Preprint
UCRL-JC-142563

The ASCI PSE Milepost: Run-Time Systems Peformance Tests

B. R. de Supinski

This article was submitted to
The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications
Las Vegas, NV
June 25-28, 2001

May 7, 2001

The ASCI PSE Milepost: Run-Time Systems Performance Tests

Bronis R. de Supinski
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
bronis@llnl.gov

Abstract: The Accelerated Strategic Computing Initiative (ASCI) Problem Solving Environment (PSE) consists of the tools and libraries needed for the development of ASCI simulation codes on ASCI machines. The recently completed ASCI PSE Milepost demonstrated that this software environment is available and functional at the scale used for application mileposts on ASCI White. As part of the PSE Milepost, we performed extensive performance testing of several critical run-time based systems. In this paper, we present microbenchmark results that compare the MPI [5], Pthreads and OpenMP [7, 8] implementations on ASCI White and ASCI Blue Pacific. Our results demonstrate that these run-time systems on White have improved sufficiently to accommodate the machine's approximately four-fold increase in processing capability over Blue Pacific.

1. Introduction

The Accelerated Strategic Computing Initiative (ASCI) Problem Solving Environment (PSE) consists of the tools and libraries needed for the development of ASCI simulation codes on ASCI machines. The recently completed ASCI PSE Milepost demonstrated that this software environment is available and functional at the scale used for application mileposts on ASCI White. This demonstration allows the ASCI code developers to focus on application development. As part of the PSE Milepost, we performed extensive performance testing of several critical run-time based systems. In this paper, we present a subset of these results that compare the MPI [5], Pthreads and OpenMP [7, 8] implementations on ASCI White and ASCI Blue Pacific; for the full report, see http://www.llnl.gov/CASC/RTS_Report/overall.html. The tests include some of the largest scale MPI testing ever conducted - up to 1536 tasks were used in the collective communication tests. Our results demonstrate that these run-time systems on White have improved sufficiently to accommodate the machine's approximately four-fold increase in processing capability over Blue Pacific.

2. Systems Tested

We present results for four different IBM SP systems. The Combined Technology Refresh (CTR) machine, which we also refer to as Blue Pacific, has 336 four-way 332 MHz PowerPC 604e SMP nodes. The other three systems all have Power3 (i.e. PowerPC 630) CPUs. Snow has 16 NightHawk I nodes, so each node is an eight-way SMP with a 222 MHz CPU clock rate. ASCI White has 512 NIghtHawk II nodes, so each node is a sixteen-way SMP with a 375 MHz clock rate. However, at the time of our testing, White was separated into two machines: Frost, a 376 node machine, and White, a 136 node machine. The primary performance differences between the machines arose from the number of GPFS nodes on each machine. Since we do not present results for the Milepost file system testing, our results for the two machines were essentially identical.

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48, UCRL-JC-142563.

