWM are determined at runtime, which limits the effectiveness of some compiler optimizations.

Figure 4 compares the performance of PSTSWM on a single core when using a number of different compiler optimizations. The computational rate is plotted as a function of an increasing number of vertical levels for a fixed horizontal grid resolution of 128x256 (*T85*). The vertical level index is second (between the longitude and latitude indices), and the tightest coupling is in the longitude and latitude directions. Thus increasing the number of vertical levels increases the traffic to main memory. These data describe performance when using routines in the ESSL library to compute Fourier transforms. From these results, we see that specifying a "strict" interpretation of the numerics (`-qstrict`) degrades performance by approximately 20%. Overall `-O3` with higher order transformations (`-qhot`) produces the fastest code when applied globally, but the advantage over `-O2` optimization is not great. In data not shown here, experiments using Fortran implementations of the (fast) Fourier transform operators show qualitatively similar results. However, using the ESSL Fourier transform routines improves performance by approximately 30% for this problem resolution, reflecting a much greater increase in the performance of the FFT operators. (Note that the ESSL FFT routines may be using both floating point pipes. In these experiments the Fortran implementations are not.)

In other data not shown here, compiling with `-qarch=450d`, i.e., telling the compiler to attempt to use the second pipe of the floating point unit, decreases the best performance for the T85 benchmark by approximately 5% when compared to compiling with `-qarch=450`. In contrast, T85 performance when using strict numerics was nearly 10% better when compiled with `-qarch=450d` than with `-qarch=450`. This is still 10% slower than the best performance when compiling with `-qarch=450`.

Figure 5 compares the performance on a single core as a function of the number of vertical levels for a number of different horizontal resolutions: T5 (8×16 horizontal computational grid), T10 (16×32), T21 (32×64), T42 (64×128), T85 (128×256), and T170 (256×512). Performance is somewhat sensitive to the number of vertical levels for small horizontal grids and for small numbers of vertical levels, but this sensitivity disappears quickly as the number of levels increases. This sensitivity is further diminished in experiments (not shown here) using the Fortran implementations of the Fourier transforms. Performance is also sensitive to the size of the horizontal grid, up to size T42. However, performance is nearly identical for the larger horizontal grids (T42, T85, and T170).

Figure 6 compares per core performance when executing independent instances of the T85 benchmark problem on 1, 2, and 4 processor cores in the

