Results from Measuring the Performance of the NAS Benchmarks on the Current Generation of Parallel Computers and Observations Drawn from these Measurements

Daniel M. Pressel
US Army Research Laboratory (ARL), Aberdeen Proving Ground, MD
dmpresse@arl.army.mil

Abstract

The NAS Division at NASA Ames Research Center has developed a highly respected and widely used set of benchmarks for parallel computers. These benchmarks are based on the needs of computational fluid dynamics applications, but appear to have relevance to other disciplines as well. Unfortunately, NAS last published a collection of results for a wide range of systems in November of 1997. Given the rapid level of change in the field of computers in general, and high performance parallel computing, it seemed appropriate to try and fill in some of the gaps. As such, the major computational assets of the ARL-MSRC is being benchmarked, with the intent to publish the results on an ARL website. While making these measurements, several observations and conclusions were drawn. These observations and conclusions will be discussed at length in this document.

1. Introduction

The NAS Division at NASA Ames Research Center has developed a highly respected and widely used set of benchmarks for parallel computers[1]. These benchmarks are based on the needs of computational fluid dynamics applications, but appear to have relevance to other disciplines as well. In Reference 2, a subset of the NAS benchmarks (BT, CG, LU, and SP) were compared to the Linpack Parallel benchmark[3], Stream benchmark [4], and peak processor speed (in MFLOPS). It was found that collectively the subset of the NAS benchmarks were the best predictors of the performance of applications in Computational Chemistry and Material Science, Climate/Weather/Ocean Modeling and Simulation, Computational Fluid Dynamics, and Computational Structural Mechanics when using 1–1152 processors on 15 different system types from 6 different vendors. Unfortunately, NAS has not published a collection of results for a wide range of systems since November of 1997. Given the rapid level of change in the field of computers in general, and high performance parallel computing, it seemed appropriate to try and fill in some of the gaps.

At the US Army Research Laboratory – Major Shared Resource Center (ARL-MSRC), a number of moderate-to-large sized systems from SGI, IBM, and most recently Linux Networx are currently in use. Additionally, three new systems from these vendors, including two 2000+ processor clusters are due to come on line in the next few months. As such, it seemed as though a good starting point would be to benchmark the distinct major systems at the ARL-MSRC, with the intent to publish the results on an ARL website. While making these measurements, several observations were drawn. These observations and conclusions will be discussed at length in this document. They are grouped into the following categories:

- The effect that node configuration has on performance.
- System usage policies and performance.
- Fine grain parallelism, message passing latency, and performance.

This project was made possible by a grant of computer time from the DoD High Performance Computing Modernization Program.

2. The Effect that Node Configuration has on Performance

Almost all high performance computer (HPC) systems are now made with one or more levels of cache memory sitting between the processor and main memory.

The purpose of the cache memory is to bridge the performance gap between the bandwidth of main memory

Table 1. Calculating the usable per processor memory bandwidth for a series of hypothetical systems representative of the current state of the art in system design

Number of processors per node	Cache line size (bytes)	Number of outstanding cache misses/ prefetch events (per processor)	Per processor usable memory bandwidth for loads (MB/sec) assuming			
			A peak memory bandwidth of 600 MB/sec (150 NS latency - 1200 MB/sec limit on chip interface)	A peak memory bandwidth of 1200 MB/sec (200 NS latency- 1200 MB/sec limit on chip interface)	A peak memory bandwidth of 2400 MB/sec (250 NS latency - 1200 MB/sec limit on chip interface)	A peak memory bandwidth of 4800 MB/sec (300 NS latency - 1200 MB/sec limit on chip interface)
1	64	1	427	320	256	213
1	64	2	600	640	512	427
1	64	4	600	1200	1024	853
1	64	8	600	1200	1200	1200
1	64	16	600	1200	1200	1200
1	128	1	600	640	512	427
1	128	2	600	1200	1024	853
1	128	4	600	1200	1200	1200
1	128	8	600	1200	1200	1200
1	128	16	600	1200	1200	1200
2	64	1	300	320	256	213
2	64	2	300	600	512	427
2	64	4	300	600	1024	853
2	64	8	300	600	1200	1200
2	64	16	300	600	1200	1200
4	64	1	150	300	256	213
4	64	2	150	300	512	427
4	64	4	150	300	600	853
8	64	1	75	150	256	213
8	64	2	75	150	300	427
16	64	1	38	75	150	213

(including the bandwidth of the node's interconnect) and the much greater ability of the processor to initiate load and store instructions. Many of the systems are equipped with a prefetching mechanism that can improve a processor's ability to stream data from main memory into the processor (usually via the caches). It is important to remember that this mechanism is still limited by the available bandwidth, and may have additional limitations as well. A more complete description of caches and prefetching can be found in References 5 and 6.