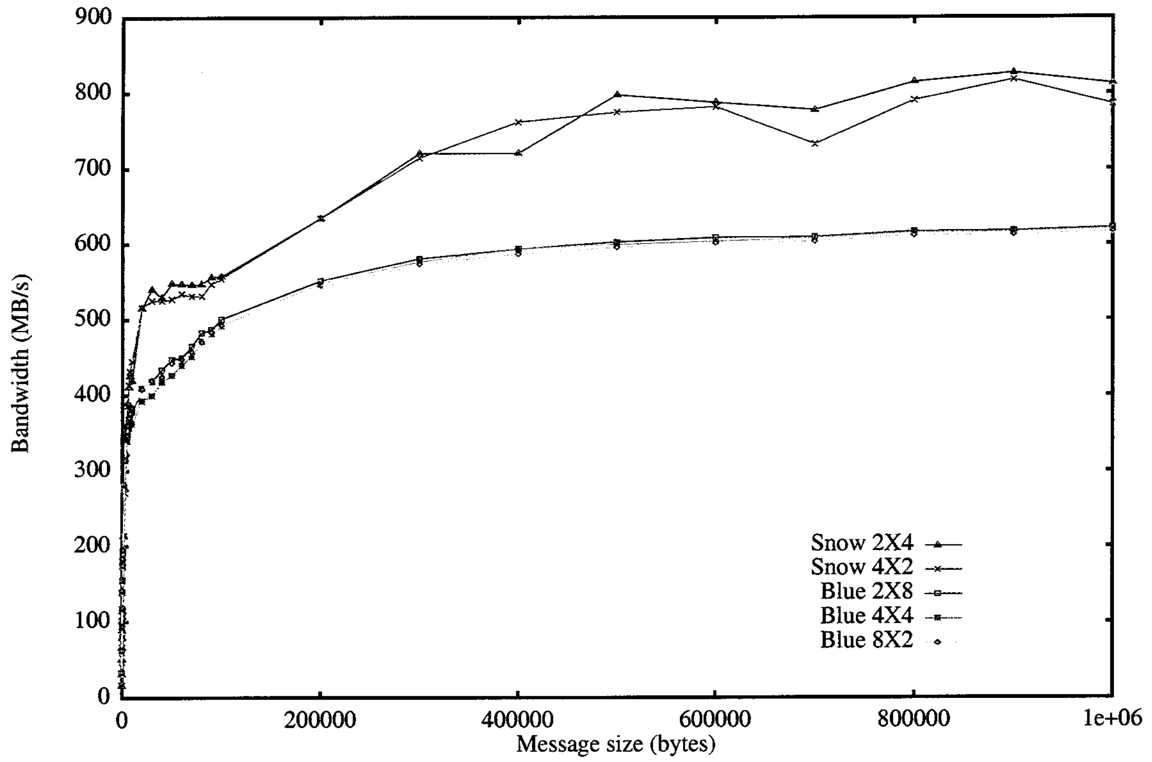

Figure 1: Stencil Bandwidths with One Task Per Node

3. Stencil Communication Pattern Testing Using NEWS05

NEWS05 measures the ability of ASCI systems to support an important message passing communication pattern, the nearest neighbor communication pattern used by two-dimensional stencil-based computations. The code's name is derived from the common name for this communication pattern, NEWS, for the North-East-West-South directions used to determine communication endpoints. The code tests all possible two dimensional grid layouts for the total number of MPI tasks in the job. The original code, a Fortran code developed by Rex Evans, exchanges messages of a size determined at compile time. All of our runs used a modified version, developed by Chris Chambreau and Bronis R. de Supinski as part of the PSE ASCI Simulation Development Environment (ASDE) project, that tests multiple message sizes during a single run. This version is available along with the full report.

Comparing results of the stencil bandwidths measured by NEWS05 on Snow to those on Blue Pacific demonstrates that improved network hardware will provide the required stencil bandwidth for our applications. For example, threaded codes that use 64 tasks will benefit from an approximately 25% increase in stencil bandwidth, as shown in Figure 1, which presents NEWS05 results using a single task per node on 8 nodes of Snow and 16 nodes of Blue (64 CPUs in both cases). These results compare the stencil bandwidths available to a fully threaded code using 64 CPUs on either machine. We note that the Snow CPUs, despite the slower clock rate, actually have greater processing power since they can complete four floating point operations per cycle, while the Blue CPUs can only complete two FLOPs per cycle. Thus, the higher bandwidth, combined with the lower surface to volume ratio anticipated with using 8 threads instead of 4, should be sufficient to compensate for the higher FLOP rates of Snow nodes.