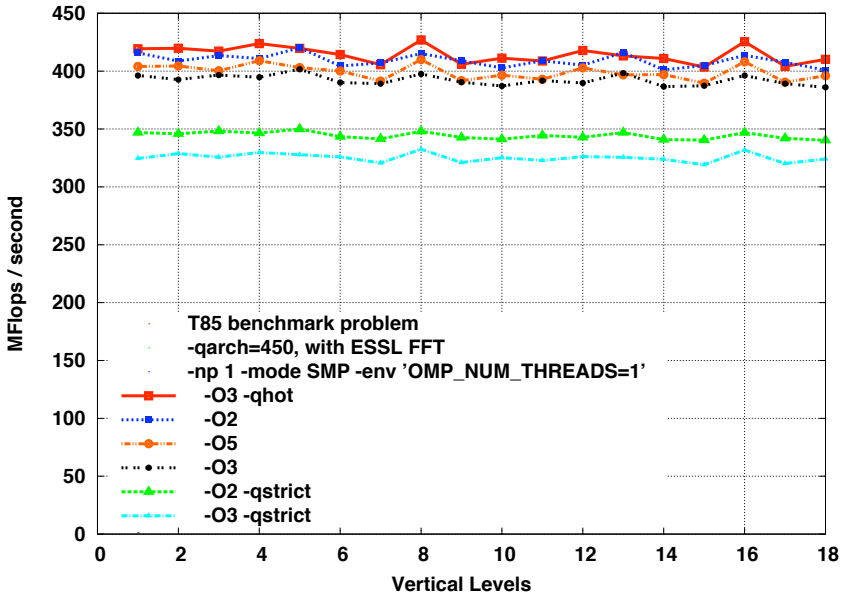

Fig. 4. PSTSWM Performance: Optimization Comparisons

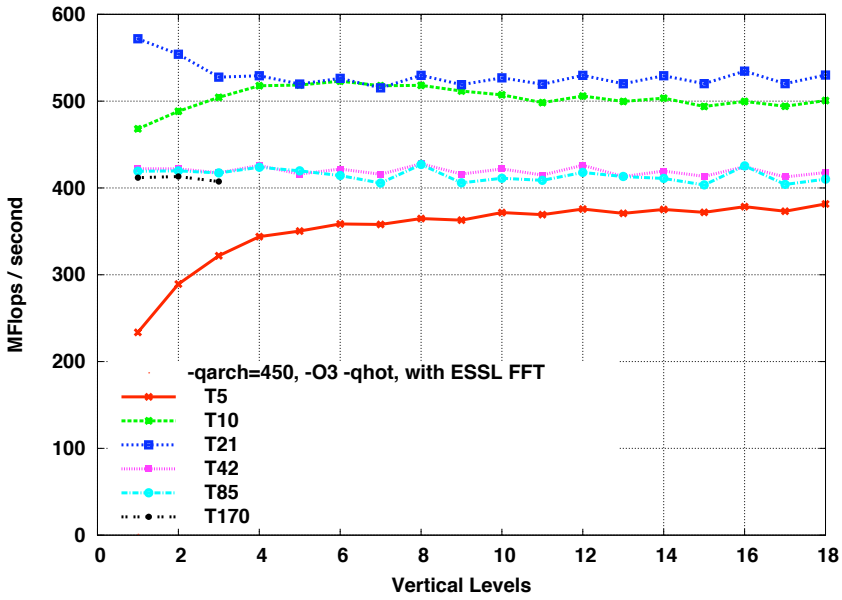

Fig. 5. PSTSWM Performance: Problem Size Analysis

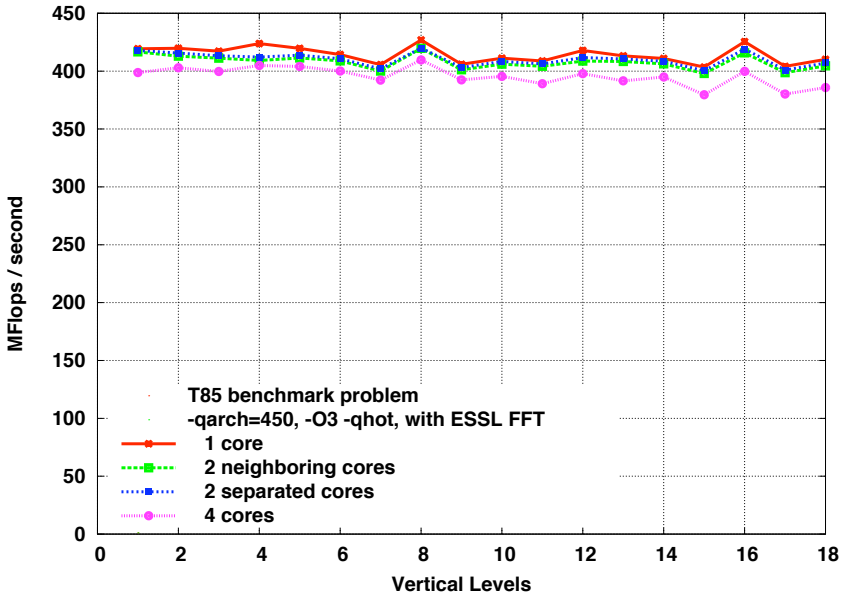

Fig. 6. PSTSWM Performance with Contention

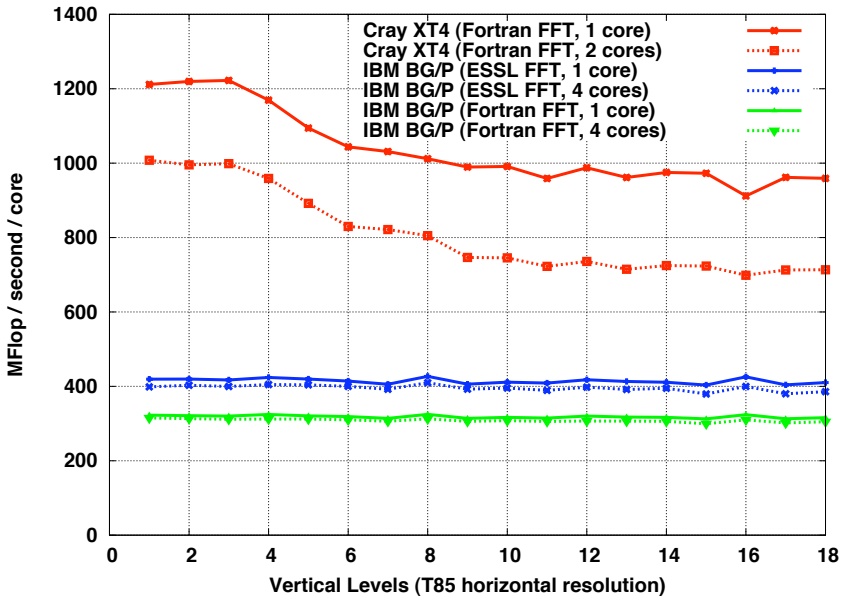

Fig. 7. PSTSWM Platform Comparison

same compute node. When running on all 4 cores, there is a small, but measurable, decrease in per core performance that grows with the number of vertical levels. This probably reflects memory contention, but is a mild performance degradation compared to most other systems with which we have experience. Finally, Figure 7 compares PSTSWM per core performance on the BG/P with that on the Cray XT4. The Cray XT4 results use only the Fortran Fourier transform implementations. Memory access patterns and contention have larger performance impacts on the XT4 than on the BG/P for this problem resolution. However, performance on the XT4 is between 4 and 3 times better than that of the BG/P for comparable experiments without memory contention and between 3 and 2.3 times better for comparable experiments with contention. The ratio of peak processor performance between the two systems is approximately 3 (when disregarding the second pipe on the BG/P floating point unit), which appears to be reflected in these results.

In summary, the PSTSWM benchmark as currently written is able to use the second floating point unit to some advantage when specifying strict numerics, but not otherwise. Versions of PSTSWM that run efficiently on vector systems such as the NEC SX-6 and the Cray X1E are not competitive with the version used in these experiments, so it is not simply an issue of vectorization. The BG/P shows little performance degradation due to memory contention or increased dependence on access to main memory. The ESSL Fourier transform routines provide a significant performance boost over the Fortran equivalents, which may partially reflect the utilization of the second floating point pipe within the math library routines.

4 Communication Benchmarks

4.1 Clock and Barrier Benchmarks

Experiments sampling `MPI_WTIME` across cores within a compute node and across the compute nodes indicate that `MPI_WTIME` is synchronized across all processor cores. The cost of reading the clock was never observed to be more than 0.24 microseconds, and was on the average 0.18 microseconds.

Figure 8 compares the performance of `MPI_BARRIER` when assigning only one MPI process to each node and when assigning 4 MPI processes to each node. It also compares performance when using the (default) optimized MPI collective and when setting `DCMF_COLLECTIVE = 0` to disable the use of the optimized collective.

We used the worst case timing over 10000 experiments for each data point in the figure. The performance of the optimized collective is consistent from experiment to experiment. For example, the minimum and maximum timings differed by less than 20% over 10000 experiments when using 8192 processes. The performance of the unoptimized collective is subject to large, but infrequent, perturbations. However, performance is consistent (20% spread between minimum and maximum timings for the the 8192 process experiment) other-

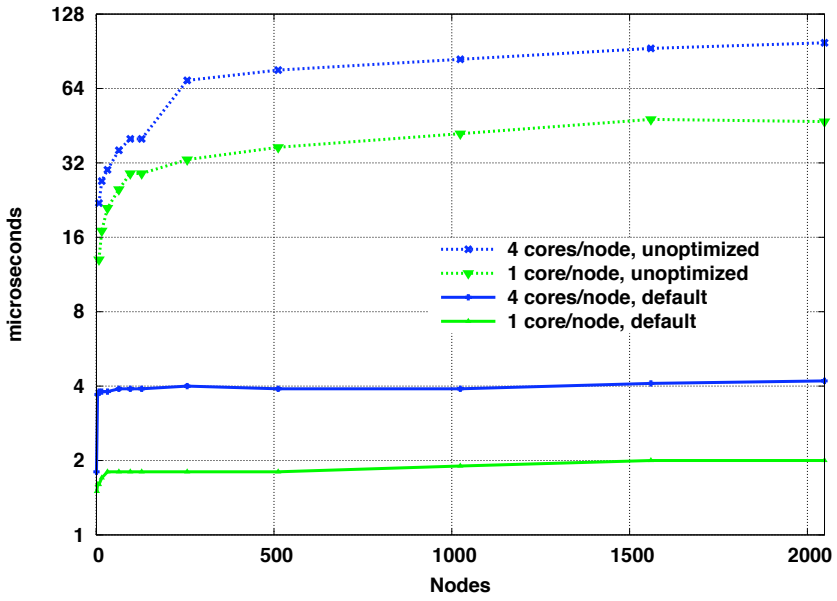

Fig. 8. MPI_Barrier Performance

wise. We removed unrepresentative outliers from the data for the unoptimized collective before identifying the "worst case" timings that are used in the figure.

As can be seen, using the optimized collectives (and the barrier network) is more than 20 times faster than not doing so for larger node counts. Also, executing a barrier using all (4) of the cores in a node is approximately twice as expensive as when using only one core per node.

4.2 MPI Benchmarks

To examine MPI communication performance, we use a custom MPI communication benchmark developed at ORNL in 1997 and used in our evaluation studies for the past 10 years. (See, for example, [3, 9–11].) The foci in these experiments are on examining performance of the many different MPI-1 point-to-point communication protocols for the exchange of data between two processes, on examining the performance impact of different types of contention, and on conducting the experiments in ways that reflect common usage. In particular, experiments are run in three modes: single exchange, a cache invalidation followed by a single exchange, and the average performance of 10 identical exchanges (without cache invalidation).

