ARMY RESEARCH LABORATORY

Evaluating Co-Array Fortran and Unified Parallel C

by Earlene L. Thompson and Daniel Pressel

ARL-MR-654 **November 2006**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-MR-654 **November 2006**

Evaluating Co-Array Fortran and Unified Parallel C

Earlene L. Thompson
Southern Illinois University

Daniel Pressel
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2006	Final	1 June 2005–1 October 2005

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Evaluating Co-Array Fortran and Unified Parallel C	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
Earlene L. Thompson* and Daniel Pressel	6UH7CL
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: AMSRD-ARL-CI-HC Aberdeen Proving Ground, MD 21005-5067	ARL-MR-654

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Southern Illinois University, Carbondale, IL 62901

14. ABSTRACT

Earlene L. Thompson worked as a summer intern at the U.S. Army Research Laboratory under the auspices of the Programming Environmental and Training component of the Department of Defense High-Performance Computing Modernization Program. Mr. Pressel was her mentor. Their project was to evaluate the benefits of two emerging languages for High-Performance Computing: Co-Array Fortran (CAF) and Unified Parallel C (UPC). Originally, the project consisted of porting a locally written High-Performance Fortran program to CAF. When it became clear that this project was beyond the intended scope for this project, another project was selected. The alternate project was to download from the web CAF, UPC, and MPI versions of the NAS Benchmarks and to run these benchmarks on a Cray X1 located at the U.S. Army High-Performance Computing Research Center. This report will discuss both parts of Ms. Thompson's internship.

15. SUBJECT TERMS

high-performance computing (HPC), parallel processing, Unified Parallel C (UPC), Co-Array Fortran (CAF)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Daniel Pressel
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	26	19b. TELEPHONE NUMBER (*Include area code*) 410-278-9151

Contents

List of Figures

Acknowledgments

We wish to thank the Department of Defense High-Performance Computing Modernization Program (DOD HPCMP) and the U.S. Army High-Performance Computing Research Center (AHPCRC) for the grant of computer time with which to carry out this project. Additionally, we thank the DOD HPCMP and the U.S. Army Research Laboratory-Major Shared Resource Center (ARL-MSRC) for prior grants of computer time used to compile the data used for comparison purposes. The DOD HPCMP, ARL, MSRC, and the Programming Environment and Training (PET) component of the DOD HPCMP have provided a combination of financial, logistical, and personal support for this project. Finally, our thanks go out to the following people from other sites who helped us in so many ways:

- Paul Muzio, Barbara Bryan, Jeff Dawson, Hung Nguyen, and the entire staff of the AHPCRC.
- Tom Kendall, Phil Matthews, and the entire system staff of the ARL-MSRC.
- Charlie Nietubicz, Raju Namburu, Dale Shires, Nancy Schepleng, and numerous other employees of and contractors for the High-Performance Computing Division, Computational and Information Sciences Directorate, ARL.
- The entire PET staff located at the ARL-MSRC.
- Tarek El-Ghazawi of George Washington University.
- Charles Koebel of Rice University.
- Haoqiang Henry Jin of the NAS group at NASA Ames Research Center.

Finally, Mr. Pressel would like to thank Earlene Thompson for perseverance during a project that proved to be a bit tougher than first expected.

INTENTIONALLY LEFT BLANK.

1. Introduction

Every year, the U.S. Army Research Laboratory (ARL) in conjunction with the Programming Environment and Training (PET) component of the Department of Defense High-Performance Computing Modernization Program (DOD HPCMP) hosts several interns. These interns are always college students, most of whom are either juniors or seniors. Their principal duties involve carrying out research involving high-performance computing (HPC) and/or the related disciplines of high-performance networking and scientific visualization. This report is based on the work and experiences of Earlene L. Thompson during her internship in the summer of 2005.

The goal of this topic was to perform preliminary work with regard to Unified Parallel C (UPC) and/or Co-Array Fortran (CAF). Neither of us had any background in either of these languages although both of us had experience using C. Additionally, the mentor had significant experience using Fortran, parallelizing programs, and UNIX/LINUX-based systems. The rationale for choosing these languages has to do with the Defense Advanced Research and Projects Agency (DARPA). They have a project called High Productivity Computing Systems (HPCS) whose goal is to develop PetaFLOP computers. There are currently three vendors working on this project. In addition to developing hardware for this project, the vendors are required to develop new programming environments to facilitate the use of the emerging platforms. One of the vendors is Cray, Inc., and they have selected these two languages as part of their proposal. While the languages have not been finalized (adopted by a national/ international standards-setting body), they appeared to have matured to the point that this would be a viable project.

With this background, it was decided that the project would consist of porting a meaningful program to one of the two languages. At the time, porting from High-Performance Fortran (HPF) to CAF seemed to be an obvious match. An early version of a workhorse program developed within our branch seemed to be a reasonable, if not perfect, choice. This program is called COMPOSE and was written by Ram Mohan and Dale Shires. COMPOSE is a composites manufacturing simulation code used to optimize the fabrication process. The HPF version of the code consists of ~3000 lines of Fortran code, including 130 lines specific to HPF. The assumption was that most of the effort in porting this program to CAF would center around these 130 lines (primarily declaration statements). When it was shown that this assumption was incorrect and that all 3000 lines would probably need to be modified (to provide access to shared arrays), it was decided that this was beyond the scope of this project. Therefore, a new project needed to be quickly selected.

Attempting to maintain the spirit of the original project, it was decided to measure the performance of the NAS benchmarks on the Cray X1. This project was selected since these are widely reported and highly respected benchmarks (*1*). Of equal importance, references on the

web indicated that a message passing interface (MPI) version of these benchmarks could be downloaded from NASA Ames Research Center, a UPC version of some of these benchmarks could be downloaded from George Washington University (GWU), and a CAF version of these benchmarks could be downloaded from Rice University. Therefore, it was expected that, given the limited amount of time remaining, we would be able to run all of these benchmarks for a range of processor counts for at least some of the smaller benchmark sizes (e.g., W, A, and B). Our results, conclusions, and the problems we ran into will comprise the bulk of this report.