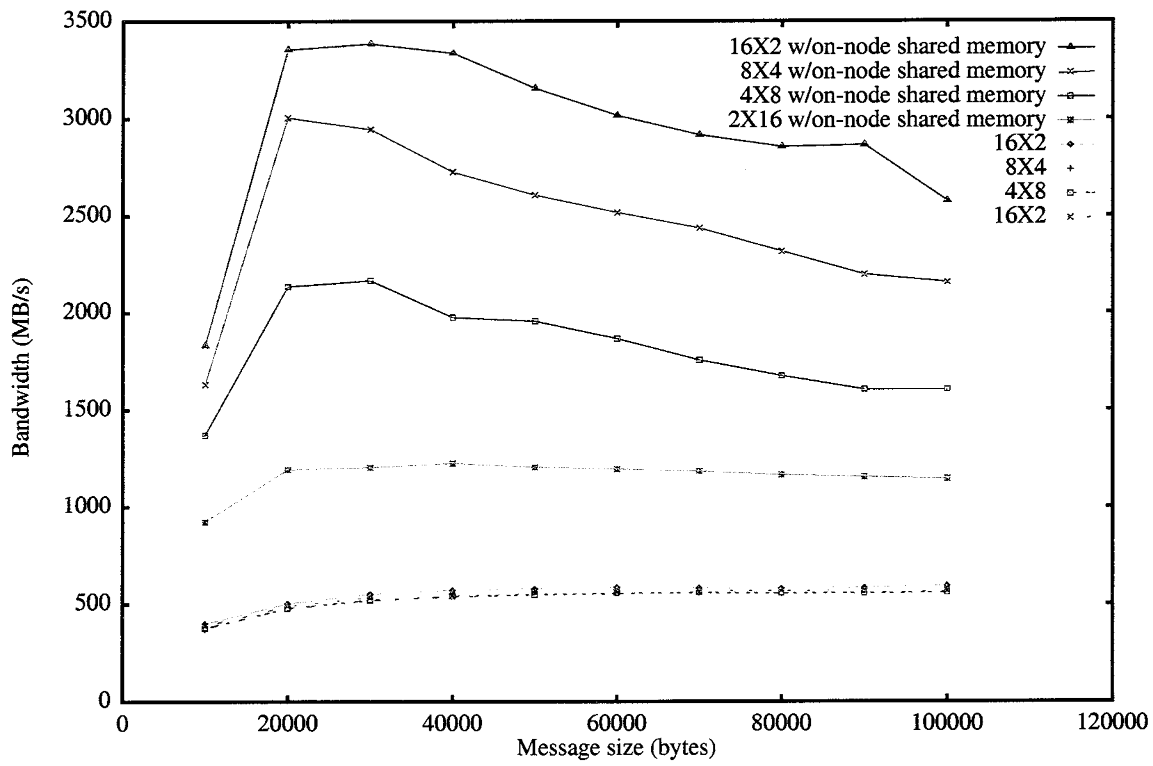

Figure 2: Stencil Bandwidths with 16 Tasks Per Node

The general result of the NEWS05 runs across message sizes is that the stencil bandwidths increase rapidly with message size, quickly approaching the asymptotic aggregate bandwidth limit. The primary users of the NEWS05 benchmark indicated that medium message sizes are most relevant to their application. Also, single task per node results for message sizes up to 100,000 bytes indicate little additional bandwidth is achieved with larger messages. Therefore, we focused the remainder of our tests on medium message sizes. Further, as demonstrated by our results in Figure 2 for 16 tasks per node on 2 nodes of Frost, using shared memory for on-node communication with multiple tasks per node provides significantly increased aggregate stencil bandwidths so we only ran scalability tests with this option.

NEWS05 was designed to guide grid layout. The results consistently indicate that the actual stencil pattern has little effect on the observed stencil bandwidth when on-node shared memory is not used for MPI communications. However, not surprisingly, when on-node shared memory communication is used with tests using multiple nodes, higher aggregate stencil bandwidths are observed with grid layouts that result in shared memory being used for more communication. In fact, the N/2 rows by 2 columns tests consistently produce the best results.

Scaling runs of NEWS05 demonstrate that aggregate stencil bandwidths in White scale well. Table 1 compares stencil bandwidths using the grid layout that minimizes off-node communication for 80000 byte messages on one node, two nodes and 96 nodes of White using on-node shared memory for MPI communications.