Figure 9 contains two graphs with the same data, the first using a linear y-axis and a logarithmic x-axis and the second being a log-log graph. Here we are comparing the bidirectional bandwidth observed by a pair of processes under

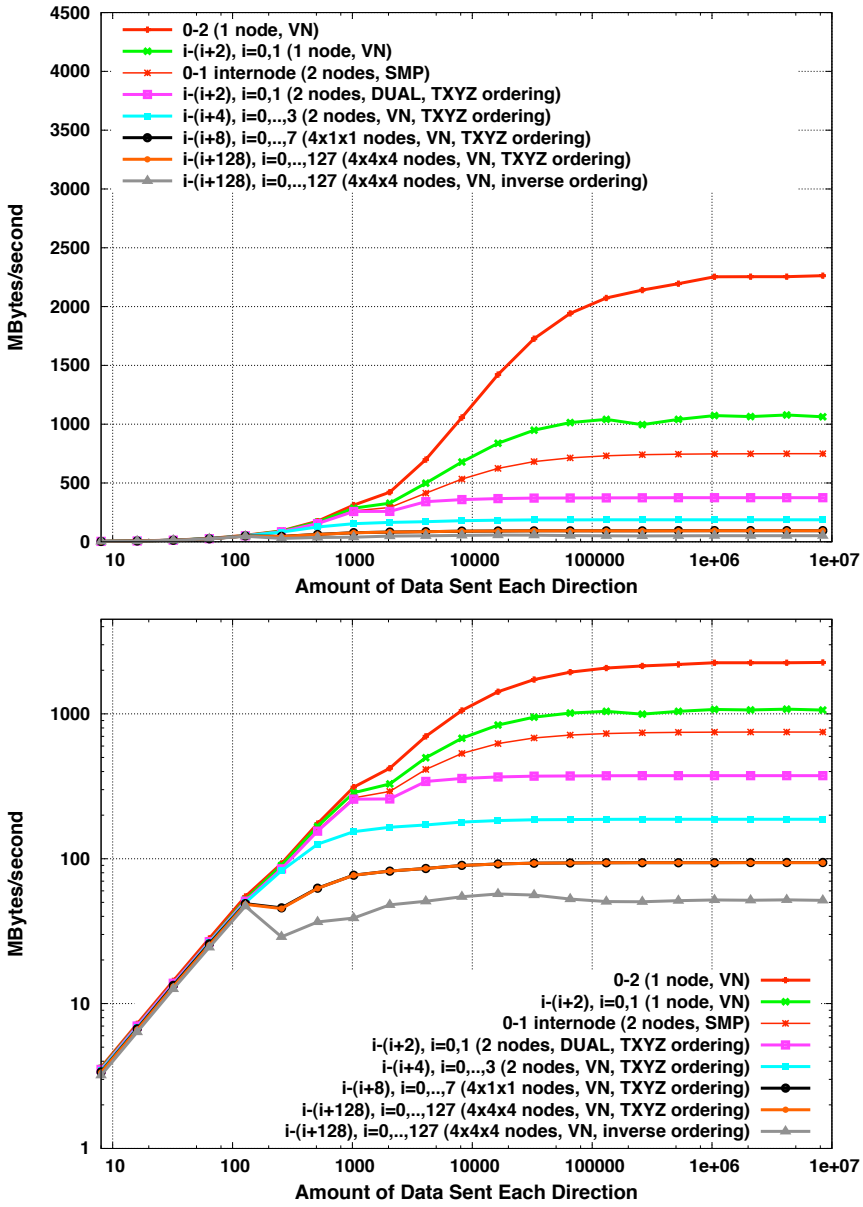

Fig. 9. MPI Bidirectional Bandwidth

a variety of circumstances. In these figures the maximum performance over all examined communication protocols for the single exchange experiment is used.

a) `0-2 (1 node, VN)` describes the bandwidth when processes assigned to core 0 and core 2 within the same compute node exchange data.
b) `i-(i+2), i=0,1 (1 node, VN)` describes the bandwidth achieved by a single pair of processes assigned to cores in the same compute node when two pairs are communicating simultaneously (process 0 with process 2 and process 1 with process 3).
c) `0-1 internode (2 nodes, SMP)` describes the bandwidth when processes assigned to core 0 in two neighboring compute nodes exchange data.
d) `i-(i+2), i=0,1 (2 node, DUAL, TXYZ ordering)` describes the bandwidth achieved by a single pair of processes assigned to cores in neighboring nodes when two pairs are communicating simultaneously.
e) `i-(i+4), i=0,3 (2 node, VN, TXYZ ordering)` describes the bandwidth achieved by a single pair of processes assigned to cores in neighboring nodes when four pairs are communicating simultaneously. In this case, every core in one node is communicating simultaneously with the analogous core in the same neighboring node.
f) `i-(i+8), i=0,7 (4x1x1 nodes, VN, TXYZ ordering)` describes the per pair bandwidth when each core in node `i`, `i=0` or `1`, is communicating with the analogous core in node `i+2` in a 4x1x1 linear array of nodes, and all four nodes are running this experiment simultaneously.
g) `i-(i+128), i=0,127 (4x4x4 nodes, VN, TXYZ ordering)` describes the per pair bandwidth when each core in node `i`, `i<32`, is communicating with the analogous core in node `i+32` in a 4x4x4 cube of nodes, and all 64 nodes are running this experiment simultaneously.
h) `i-(i+128), i=0,127 (4x4x4 nodes, VN, inverse ordering)` describes the same experiment as (g) except that the node numbering is inverted for nodes 33 to 64. This forces node 0 to communicate with the most distant node in the 4x4x4 cube, and maximizes link contention for all pairs.

From these data we see that performance for small messages is identical for all of the above scenarios. For larger messages intranode bandwith without contention is highest, followed by intranode bandwidth with contention, followed by internode bandwidth without contention, etc., as would be expected. In particular, the intranode bandwidth with contention is half that of intranode bandwith without contention, so the two pairs of processes are sharing a fixed intranode bandwidth capacity. Similarly, a single pair of processes in two different nodes achieves twice the internode bandwidth as two pairs and four times as much as four pairs, indicating equal sharing of available internode bandwidth. The 4x1x1 node experiments, where all 8 process pairs contend for a single network link, demonstrates another halving of performance per process pair, or the same total bandwidth shared between all 8 process pairs. The first 32 node experiment achieves identical performance with that of the 4x1x1 experiment, reflecting the network contention in the 4x4x4 cube with the TXYZ node ordering. Forcing