Table 1 demonstrates how the available per processor memory bandwidth varies depending on the design of the system. It should be noted that these numbers were not chosen to represent any one system, but are generally representative of the current state of the art.

It should be noted that for systems with a relatively limited per node memory bandwidth, either adding support for prefetching and/or increasing the cache line size will be of limited benefit. Furthermore, adding additional processors to the node may have limited effect on performance. In fact, since a multiprocessor operating system kernel will in general incur a greater level of overhead when going from a single processor node to a dual processor node, one might actually lose performance on a per node basis. However, if the application was well tuned for large caches and if the cache is large enough relative to the amount of work assigned to the processor, then the application may no longer be memory bandwidth limited. A prime example of where this worked can be seen in Figure 1, where for larger numbers of processors the performance of the LU benchmark for the class B data set is quite impressive. Unfortunately, as one can see in Figure 2, some applications will need more than 512KB of cache if they are to ever reach this point. In this figure, the delivered performance of the SGI Origin 3000 is very similar to that of the Pentium 4 cluster, even though the

peak-speed on a per processor basis of the cluster 7.5 times greater! The key difference appears to be that the SGI Origin processors are each equipped with 8MB of cache, while on the Pentium 4 cluster, each processor is equipped with only 512KB of cache. Clearly, as equipped, problems represented by the CG benchmark would be best run on other platforms.

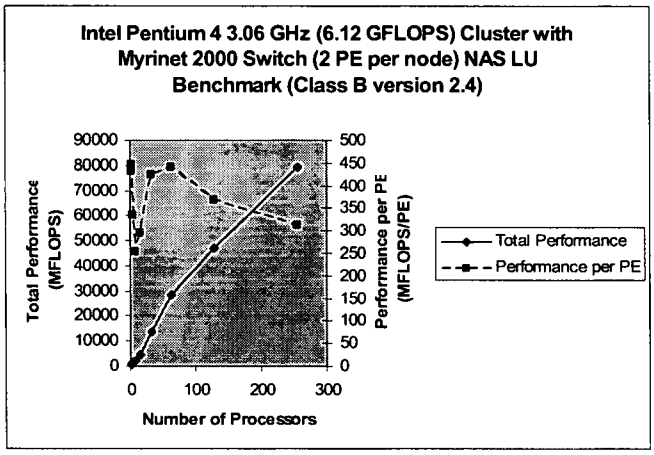

Figure 1. An example of limited memory bandwidth hurting the performance of the second processor on a two processor node until the amount of work per processor is reduced to the point that the working set fits in the 512 KB cache

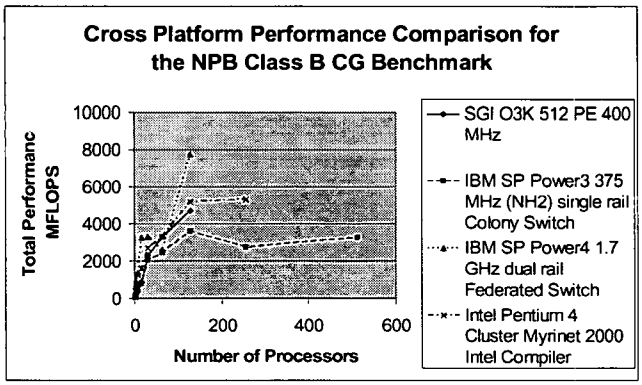

Figure 2. An example of an application that's working set fits in an 8 MB cache, but fails to fit into a 512 KB cache, even for larger numbers of processors

There is also the question of when using nodes with more than one processor (generally referred to as SMP [Symmetric Multi-Processor] nodes, how many of the processors should be used? This might seem to be a silly question. Why would one not want to use all of the processors per node? In fact, when running these benchmarks, an effort was made to run them under production conditions. Therefore, it is safe to assume that in many cases all of the processors of at least some nodes were being used. Unfortunately, there are two draw backs to using all of the processors on a node. The first is that one is competing with the operating system for the attention of at least one processor. For finer grained applications (especially those written using Unified Parallel C [UPC], High Performance Fortran [HPF], OpenMP, and SHMEM), one may see improved performance when using N-1 processors per node. A related argument arises when overlapping communication with computation. In that case, the communication can put additional strain on both a processor and the memory system.