These results clearly demonstrate that point-to-point MPI messaging performance scales to running MPI-everywhere on 96 nodes. In fact, the greatest per-task stencil bandwidth is observed with 16 tasks per node and 96 nodes, which outperforms even 15 tasks on a single node. Although this anomaly is due in part to normal statistical variation in

Table 1: Scaling of Stencil Bandwidths

Tasks Per Node	15			16		
Nodes	1	2	96	1	2	96
Aggregate Bandwidth (MB/s)	1,728	2,525	139,819	1,702	2,849	180,198
Per-Task Bandwidth (MB/s)	115.2	84.17	97.10	106.4	89.03	117.3

network measurements, we primarily attribute the general improvement in per task bandwidth when the number of nodes is increased to the increased parallelism in the network and switch adapters.

4. OpenMP Testing

The LLNL OpenMP Performance Suite is a set of OpenMP performance tests implemented in both Fortran and C. The codes were developed by Bor Chan as part of the PSE/ASDE project. This suite measures the cost of using individual OpenMP directives and of the auxiliary OpenMP locking calls. Like previous OpenMP benchmarks [1], our suite compares the time to perform some work using the directive being measured to the time of a reference measurement that captures the cost of performing similar work without using the directive. In our tests, the work is simply a spin wait with a duration parameter. The reported result is the difference of the two timings, which captures the directive's overhead. The choice of reference measurement is critical to the accuracy of the benchmark. Many of our suite's benchmarks provide improved accuracy since the reference runs include the cost an OpenMP parallel construct, while previous benchmarks used the cost of performing the work in a serial region.

In the interest of space, we summarize the results of our OpenMP tests. The results for the full range of tests, including actual output records, are available on the Web. As part of the PSE milepost, we obtained results on both Blue and Snow using three different compilers: IBM's xlc and xlf and KAI's guidec. Each executable was compiled on the target host using optimization level -O3 -qtune=auto -qarch=auto as well as any flags required to activate the OpenMP directives. The most important of our results indicate some scaling problems with our OpenMP implementations. In particular, the performance of the parallel for (DO in Fortran) construct suffers significantly with the dynamic and guided scheduling options when the number of threads is increased from four to eight. Some of the tests of OpenMP synchronization constructs indicate a similar performance drop at eight threads. We believe these problems may be related to Pthreads performance problems discussed in the next section. In any event, none of the compilers emerges as the clear best choice based on our results. For some OpenMP constructs, such as the parallel for/DO construct with guided scheduling, the IBM compilers are clearly superior, while the KAI compiler offers significantly better performance for others, such as the barrier construct.

5. Sphinx Testing

Sphinx is an integrated parallel microbenchmark suite. It was adapted from the Special Karlsruhe MPI (SKaMPI) Benchmark suite [9] by Bronis R. de Supinski and other members of the PSE/ASDE project, including John May and Bor Chan. LLNL adaptations include extensive tests of the Pthreads interface [4] and an on-going integration of the LLNL OpenMP Performance Suite. In addition, several new MPI tests have been added primarily focusing on the performance of collective operations,