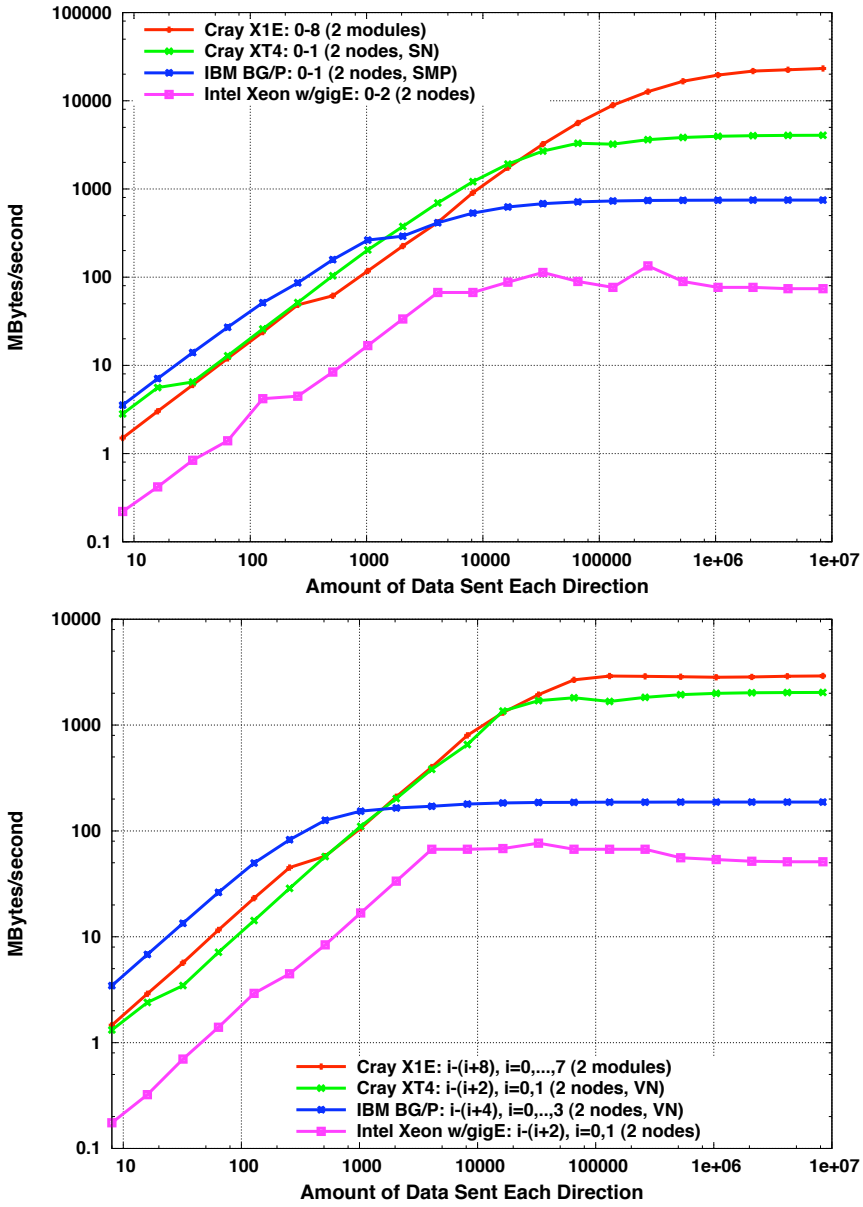

Fig. 10. MPI Bidirectional Bandwidth Platform Comparison

additional contention as in experiment (h) further halves the per process pair performance. The results described in Figure 9 are very consistent and regular, providing a simple performance model based solely on network topology and number of cores per compute node.

Figure 10 compares MPI bidirectional bandwidth performance on the BG/P with that on the Cray X1E, Cray XT4, and an Intel Xeon cluster with a Gigabit Ethernet network. In the first graph the comparison is between single process pair internode bandwidth. For small messages, up to 1024 bytes, the BG/P achieves the best MPI-1 performance, and is over twice as fast as the Cray systems for most of this range of message sizes. For large messages, the Cray X1E achieves 6 times the bidirectional bandwidth of the Cray XT4, and 31 times that of the IBM BG/P. The second graph in Figure 10 compares MPI bidirectional bandwidth under various levels of contention for the same four systems. The performance comparison for small message sizes is unchanged. For large messages, all of the systems show contention, with the Cray X1E and IBM BG/P demonstrating the most predictable behavior. The 2-node internode performance comparison is a simple function of the single pair performance and the number of processes per compute node. In this case the X1E is now only 15 times faster than the BG/P for large message sizes, while the XT4 is now 10 times faster. For the multiple node experiments, the network and process layout becomes a factor, but nothing is mysterious about these data.

In summary, BG/P MPI-1 point-to-point latency is very good, while MPI-1 bidirectional bandwidth on the torus network is significantly less than that on the Cray systems. These experiments do not, however, evaluate the ability of the BG/P (and the other systems) to avoid link contention and maintain high appregate bandwidth in real application codes at scale.

Comparing performance between the different MPI protocols, `MPI_SENDRECV` and `MPI_ISEND/MPI_RECV` protocol are typically the best performers for intranode communication for small messages and for large messages without contention. For larger message sizes, the `MPI_IRECV/MPI_SEND` and `MPI_IRECV/MPI_RSEND` protocols are fastest for intranode with contention. For internode communication, `MPI_SENDRECV` and `MPI_ISEND/MPI_RECV` are among the fastest protocols, with or without contention. The sensitivity of performance to the choice of protocol is a function of message size and contention, and can not be addressed adequately in this paper. However, the sensitivity is greatest for small messages, where the fastest protocol can be twice as fast as the slowest protocol. For the largest messages, performance for most protocols differs by less than 1%.

5 Application Benchmarks

5.1 POP

The Parallel Ocean Program (POP) [12, 13] is a global ocean circulation model developed and maintained at Los Alamos National Laboratory. It is used for high resolution studies and as the ocean component in the Community Climate

System Model (CCSM). The code is based on a finite-difference formulation of the 3D flow equations on a shifted polar grid. POP performance is characterized by the performance of a baroclinic phase and a barotropic phase. The 3D baroclinic phase typically scales well on all platforms due to its limited nearest-neighbor communication. In contrast, the barotropic phase is dominated by the solution of a 2D, implicit system whose performance is sensitive to network latency and typically scales poorly on all platforms. For our evaluation we used version 1.4.3 of POP with a few additional parallel algorithm tuning options (due to Yoshida) [14]. The current production version of POP is version 2.0.1. While version 1.4.3 and version 2.0.1 have similar performance characteristics, the intent here is to use version 1.4.3 to evaluate system performance characteristics, not to evaluate the performance of POP.

We consider results for both one degree and tenth degree fixed size benchmark problems. Both benchmark problems use a displaced-pole longitude-latitude horizontal grid with the pole of the grid shifted into Greenland to avoid computations near the singular points. For the one degree problem the grid spacing is approximately one degree in longitude (100km) around the equator, utilizing a 320×384 horizontal grid and 40 vertical levels. This benchmark problem is configured as closely as possible to the way that POP is used for long climate simulations in the CCSM. For the tenth degree problem the grid spacing is 0.1 degree in longitude (10km) around the equator, utilizing a 3600×2400 horizontal grid and 40 vertical levels. This resolution resolves eddies for effective heat transport and is used for ocean-only or ocean and sea ice experiments.