3. System Usage Policies and Performance

System usage polices can affect performance in a variety of ways. This can range from being unable to schedule a job, or at least unable to get it scheduled in a reasonable amount of time, to having jobs fail shortly after they start to run, to the more common problem of poor/highly variable levels of performance. Before continuing on, the author wishes to thank Thomas Kendall, Phillip Matthews, and the entire staff of the ARL MSRC for their help in working out solutions to many of these problems.

Based on this experience, it is recommended that all systems be configured in the following manner:

- The default core dump size should be set to zero. Most users do not want their core dumps and the process of dumping several (possibly hundreds of) Gigabytes of core can be very disruptive. At the same time, the hard limit for the core dump size should be sufficiently large as to support the creation of a core dump when needed.
- The queuing system should not by default initiate a core dump of a user's application when the user requests that the job be terminated.
- On some systems the default number for the maximum number of files that a job may have open may be insufficient to run MPI jobs on larger numbers of processors. Since it may be difficult for the user to identify what is going wrong, it is suggested that the default value should be in the range of 1500-2000 files for systems with fewer than 1024 processors. For larger systems running very large pure MPI jobs (as opposed to hybrid MPI/OpenMP jobs), one may need to increase the value of this limit to well beyond 2000 files.
- The default queuing policy on systems with 2–4 processors per node should be to assign a job as many dedicated nodes as possible, even if the job did not request dedicated nodes, completely filling those nodes before filling up shared nodes.

- The default queuing policy on systems with SMP nodes with 8 or more processors per node should be to assign a job as many dedicated nodes as possible, even if the job did not request dedicated nodes, using N-1 processors per node. Any left over processes could then be mapped to partially filled shared nodes. Only when the system runs out of partially filled shared nodes should the queuing system make use of the last processor per node. Even then, one might argue in favor of leaving such processors for use by debug jobs, interactive jobs, post processing jobs, and high priority jobs that would otherwise be delayed.
- The default memory usage policy on clusters of SMPs (this includes those running UNIX as well as those running Linux) should be tuned in such a manner as to reduce/eliminate the potential for paging. At the same time, the hard limits should be set high enough that users running shared memory and hybrid runs are supported. Provisions should also be made on at least some systems to support those running serial jobs with oversized memory requirements.

Current DoD HPCMP policies discourage users from "stuffing" the queues with large numbers of jobs, especially if they are hard to schedule jobs. However, sometimes the presence of these jobs is the best argument that there is a need to provide support for such jobs. Furthermore, once the resources have been made available, if the job finishes after just a few hours (not to mention those runs that may take less than one hour), the site would face being penalized twice. Once for the expansion factor, and a second time for the low utilization level while the system is being drained. If the queues filled with some critical mass of jobs requesting similar resources, then at least the low utilization level can be amortized over a more reasonable level of work. Additionally, one should be able to eliminate many of the periods of low utilization due to draining by running a significant number of hard to schedule jobs one after another. This does not mean that the users should have carte blanche to stuff the queues with an unreasonable number of jobs. As was discovered when submitting the runs for this study, stuffing queues to the extreme may inadvertently interfere with the normal functioning of the queuing system.

Up until now, most of what has been discussed applied either solely to clusters of SMPs, or to both clusters of SMPs and to large SMPs such as the SGI Origin 3000. However, in carrying out this study, it was observed that certain aspects of the SGI Origin 3000, and presumably other large SMPs, require special attention. At the ARL-MSRC, most jobs run on the SGI Origin 3000 run in CPU sets (sometimes call processor sets). In theory this gives these jobs sole access to the processors in their set. In most cases this helps to reduce the variability in run time that the SGI Origins are notorious for. Even so, there continued to be a limited number of complaints concerning the variability in run time. The general wisdom was that for one reason or another the job was run outside of a processor set. At the very least, such jobs are at an increased risk of moving around the system. For an architecture that incorporates a Non Uniform Memory Architecture (NUMA), this will almost certainly increase the memory latency, while decreasing the usable memory bandwidth. Under certain circumstances, jobs running outside of a processor are also subject to time sharing, which needless to say is highly undesirable when talking about parallelized applications.