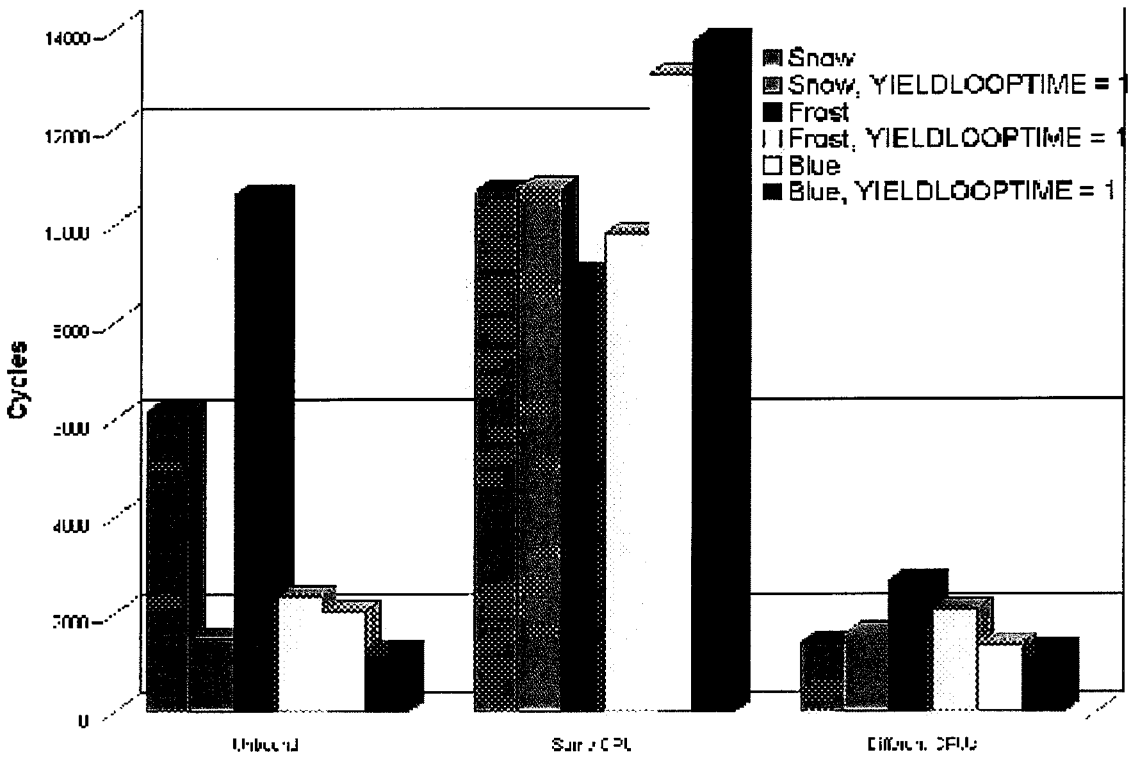

Figure 3: Performance of Pthread Mutex Pingpong

including the first widely available tests that accurately measure the operation latency of fan-out collective operations such as MPI_Bcast [3]. The entire suite is implemented in C and has been run on a wide variety of platforms. Our Milepost Sphinx testing has revealed some significant performance problems with IBM's Pthreads implementation and also led to the implementation of an accurate and scalable test of the performance of MPI_Scan.

Sphinx Pthreads Results

We ran the full set of Pthreads tests included in Sphinx on Blue, Snow and Frost. We normalized our results to account for the different clock rates of the CPUs on the machines. This normalization is reasonable since we expect that the Pthreads routines to offer little opportunity to exploit the superscalar features of the PowerPC architecture. Our results generally indicate that IBM has improved their Pthreads implementation significantly. For example, we have observed an approximately 60% reduction in the number of cycles required to create a thread. However, we observed a substantial increase in the cost of exchanging information between threads, as measured by our condition pingpong and mutex pingpong tests [4], either through condition variables or mutexes when the threads are not bound to specific CPUs by the user. Further testing revealed that setting the YIELDLOOPTIME environment variable to any value eliminates the problems for mutexes, as shown in Figure 3. We are working with IBM in identifying the reason for this anomaly, and continuing to investigate the results for condition variables.

Sphinx MPI Results

We have run the full range of MPI performance tests included in Sphinx on Blue, Snow, Frost and White using both IBM's MPI implementation and MPICH implementation that uses the MPL device. Our results for the point-to-point tests, which consist primarily of pingpong measurements