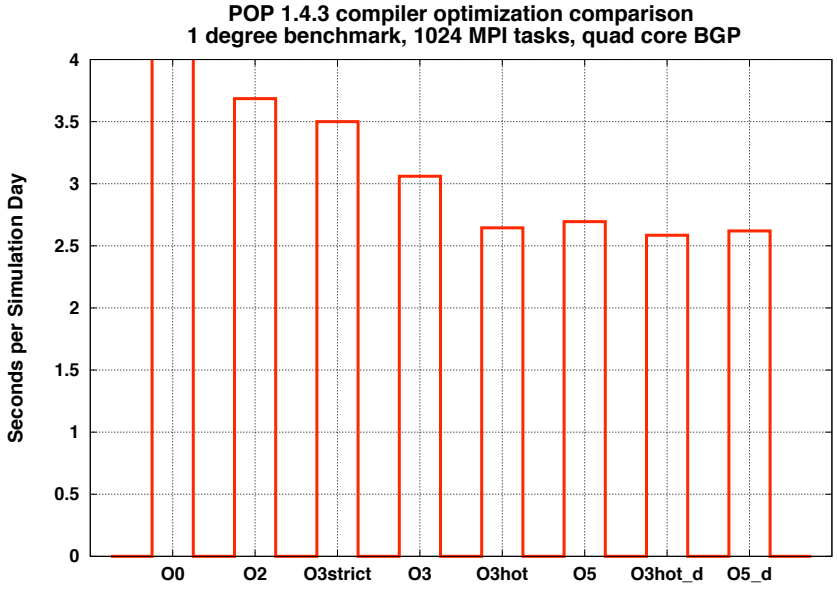

Fig. 11. POP Performance: Compiler Optimization Comparisons

We began by evaluating the impact of different compiler optimizations. Figure 11 compares seconds per simulation day for the one degree benchmark when using 1024 processes for the following compiler options: `-O0`, `-O2`, `-O3 -qstrict`, `-O3`, `-O3 -qhot`, `-O5`, `-O3 -qhot -qarch=450d`, and `-O5 -qarch=450d`. Other optimization flags were also examined, but these capture the major issues in the comparison. The `-O0` value is over 10 seconds. As can be seen, `-O3 -qhot` and `-O3 -qhot -qarch=450d` achieve the best performance. Experiments using executables compiled with `-qarch=450d` led to execution time errors for certain benchmark configurations. Consequently, we tried executables compiled with both `-O3 -qhot` and `-O3 -qhot -qarch=450d` in all experiments and report the best observed performance. Qualitatively, results were identical for a similar comparison of compiler optimizations using the tenth degree benchmark and 4000 processes.

Figures 12 and 13 describe POP performance for the one degree benchmark problem. Executables compiled with `-qarch=450d` were (slightly) faster than executables compiled with `-qarch=450d` up to 1024 processes, but not for larger process counts. Performance when using the `TXYZ` ordering was comparable to that when using the `XYZT` ordering.

The first graph in Figure 12 compares POP performance in simulated years per day when using only one core per compute node (SMP mode) and when using all cores per node (VN mode). It also compares using a standard conjugate gradient algorithm solver for the linear system in the Barotropic phase with using the Chronopoulos-Gear variant (C-G) [15]. C-G halves the number of calls to `MPI_Allreduce` used to compute inner products at the cost of some additional computation. On a per process basis, SMP-mode performance is slightly better than VN mode, though not on a per node basis. The C-G variant of the linear solver improves performance by approximately 5% when running in VN mode. The second graph in Figure 12 compares the performance of the Barotropic and Baroclinic phases in seconds per simulated day for the SMP and VN modes, both using the C-G-based solver. Here we can see that SMP and VN modes achieve the same performance for the Baroclinic phase, while the communication-bound Barotropic phase is scaling much better when run in SMP mode. It appears that the halo updates used in the residual calculations and the reduction operations used in the inner product calculations are not as efficient when using all cores in a compute node. However, because the cost of the Baroclinic phase dominates, this performance degradation in the Barotropic phase in VN mode does not have a significant impact on VN-mode performance.

The first graph in Figure 13 compares performance on the quad-core BG/P system with performance on the dual-core XT4 system. The XT4 shows a much stronger preference for running on a single core per compute node (SN mode), achieving higher maximum throughput than when running on both cores (VN mode), with SN-mode Barotropic performance twice that of VN mode for the same process count. Performance on the XT4 peaks at between 2000 and 3000 processes, while performance on the BG/P continues to improve out to at least 8192 processes. Achieved performance on the XT4 is still higher up to 8192