2. The Choice of Hardware and Related Topics

When selecting a platform on which to carry out this research, the obvious choice was a Cray X1 or the newer Cray X1E. The U.S. Army High-Performance Computing Research Center (AHPCRC), which is supported by both the U.S. Army and the DOD HPCMP, has a small Cray X1 and a much larger Cray X1E. We requested and received access to the Cray X1 which is reserved primarily for educational purposes. The compilers on this system support both UPC and CAF. The system consists of 4 nodes, each with 16 SSP processors and 16 GB of main memory. Each SSP processor has a scalar unit rated at 400 MFLOPS and a vector unit rated at 3.2 GFLOPS. Jobs may either request processors in terms of SSPs or MSPs. In the later case, each MSP consists of four tightly coupled SSPs, with the compiler being called on to split the work among as many of the SSPs in each MSP as it can. Additional information on the design of the Cray X1 can be found on the Cray website http://www.cray.com.

It should also be noted that the processors are grouped into nodes, with each node consisting of 16 SSPs (4 MSPs). Ordinarily, one of these nodes is reserved for use as an Input/Output node/Login node, which can also be used to run interactive jobs using up to 16 SSPs (4 MSPs). The batch system has dedicated access to the other three nodes. We were given special access to this system to run 49 processor jobs, but were unable to run 64 processor jobs. The main concern here is that it is very difficult to show high levels of performance when using all of the processors in a system (almost any system). There is enough interference from system daemons, which even if everyone is asked to logout, rarely will one show anything approaching linear speedup. In fact, most large systems have dedicated I/O nodes that may not even be mentioned when discussing the system's size.

Another issue is the unusual approach that Cray takes to scheduling jobs on their systems. Jobs of higher priority may actually cause a lower priority job to lose its processors for an extended period of time. Even when dealing with jobs of equal priority, various forms of time sharing seem to be taking place. This can range from having a job stream rolled out when a new job starts up, to having two or more jobs actually time sharing their processors with varying time quanta (probably ranging from under a second to many minutes in duration). The documentation indicated that this could result in widely varying run times, but should not

significantly affect the CPU time. In some cases, the operating system left hints in the output file as to ways to improve the run time. In some cases, we took advantage of these hints and reran the jobs. In other cases, the effect seemed to be small enough that the hint was ignored and work continued on with other runs. In most cases, these effects were most pronounced when using small numbers of processors (e.g., 1–4 SSPs or 1 MSP).

It should be noted at this point that our principal goal was getting our feet wet when using either CAF and/or UPC. It was not anticipated that there would be sufficient time to worry about modifying the code, or even putting in a modest number of compiler directives designed to improve either vectorization or the performance of the code when running in MSP mode. Therefore, it was expected that our runs would be made with the highest levels of optimization (scalar, vector, and MSP) supported by the compiler, and we would just have to hope for the best.

3. Porting COMPOSE to HPF

Since HPF and CAF are both extensions to the Fortran standard, it was assumed/hoped that only modest changes would be required to port a program from HPF into CAF. This would possibly be the case when dealing with a program using structured grids. However, COMPOSE uses unstructured grids and was difficult to parallelize using HPF in the first place. After spending several weeks working on this project, we developed serious misgivings about it. From our slightly naïve perspective, it appeared as though it would be necessary to use full Co-Array syntax for virtually all of the array references in order to support a mix of local and remote memory accesses on a seamless basis. In contrast, HPF had largely done that automatically based on distribution statements in the declaration section of each program unit.

This was significantly more work than we had anticipated. Since parallelization is normally an all or nothing proposition—some forms of shared memory parallelism (e.g., OpenMP) being the principal exception—it was decided to look for another project to work on. It is possible that it might have been easier to parallelize COMPOSE using UPC, but we did not look into that possibility. Even if the parallelization had been straightforward, using UPC would have had two distinct disadvantages. The first is that translating 3000 lines of Fortran code into C code by hand is nontrivial (there are automatic Fortran-to-C translators, but their output is so cryptic as to make the translated code unmaintainable). The second issue is that it can be difficult to obtain a reasonable level of performance on a vector processor when using C code. Therefore, it was decided to look for an entirely new project.

4. The New Project: Running the NAS Benchmarks on the Cray X1

It was desirable that any new project have the following characteristics:

- Use CAF and/or UPC,
- Have another version parallelized using either MPI and/or OpenMP available for comparison purposes, and
- Fit into the remaining time frame given the available human and computer resources.

Based on these criteria and Mr. Pressel's prior experience running the MPI version of the NAS Parallel Benchmarks at the U.S. Army Research Laboratory – Major Shared Resource Center (ARL-MSRC), it was decided to try and run these benchmarks. A search of the web showed that researchers at George Washington University (GWU) and elsewhere had already ported several of the NAS benchmarks to UPC. A copy of these benchmarks was freely available from a website at GWU. Similarly, a copy of a CAF version of the NAS benchmarks was freely available from a website at Rice University. While the latter was based on a slightly older version of the NAS benchmarks (version 2.3 while the most recent MPI version is 2.4), this did not appear to be much of a problem.

An important reason for selecting this project is that the original software was developed by the NAS Division at NASA Ames Research Center and is one of the most 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 Pressel and Jelani (*2*), a subset of the NAS benchmarks (BT, CG, lower uppercase decomposition [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 six different vendors.

The UPC versions of the NAS benchmarks (NAS-UPC) were easily downloaded from the GWU site, and Mr. Pressel had previously downloaded the MPI versions of the NAS Benchmarks (NAS-MPI) (version 2.4) for another project. Efforts to download the CAF version of the benchmarks (NAS-CAF) from the website failed. However, with help from Charles Koebel, we were able to obtain a copy of those benchmarks as well. It would be nice to report that our troubles ended there, but unfortunately, this is not the case. Even though both of us made a serious effort to compile and link both NAS-UPC and NAS-CAF, we kept meeting

with failure. Jeff Dawson of the AHPCRC provided additional guidance, but we just could not get those jobs to compile and link no matter what we tried (including renaming the files, and making simple changes to code in response to compiler error messages).