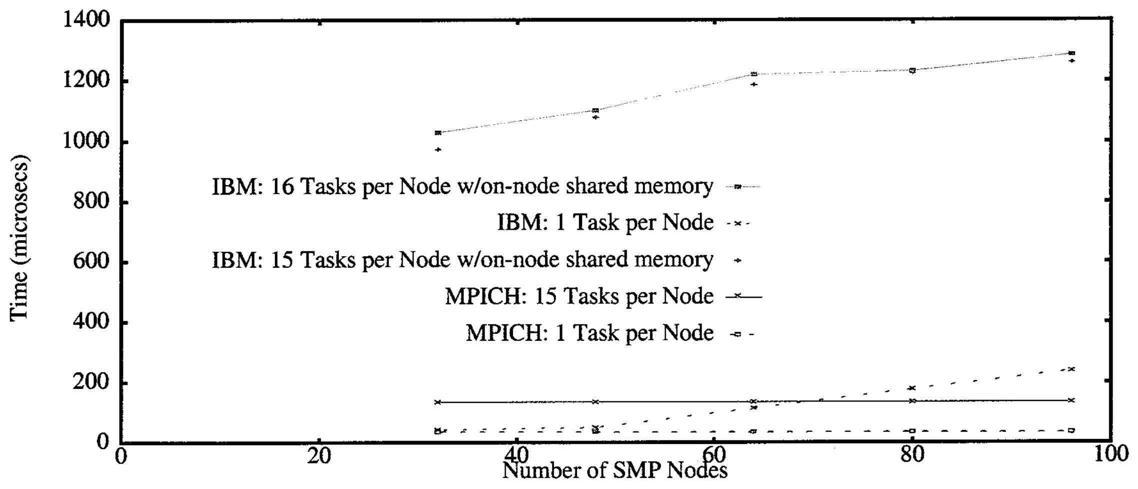

Figure 4: Overhead for MPI_Scan of 256 Bytes

for various combination of MPI send and receive operations, are consistent with the results of our NEWS05 testing. In particular, using on-node shared memory provided significantly greater bandwidth on all machines, which is in contrast to previous testing on Blue which had shown no significant difference between using on-node shared memory or the switch for communication between tasks located on the same node. Our comparisons of MPICH and IBM's MPI generally showed significantly better performance for IBM's implementation on all of the machines, both for point-to-point and collective operations, although there were some exceptions.

Our initial results motivated the implementation of a test for MPI_Scan that accurately measures its operation latency, i.e. the time required from the start of the operation until it is completed at all participating MPI tasks. The original test of MPI_Scan in Sphinx simply measured the time at task zero for repeated calls to MPI_Scan. Our results for this test, which measures the overhead at task zero of the MPI_Scan operation, indicate that the performance of MPICH's MPI_Scan implementation was constant as the number of tasks is increased, as shown in Figure 4. The MPICH implementation uses a linear algorithm in which task zero sends to task one which then sends to task two and so on until finally task N-1 receives the partial result from task N-2. Thus, our original test suffers from the pipelining effect [2] in which the messages from successive MPI_Scan calls are completely overlapped. We solved this problem similarly to our method for accurately measuring MPI_Bcast operation latency [3]: have task 0 wait to receive an acknowledgment from task N-1 between successive calls to MPI_Scan. Results for this test, as shown in Figure 5, demonstrate that the cost of MPI_Scan under MPICH increases linearly with the number of tasks, as we expect from examining the implementation.

Our results for other MPI collective operations helped reveal some performance deficiencies in IBM's MPI library. Some were caused by optimizations appropriate for systems with uniprocessor nodes but actually degraded performance with SMP nodes. We are working with IBM to resolve other performance problems for the collective implementations. For example, collective operation performance does not improve as significantly as we would expect with on-node shared memory compared to using the switch for all MPI communication, given the approximate reduction of three levels in communication tree height with sixteen way SMP nodes.