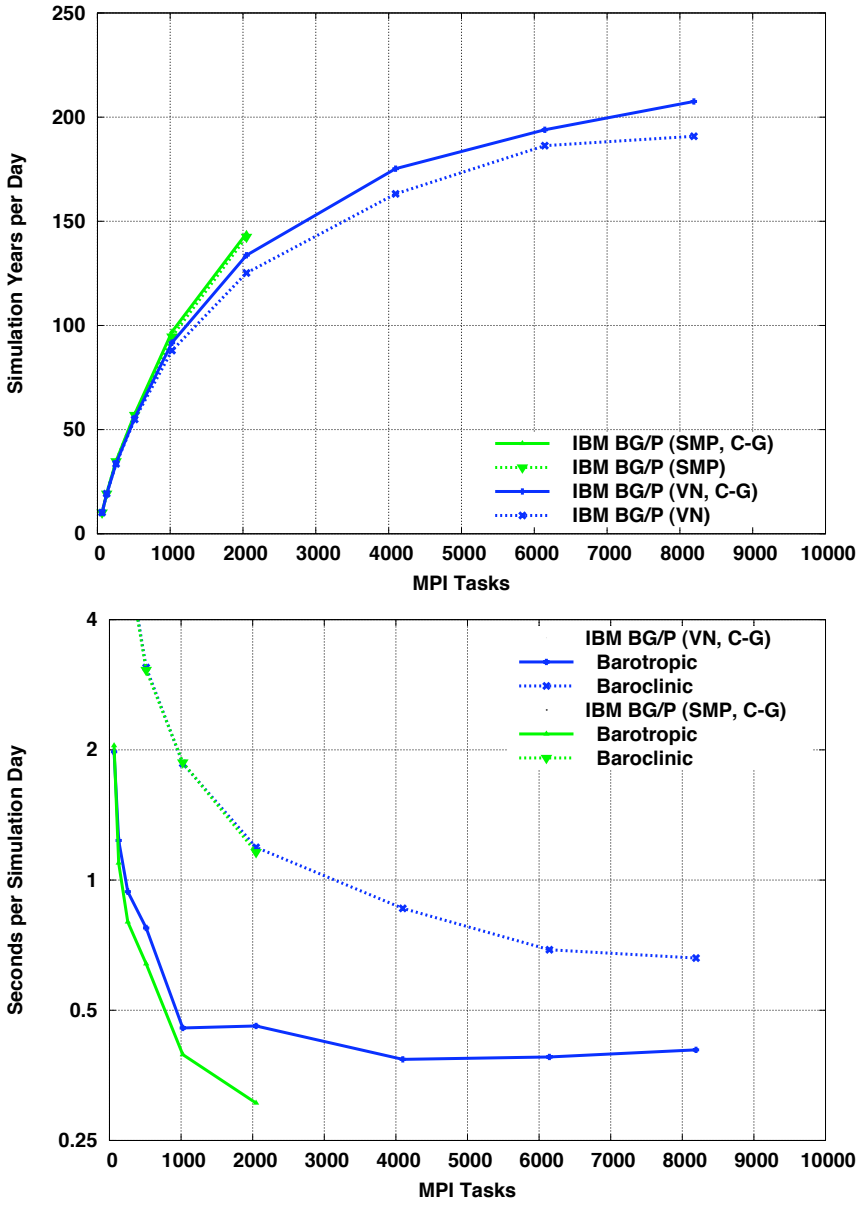

Fig. 12. POP Performance: One-Degree Benchmark

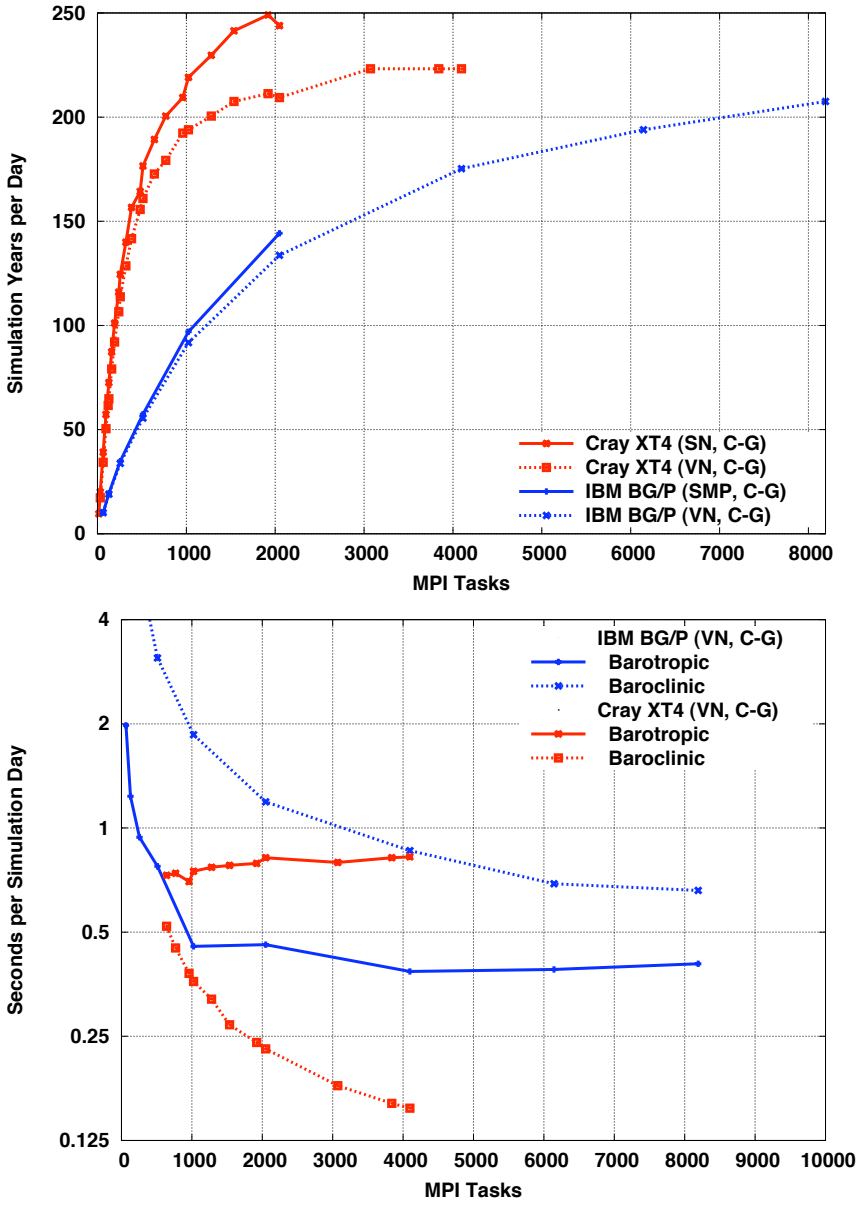

Fig. 13. POP Platform Comparison: One-Degree Benchmark

processes. The second graph in Figure 13 compares BG/P and XT4 performance for the Barotropic and Baroclinic phases when using all cores per compute node. Differences in performance characteristics in the two systems are clear in this figure. The Baroclinic performance is much better on the XT4 than on the BG/P, and it even appears to scale better. In contrast, Barotropic performance is much better on the BG/P than on the XT4, though performance is essentially flat on both platforms for more than 1024 processes. While there is potential for continued scaling on the BG/P for larger process counts, Baroclinic performance improved by only 30% when doubling from 4096 processes to 8192 processes, so we do not expect performance to increase significantly when using more than 8192 processes.

Figures 14 and 15 describe the POP performance for the tenth degree benchmark problem. Executables compiled with `-qarch=450d` were 10% faster than executables compiled with `-qarch=450` when run with 1280 processes, but the performance advantage decreased to less than 2% for runs with 8000 processes. As with the one degree benchmark, performance when using the `TXYZ` ordering was comparable to that when using the `XYZT` ordering. Experiments with other orderings likewise did not show any significant performance advantage over `TXYZ`.

The first graph in Figure 14 compares POP performance for VN and SMP modes, and with and without the C-G variant of the linear solver. For this large problem, scaling is linear out to 8000 processes and performance is insensitive to the execution modes and to the linear system solver variant. The second graph in Figure 14 compares the performance of the Barotropic and Baroclinic phases in seconds per simulated day for the SMP and VN modes, both using the C-G-based solver. The Baroclinic timings are for process 0 only. While this was sufficient for the one degree problem, there is some load imbalance in the Baroclinic phase that would normally be mistakenly attributed to the Barotropic phase (whose timings are also reported only for process 0). To disambiguate the timings, the experiments were rerun with a barrier placed just before the start of the Barotropic phase. The process-0 time spent in the barrier is also plotted. (This additional barrier decreases overall POP performance very little.) As with the one degree benchmark results, the Baroclininc performance is not sensitive to execution mode. The C-G solver variant is a little slower than the standard formulation of the conjugate gradient solver for this problem size and these process counts, but the Baroclinic phase is the dominant contributer to total execution time and the performance difference between the two solver algorithms has little impact. In particular, the Baroclinic load imbalance, as measured by the process-0 timing barrier, is as large as the total cost of the Barotropic phase for 8000 processes.