Recent measurements indicate that there are at least two other circumstances in which the performance of a job is subject to degradation, even when the job is run in a processor set. It appears that if the system becomes overloaded (the number of active processes exceeds the number of processors), the performance of all of the jobs on the system can be adversely affected by a factor of two or more. This can be difficult to deal with if any jobs are allowed to run outside of a processor set. The load factor may appear to be reasonable, however if any of the processor sets are underutilized, the effect of running any jobs outside of a processor set in an attempt to improve the apparent utilization level can be quite unfortunate. The second scenario involves jobs that use significantly more than "their fair share" of memory. On most clusters, such jobs cannot even run. However, one of the benefits of a large shared memory system is its support for this type of job. Of course when taken to the extreme, there is a risk of either paging/or of jobs being unable to run due to a lack of available resources. This is not what is being discussed here. Rather, the problem appears to be that effectively the SGI Origins behave as though they have a multi-banked memory system. If some of the banks fill up, the remaining jobs may inadvertently experience hot spots in the memory system. While this author is well known for running such jobs from time to time, and is therefore reluctant to recommend that such jobs be banned, it does appear as though an increased level of prudence would be advisable in the future. In order to better illustrate these points, Figure 3 shows a frequency count for the ratio of the worst run time versus the best run time for each of the benchmarks (Class W 1–64 processors), (Class A, B, C, and D 1–256 processors) on the ARL MSRC's 256 processor Pentium 4 cluster. Similarly, Figure 4 is for the ARL MSRC's 1024 processor IBM SP with Power3 (NH2) processors (Class W 1–64 processors), (Class B and C 1–529 processors),

and (Class D 1–1024 processors). Figure 5 is for the ARL MSRC's 512 processor SGI Origin 3000, although only a handful of runs were done using more than 256 processors. On the ideal system, all of the values should be as close to 1.0 as possible.

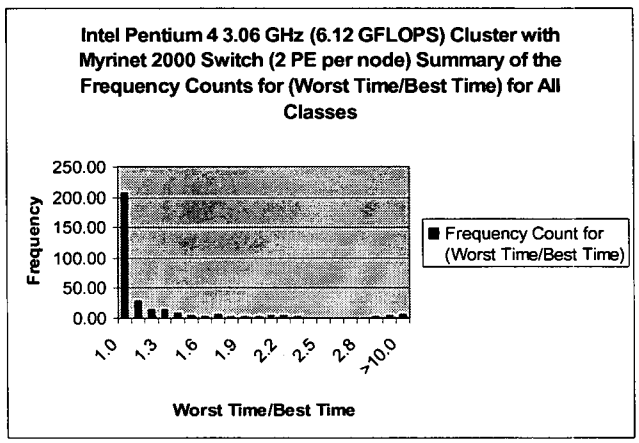

Figure 3. The variability in run times on the Intel Pentium 4 cluster at the ARL-MSRC (0.1 increment binning)

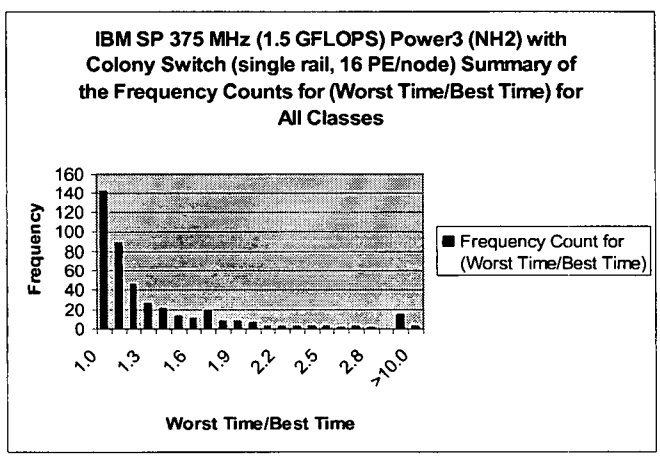

Figure 4. The variability in run times on the IBM SP Power3 (NH2) at the ARL-MSRC (0.1 increment binning)