Eventually, guidance was received from GWU that the version of the NAS-UPC benchmarks that is on their web site is based on a later version of the UPC standard than that supported by the Cray compilers we had access to, and that it would be impossible to run these benchmarks on the Cray X1 (*5*). Apparently, another version of the NAS-UPC benchmarks does exist, but was not currently available for downloading. There were many problems with NAS-CAF benchmarks. Unfortunately, the source of several of those problems remains a mystery. Fortunately, the NAS-MPI benchmarks all compiled on the first try. Unfortunately, we were unable to get the LU benchmark to run properly on the Cray X1, even when all optimizations were disabled using the –O0 option.

5. Procedure

In general, Cray seems to recommend using MSP mode. While it may be possible that for well-tuned code that will normally be the best choice, there were serious concerns that the compiler would have trouble using all of the SSPs per MSP when running untuned code in MSP mode. Therefore, it was decided that initially we would concentrate on running in SSP mode. Eventually MSP mode runs were also made, but in many cases they were considerably slower than SSP mode runs using a comparable number of processors (remember, 4 SSPs = 1 MSP, so an example of a comparable number of processors is 4 MSPs vs. 16 SSPs).

In order to be as fair as possible, without actually adding compiler directives or in some other manner tuning the benchmarks, it was decided to use the highest level of optimization possible. The documentation for the Fortran compiler indicates that –O3 may in fact not be the highest level of optimization possible. Therefore, the Fortran runs were compiled using –O scalar3, vector3, ssp for SSP mode runs, and –O scalar3,vector3 for MSP mode runs. For MSP mode runs, -O scalar3, vector3, stream3, aggress was also tried. However, it seemed to make little or no difference in performance, and some jobs actually ran slightly slower with this combination of options.

It was observed that the embarrassingly parallel (EP) benchmark ran substantially slower on the Cray X1 than on other systems Mr. Pressel had previously benchmarked. Since the EP benchmark spends almost all of its time calculating random numbers, it was assumed that the compiler had failed to vectorize the random number generator. After carefully rereading the documentation, it was found that the benchmarks come with several alternative random number generators. One of the alternative random number generators (randdpvec) is computationally less efficient than the default random number generator. However, since it is written in a

manner that supports automatic vectorization by the compiler, it was expected to significantly improve the performance of the EP benchmark, and probably to a lesser extent the performance of some of the other benchmarks as well. Therefore, all of the SSP and MSP runs were made one more time.

The documentation for the C compiler indicated that –O3 was the maximum level of optimization. Therefore, the only other option that was required was –h SSP, which specifies SSP mode. Historically, C code has not performed well on vector processors. Furthermore, it is not clear which sorting algorithms would be the best match for a vector processor. However, it is likely that, as with the random number generator, it would not be the best choice for a scalar processor. Therefore, we did not expect particularly good performance for the IS benchmark on the Cray X1, and unfortunately our expectations were met.

Ms. Thompson made her runs using the 16 processes (SSP) interactive partition, while Mr. Pressel worked with Hung Nguyen of the AHPCRC to work out the details of how to submit batch jobs. For both the interactive and batch jobs, there were a number of problems associated with the way the memory system works. In particular, most jobs are limited to using 1 GB of main memory per process when running in SSP mode (4 GB in MSP mode), and there is no virtual memory. For interactive jobs, one can easily override this using either the –c or –m options for mpirun (up to the limits of available memory on the interactive node – 16 GB in this case). However, for batch jobs, the only solution to the problem is to use the –m exclusive option in combination with the –N option. This specifies how many processes will run on each node, and allows the processes to use as much memory as they need (up to the available limits on the node, 16 GB in this case). Unfortunately, it frequently meant that a dedicated node was required, when a more flexible system might have been able to make better use of the resources.

An associated problem is that one must place a power of 2 number of processes on each node. In some cases, this meant that more nodes were needed than would have been required with a more flexible system. In other cases, it actually prevented us from running some Class D (over time several problem sizes, called classes, have been defined for the NAS benchmarks—W, A, B, C, and D) jobs. They simply could not be scheduled on this system, even though with a more flexible system it should have been possible to run some of these jobs. An example of this is when scheduling a 25 processor job on the three nodes of the batch partition, the ideal schedule would be to place 8 processes on each of two nodes, and 9 processes on the third node. Instead, the best that could be done using only three of the nodes was to schedule 16 processes on one node, and the remaining 9 processes on a second node. While it might have been possible to run the Class D BT benchmark with the ideal schedule, the less-than-ideal schedule lacked sufficient memory with which to run this job. It should be noted that in order to use more than 1 GB of memory/SSP, one needs to request exclusive nodes when running in batch mode. There was no way in which to run this job using all four nodes.

Since we were unable to run NPB-CAF and NPB-UPC, a literature search was conducted using Google. Several papers were found that provided additional results for these benchmarks (*6–8*). Results excerpted from those papers have been used to supplement our own results. Both sources of results will be used in the remainder of this work.

6. Results

This project was intended to concentrate on learning as much as possible in the short time available about CAF and/or UPC. Unfortunately, as was previously mentioned, we were unable to get any of the available NAS benchmarks written in either of these languages to run on the Cray X1. However, others with more time/expertise have faired better. Therefore, the following two charts will compare the performance of NPB-MPI based on our measurements, to those reported for NPB-CAF and NPB-UPC as reported in the literature.

Figure 1 shows that for the BT benchmark of version 2.4 of NPB-MPI (Class B), it vectorized poorly, if at all, on the Cray X1. This was a surprise since Saini and Bailey (*9*) clearly indicate that an earlier version of this benchmark was able to achieve 20% of peak on the Cray C90. After consulting with the NPB group at NAS, we were given access to a specially written version of NPB-MPI. Clearly it performs much better. On the other hand, it is important to note how poorly the UPC version of this benchmark performed. One should also note that, in some cases, the MSP runs appear to have been using only one of the four SSPs per MSP, resulting in their poor showing.