The first graph in Figure 15 compares performance on the quad-core BG/P system with performance on the dual-core XT4 system for the tenth degree benchmark. The XT4 again shows more sensitivity to the execution mode than the BG/P, but using all of the cores in a compute node is now preferable to using only one. (Performance for 20000 processes, not plotted, is 19.6 years per day, while SN mode on the same number of compute nodes achieves 15.8 years

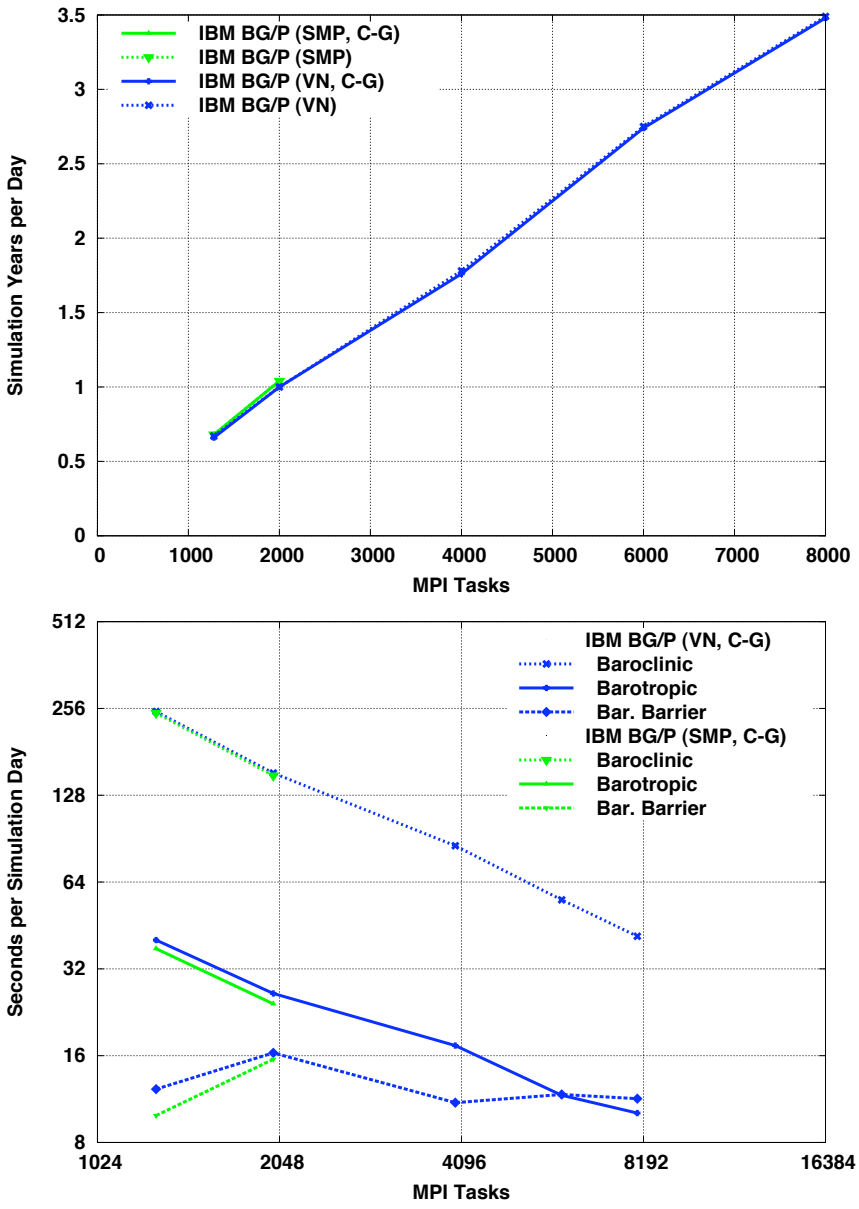

Fig. 14. POP Performance: Tenth-Degree Benchmark

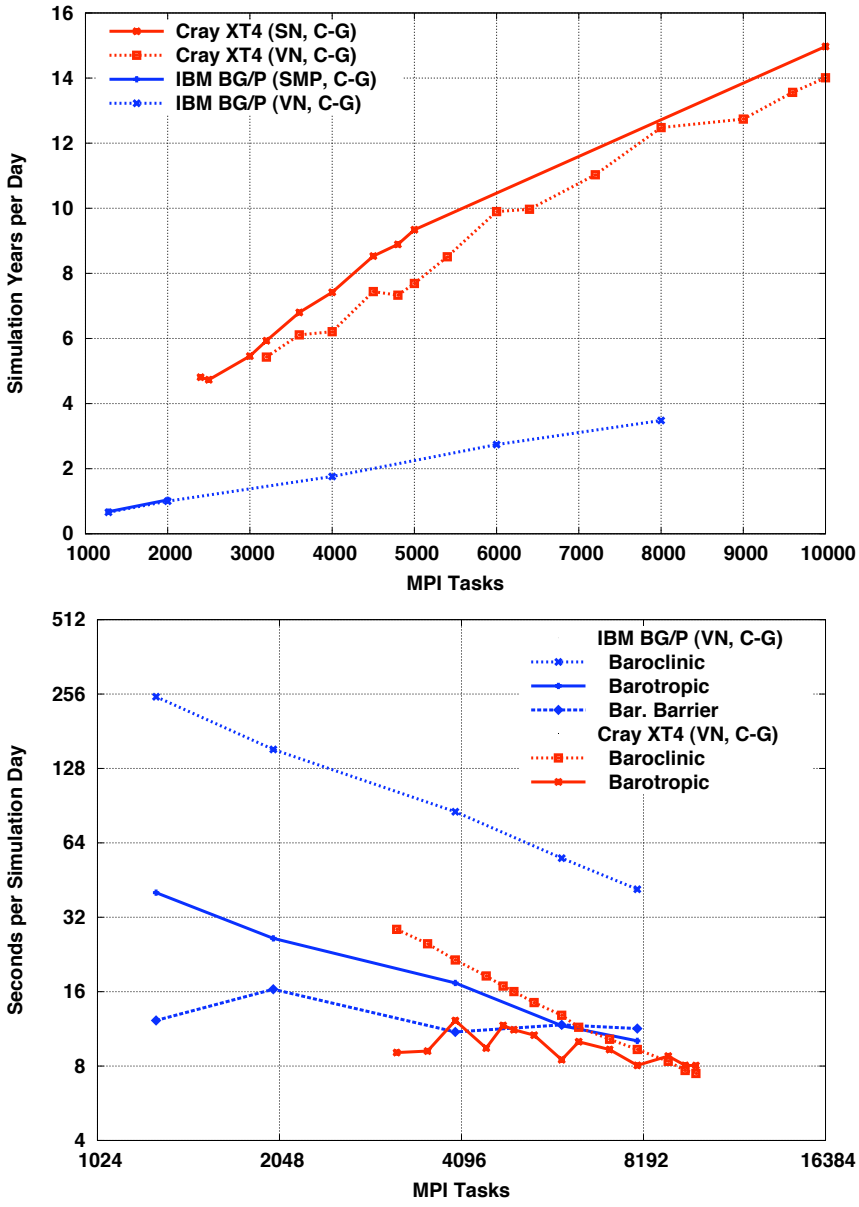

Fig. 15. POP Platform Comparison: Tenth-Degree Benchmark

per day.) The XT4 performance is approximately 3.6 times that of the BG/P for 8000 processes. The second graph in Figure 15 compares BG/P and XT4 performance for the Barotropic and Baroclinic phases. In this figure, a timing barrier was not used to remove load imbalances from the XT4 Barotropic phase timings. For example, load imbalances may be the source of the somewhat erratic behavior in the Barotropic phase performance on the XT4. As with the one degree benchmark, the Baroclinic phase runs much faster on the XT4 than on the BG/P, and it appears to scale better on the XT4 as well. Performance of the Barotropic phase on the BG/P continues to increase out to 8000 processes, though it is still somewhat slower than on the XT4. XT4 Barotropic performance has stopped improving beyond 8000 processes, and is the dominant phase when using more than 10000 processes. It appears that performance would continue to scale out to much larger process counts on the BG/P, especially since the load imbalance in the Baroclinic phase should decrease as the cost of the Baroclinic phase decreases.