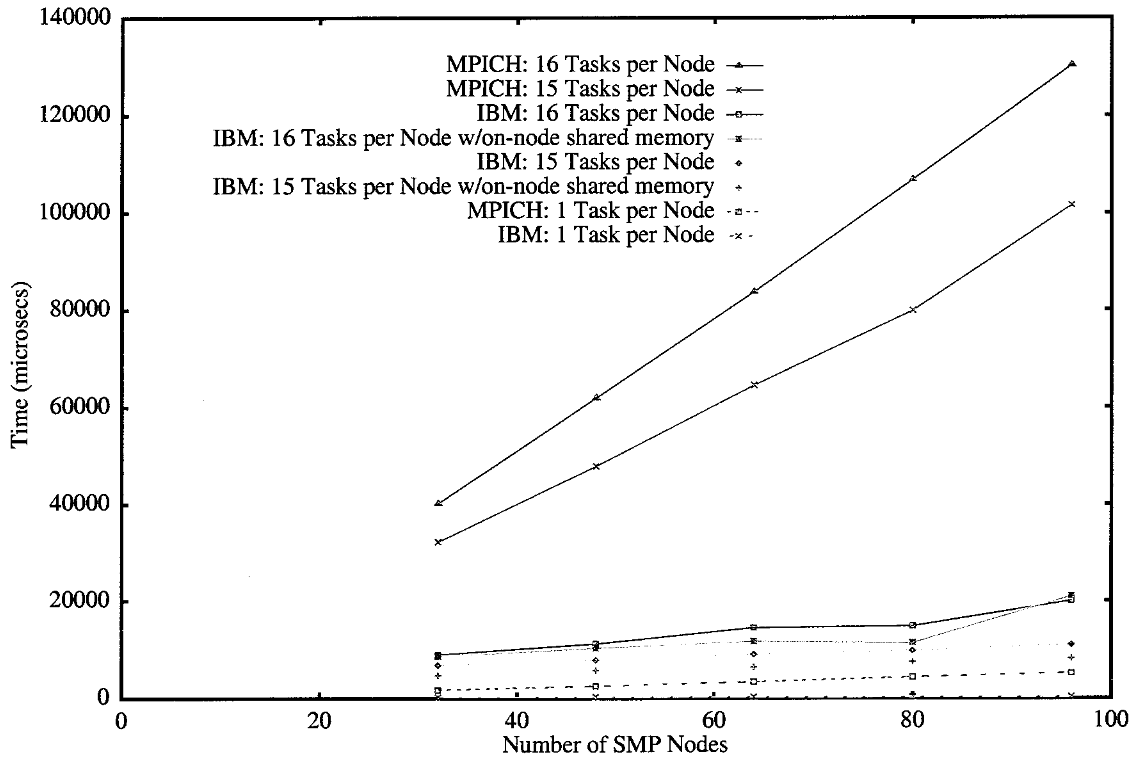

Figure 5: Operation Latency for MPI_Scan of 256 Bytes

6. Conclusion

We presented results of the run-time systems performance testing portion of the ASCI PSE Milepost. Our results demonstrate that the performance of these systems is sufficient to support ASCI White's approximately four-fold increase in processing power over Blue Pacific. We identified a performance anomaly for the Pthreads implementation on White: setting the YIELDLOOPTIME environment variable has a significant impact of the performance of Pthread mutexes with unbound threads. Finally, we presented results for the first scalable, accurate benchmark of the MPI_Scan collective operation; it shows the poor scaling of the MPICH implementation while indicating that IBM's is a more scalable implementation.

7. References

[1] J.M. Bull, "Measuring Synchronisation and Scheduling Overheads in OpenMP," *Proceedings of the First European Workshop on OpenMP*, 1999, pp. 99-105.

[2] M. Bernaschi and G. Iannello, "Collective Communication Operations: Experimental Results vs. Theory," *Concurrency: Practice and Experience*, 1998, Vol. 10, No. 5, pp. 359-386.

[3] B.R. de Supinski and N. Karonis, "Accurately Measuring Broadcasts in a Computational Grid," *Proc. of the 8th Intl. Symp. on High Performance Distributed Computing*, 1999, pp. 29-37.

[4] B.R. de Supinski and J. May, "Benchmarking Pthreads Performance," *Proc. of the 1999 Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, 1999, pp. 1985-1991.

[5] W. Gropp, E. Lusk, N. Doss and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, 1996, Vol. 22, No. 6, pp. 789-828.

[6] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," *International Journal of Supercomputing Applications*, 1994, Vol. 8, No. 3/4, pp. 165-414.

[7] OpenMP Architecture Review Board, *OpenMP Fortran Application Program Interface*, 1997.

[8] OpenMP Architecture Review Board, *OpenMP C and C++ Application Program Interface*, 1998.

[9] R.H. Reussner, "User Manual of SKaMPI, Special Karlsruher MPI-Benchmark," *Tech. Report*, University of Karlsruhe, 1998.