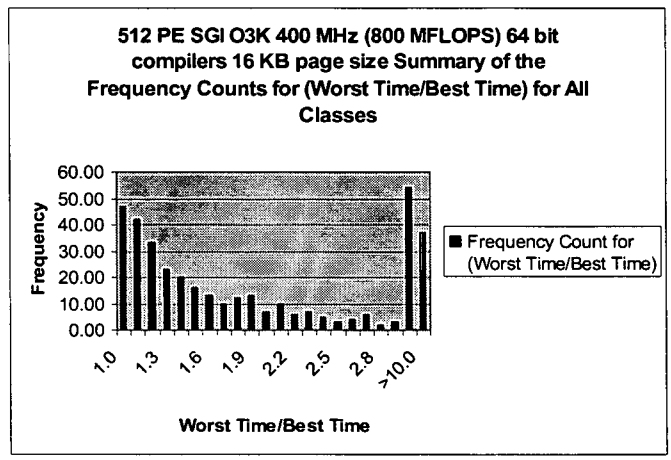

Figure 5. The variability in run times on the SGI Origin 3000 at the ARL-MSRC (0.1 increment binning)

4. Fine Grain Parallelism, Message Passing Latency, and Performance

One problem that appears to have received little attention in the literature, but which became painfully obvious when running these benchmarks is the question of what constitutes fine grain parallelism. Traditionally, programs that are successfully parallelized using PVM or MPI are considered to have a coarse grain of parallelism. Those using HPF, UPC, or OpenMP have a fine grain of parallelism. Those parallelized using SHMEM most frequently fall somewhere in between. However, with increasing processor speeds, many of the NPB runs for classes W, A, and B showed limited scalability past 128 processors (in some cases past 16–32 processors) on the newer systems. This is a clear indication that these runs should no longer be considered to be coarse grained. The fault is not with the runs, nor is it with the systems. Rather, as everything got faster, the time between synchronization events has shrunk below the threshold for coarse grained runs. This is a clear warning sign for those running programs parallelized several years ago. It may be possible to run existing programs at the current levels of performance using ever fewer processors for some time to come. However, if one expects to run the current problem sizes on the same or larger numbers of processors, then it may be time to revisit the strategy used to parallelize the programs. This is most likely to be the case with programs parallelized using either a hybrid programming strategy or a software virtual shared memory (SHMEM, HPF, UPC, Linda, Co-Array Fortran, Global Arrays, etc.). As processors get faster, it is easy to see how the bandwidth for interprocessor communication might be able to keep up. However, the limits on message passing latency seem to be harder to avoid.

5. Conclusions

So far the NPB benchmarks (classes W-D) have been run on four of the largest systems at the ARL-MSRC. The FT benchmark would not compile with the Intel compiler. Experiments are underway to compare the performance of runs compiled with the Intel and PGI compilers on the Pentium 4 cluster.

A number of observations and conclusions that were made while running these jobs have been discussed. It is expected that additional systems will be benchmarked as time permits and that the results from all of these runs will be published on the web at a future date. It is our hope that others will undertake similar studies at their sites, and similarly publish their results on the web.

References

1. The NAS Benchmark (NPB) home page can be found at http://www.nas.nasa.gov.

2. Dongara, J., "Linpack Benchmark-Parallel." table for the Linpack Benchmark, published electronically at http://www.netlib.org.

3. McCalpin, J., "Equivalent MFLOPS" table for the STREAM Benchmark." published electronically at http://www.cs.virginia.edu/stream.

4. Pressel, Daniel M. and Jelani Clay, "Benchmarking the Benchmarks." *ARL-TR-2805*, US Army Research Laboratory, September 2002.

5. Pressel, Daniel M, "Cache-Based Architectures for High Performance Computing." *ARL-MR-528*, US Army Research Laboratory, February 2002.

6. Pressel, Daniel M., "Fundamental Limitations on the Use of Prefetching and Stream Buffers for Scientific Applications." *ARL-TR-2538*, US Army Research Laboratory, June 2001.