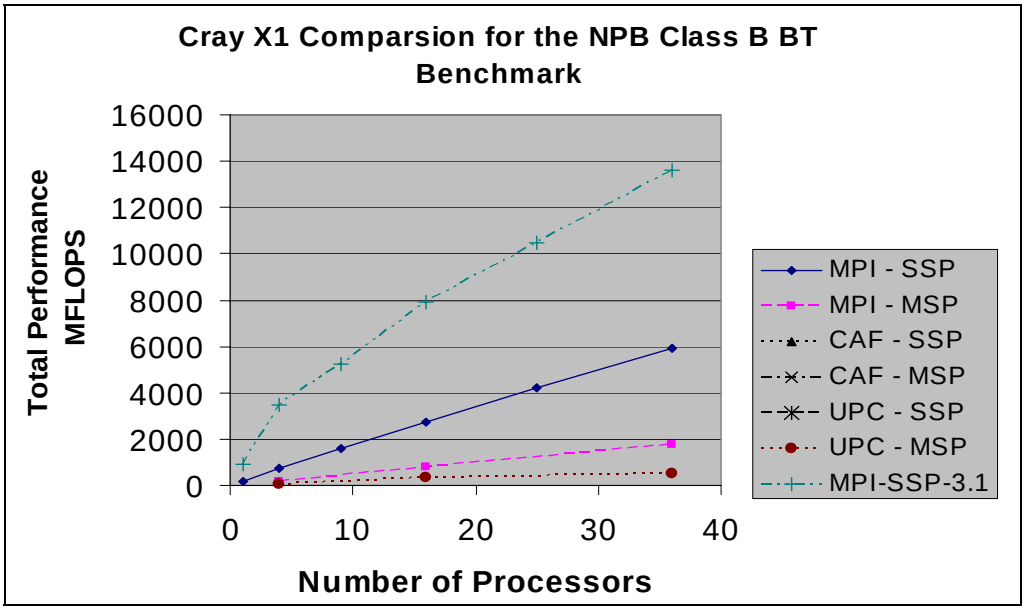

Figure 1. A comparison of the performance of different implementations/modes of running of the NPB 2.4 BT benchmark on the Cray X1.

Similarly, figure 2 shows the results for the MG benchmark of NPB version 2.4 (Class B). However, in this case, the compiler was able to achieve a significant level of vectorization for NPB-MPI. Furthermore, the compiler was able to make efficient use of all four SSPs per MSP. It is interesting to note that the per-SSP performance when running in MSP mode actually exceeded that of runs done in SSP mode by a slight amount. The performance of NPB-CAF (based on version 2.3 of NPB-MPI) appears to be competitive with that of the NPB-MPI in this case. The NPB-UPC benchmark when running in MSP mode was almost as good, although it appears to be running into scalability problems. It is surprising how poorly the SSP mode run for the NPB-UPC benchmark performed.

Unfortunately, for most of the other benchmarks, we were unable to find sufficient results for NPB-CAF and/or NPB-UPC to make it worthwhile to make comparisons. Figures 3–10 will compare the results for each of the eight NPB-MPI version 2.4 (Class B) benchmarks on a cross platform basis (*10*). It should be noted that on a per benchmark basis, the SSP and MSP runs were compared and whichever set of runs showed the best performance at 32 processors (36 in the cases of BT and SP), that set was used for this set of comparisons. These charts will stop at 32 (36) processors since that was the limit of what could be run on the Cray X1 we were using. In general, all of the remaining systems continued to show scaling for these benchmarks when using larger numbers of processors. Presumably, the same would be true when using a larger Cray X1.

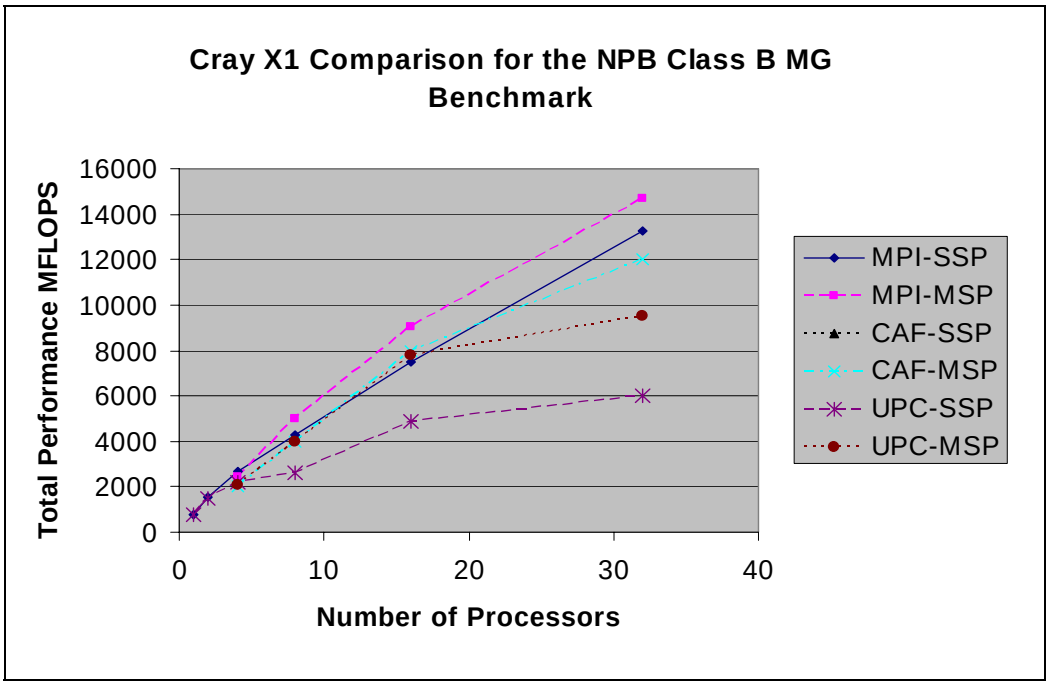

Figure 2. A comparison of the performance of different implementations/modes of running of the NPB 2.4 MG benchmark on the Cray X1.