In summary, the POP benchmarks scale well on the BG/P architecture. Performance is less than on the XT4 when running on the same number of processes, particular for the large computation-bound benchmark. When communication cost dominates on the XT4, performance on the BG/P becomes competitive for large process counts.

6 Conclusions

These performance results are preliminary, and provide at best a snapshot of the performance characteristics of the IBM BG/P architecture. However, our preliminary evaluation indicates that the multiple networks and the compute node design and software stack do enhance application scalability, much like on the BG/L system [16]. The processor is relatively slow compared to HPC systems from other vendors, which puts a premium on the scalability of the application codes. The relatively low bandwidth of the torus network also makes it important to consider the assignment of processes to processors (and the exploitation of OpenMP parallelism within the compute node) in order to minimize the communication requirements for applications with high bandwidth requirements. The larger ORNL evaluation effort, and similar activities at Argonne National Laboratory and other sites with BG/P systems, are investigating system performance with additional microbenchmarks and with a large number of application codes. This additional information will be available soon and should provide a more comprehensive evaluation of BG/P performance.

7 Acknowledgements

This research used resources (Xeon cluster) of Oak Ridge National Laboratory and resources (Cray X1E, Cray XT4, and IBM BG/P) of the National Center

for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. It also used resources (IBM p575 cluster) of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. We thank our performance research colleagues at Oak Ridge National Laboratory for support in collecting these data, in particular Michael Bast for his efforts in getting the IBM BG/P system installed and running.

References

1. Gropp, W., Snir, M., Nitzberg, B., Lusk, E.: MPI: The Complete Reference. MIT Press, Boston (1998) second edition.
2. Dagum, L., Menon, R.: OpenMP: an industry-standard API for shared-memory programming. IEEE Computational Science & Engineering **5** (1998) 46–55
3. Alam, S.R., Barrett, R.F., Fahey, M.R., Kuehn, J.A., Larkin, J.M., Sankaran, R., Worley, P.H.: Cray XT4: An Early Evaluation for Petascale Scientific Simulation. In: Proceedings of the ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC07), Nov. 10-16, 2007. IEEE Computer Society Press, Los Alamitos, CA (2007)
4. Worley, P.H.: Comparison of the Cray XT3 and XT4 Scalability. In R. Winget and K. Winget, ed.: Proceedings of the 49th Cray User Group Conference, May 7-10, 2007, Eagan, MN, Cray User Group, Inc. (2007)
5. Oliker, L., Carter, J., Wehner, M., Canning, A., Ethier, S., Mirin, A., Bala, G., Parks, D., Worley, P., Kitawaki, S., Tsuda, Y.: Leading Computational Methods on Scalar and Vector HEC Platforms. In: Proceedings of the ACM/IEEE Intl. Conf. for High Performance Computing, Networking and Storage (SC05), Nov. 12-18, 2005. IEEE Computer Society Press, Los Alamitos, CA (2005)
6. Dongarra, J., Croz, J.D., Duff, I., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Software **16** (1990) 1–17
7. Worley, P.H., Foster, I.T.: Parallel Spectral Transform Shallow Water Model: a runtime–tunable parallel benchmark code. In Dongarra, J.J., Walker, D.W., eds.: Proc. Scalable High Performance Computing Conf. IEEE Computer Society Press, Los Alamitos, CA (1994) 207–214
8. Worley, P.H., Toonen, B.: A users' guide to PSTSWM. Technical Report ORNL/TM–12779, Oak Ridge National Laboratory, Oak Ridge, TN (1995)
9. Drake, J.B., Hammond, S., James, R., Worley, P.H.: Performance tuning and evaluation of a parallel community climate model. In: Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC99), Nov. 13-19, 1999. IEEE Computer Society Press, Los Alamitos, CA (1999)
10. Dunigan, Jr., T.H., Fahey, M.R., White III, J.B., Worley, P.H.: Early Evaluation of the Cray X1. In: Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC03), Nov. 15-21, 2003. IEEE Computer Society Press, Los Alamitos, CA (2003)
11. Worley, P.H., Dunigan, Jr., T.H., Fahey, M.R., White III, J.B., Bland, A.S.: Early evaluation of the IBM p690. In: Proceedings of the IEEE/ACM SC2002 Conference, Nov. 16-22, 2002. IEEE Computer Society Press, Los Alamitos, CA (2002)
12. Smith, R.D., Dukowicz, J.K., Malone, R.C.: Parallel ocean general circulation modeling. Phys. D **60** (1992) 38–61

13. Jones, P.W., Worley, P.H., Yoshida, Y., White III, J.B., Levesque, J.: Practical performance portability in the Parallel Ocean Program (POP). Concurrency and Computation: Practice and Experience **17** (2005) 1317–1327
14. Worley, P.H., Levesque, J.: The performance evolution of the Parallel Ocean Program on the Cray X1. In R. Winget and K. Winget, ed.: Proceedings of the 46th Cray User Group Conference, May 17-21, 2004, Eagan, MN, Cray User Group, Inc. (2004)
15. Chronopoulos, A., Gear, C.: s-step iterative methods for symmetric linear systems. J. Comput. Appl. Math. **25** (1989) 153–168
16. de Supinski, B.R., Schulz, M., Bulatov, V.V., Cabot, W., Chan, B., Cook, A.W., Draeger, E.W., Glosli, J.N., Greenough, J.A., Henderson, K., Kubota, A., Louis, S., Miller, B.J., Patel, M.V., Spelce, T.E., Streitz, F.H., Williams, P.L., Yates, R.K., Yoo, A., Almasi, G., Bhanot, G., Gara, A., Gunnels, J.A., Gupta, M., Moreira, J., Sexton, J., Walkup, B., Archer, C., Gygi, F., Germann, T.C., Kadau, K., Lomdahl, P.S., Rendleman, C., Welcome, M.L., McLendon, W., Hendrickson, B., Franchetti, F., Kral, S., Lorenz, J., Uberhuber, C.W., Chow, E., Catalyurek, U.: Bluegene/l applications: Parallelism on a massive scale. International Journal of High Performance Computing Applications **22** (2008) 33–51