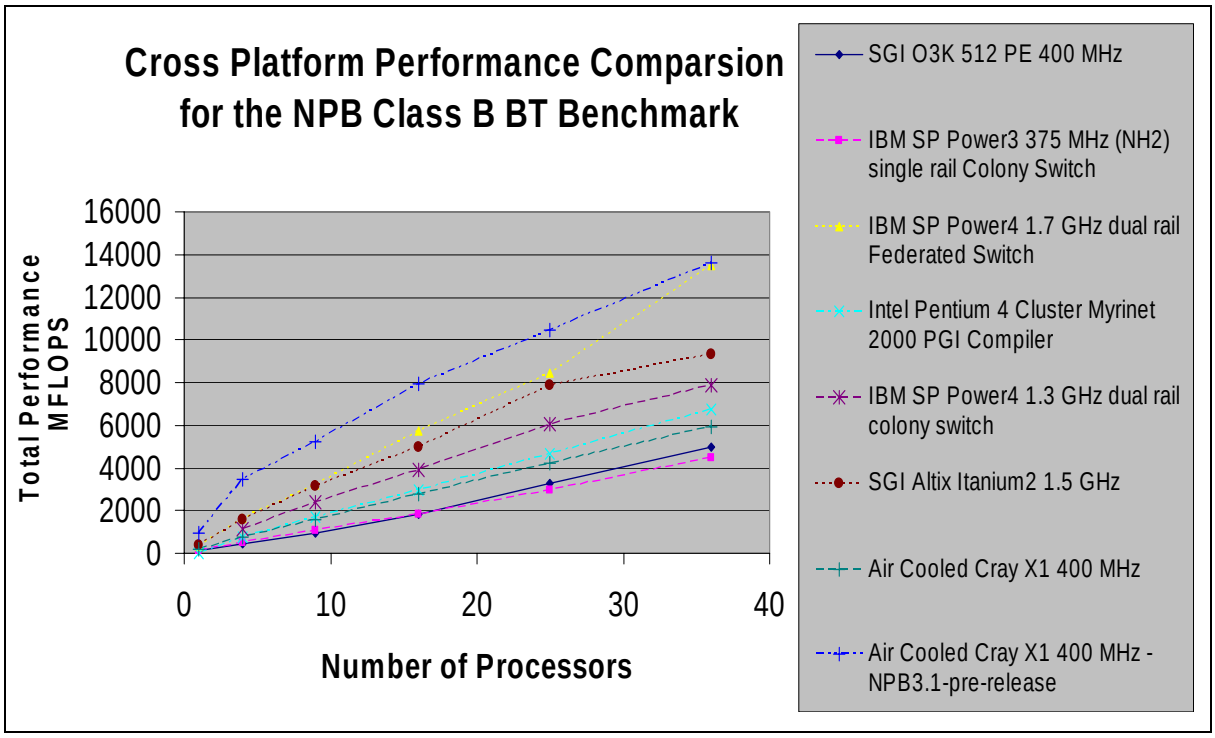

Figure 3. Cross-platform performance comparison for the NPB Class B BT benchmark.

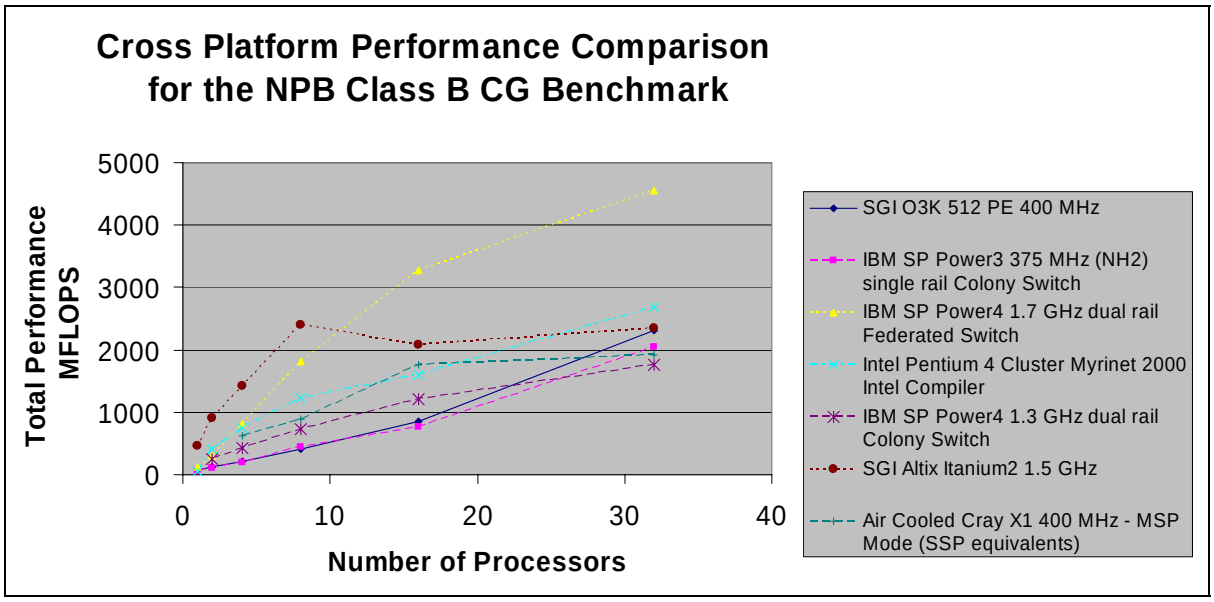

Figure 4. Cross-platform performance comparison for the NPB Class B CG benchmark.

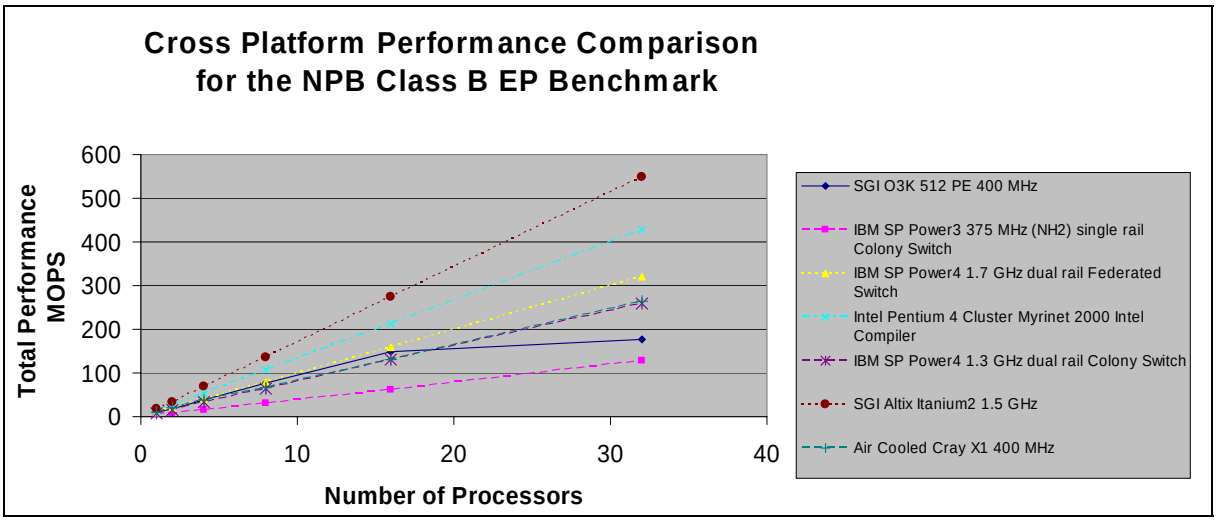

Figure 5. Cross-platform performance comparison for the NPB Class B EP benchmark.

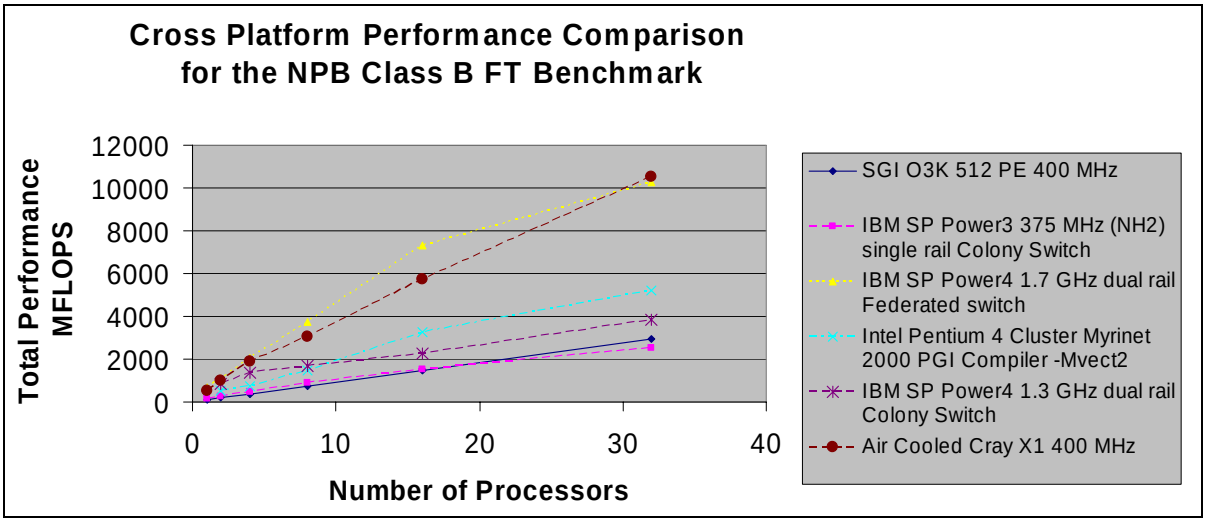

Figure 6. Cross-platform performance comparison for the NPB Class B FT benchmark.

Reviewing figures 3–10, there are some concerns over the scalability of these benchmarks on the Cray X1. In general, given a fixed problem size, for large numbers of processors, one will normally see issues with scalability. However, for 32–36 processors and the Class B NPB-MPI benchmarks, one rarely observes these problems. There are two ways to interpret this data. Cray could still have problems with the implementation of MPI on the Cray X1 (something that they have previously warned about in their training classes). Alternatively, when going to larger numbers of processors, the available parallelism for use in vectorizing the code may be sufficiently limited as to result in suboptimal vector lengths (normally referred to as short vectors).

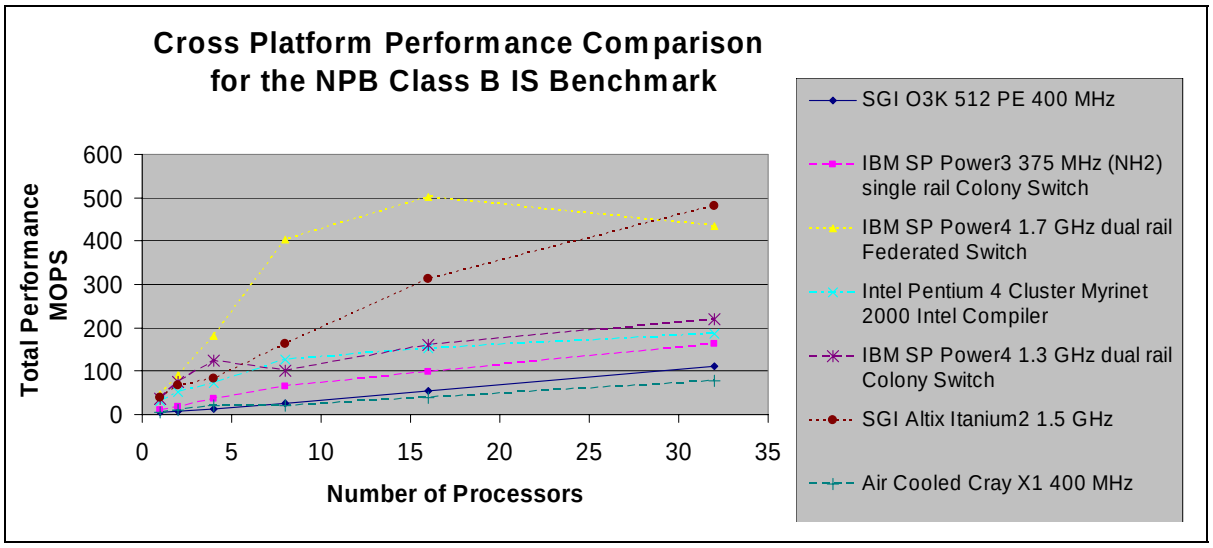

Figure 7. Cross-platform performance comparison for the NPB Class B IS benchmark.

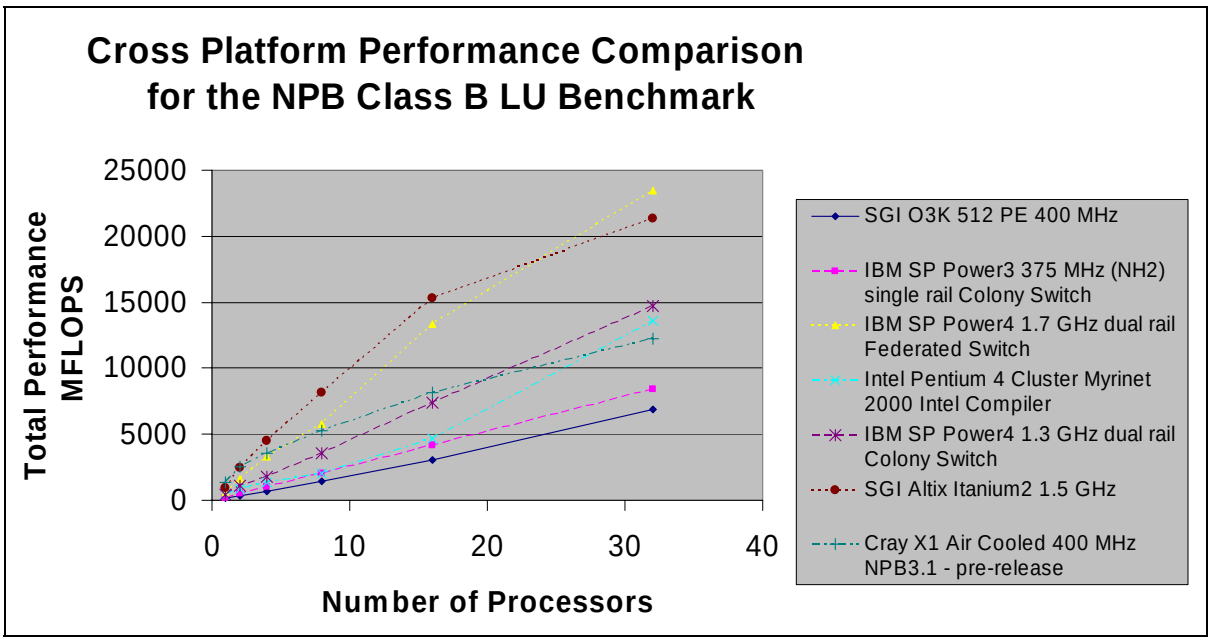

Figure 8. Cross-platform performance comparison for the NPB Class B LU benchmark.

Figures 11 and 12 show the parallel efficiency of the Cray X1 and three other systems for the BT and LU benchmarks as a function of the problem size (Class). Ideally, the parallel efficiency should be close to 1.0, and numbers below 0.70 (70%) are generally considered to be undesirable. It is interesting to note that, regardless of the cause, the Cray X1 has a significantly lower level of parallel efficiency than the other three systems. This helps to explain why, in general, when looked at on a per-SSP basis, the Cray X1 is at best a good performer.

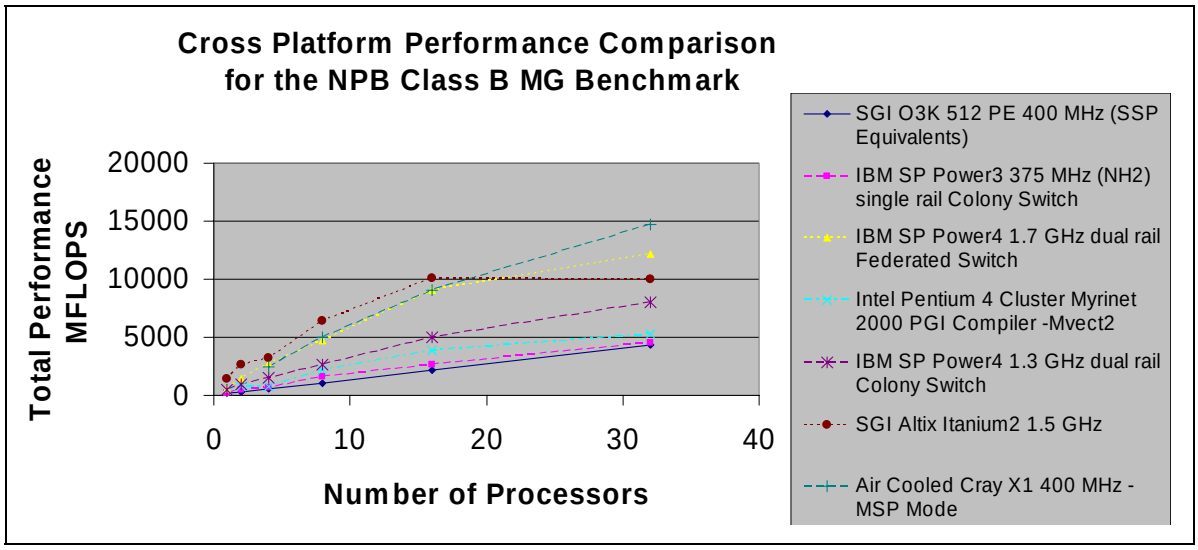

Figure 9. Cross-platform performance comparison for the NPB Class B MG benchmark.

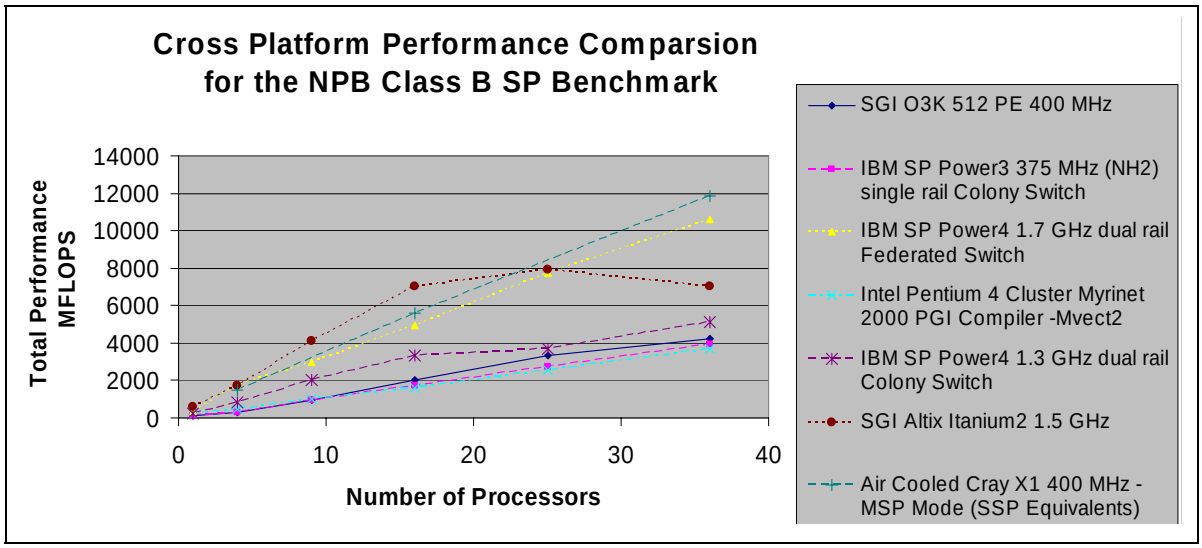

Figure 10. Cross-platform performance comparison for the NPB Class B SP benchmark.

7. Observations and Conclusions

Compared to HPF, CAF is not a simple extension to the Fortran programming language. UPC is still an evolving target, which makes it difficult to write programs that are portable between compilers. UPC, CAF, and HPF all appear to suffer from the same limitation; they are likely to generate a large number of short messages. This makes programs written in those languages

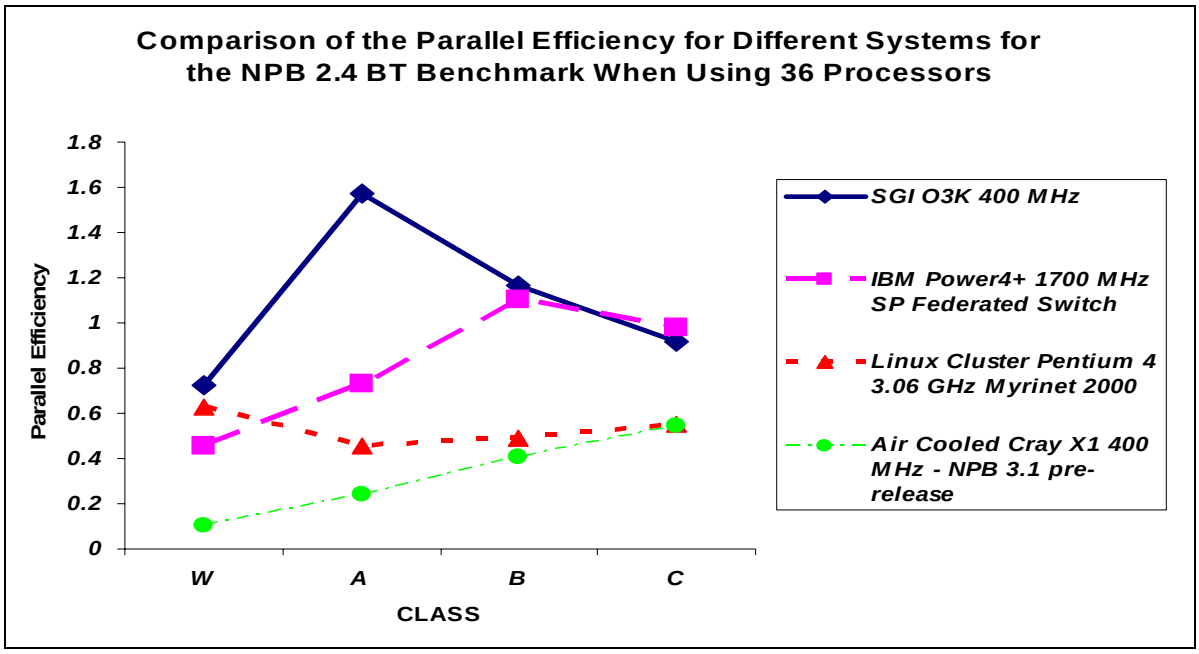

Figure 11. Comparison of the parallel efficiency for different systems for the NPB 2.4 BT benchmark when using 36 processors.

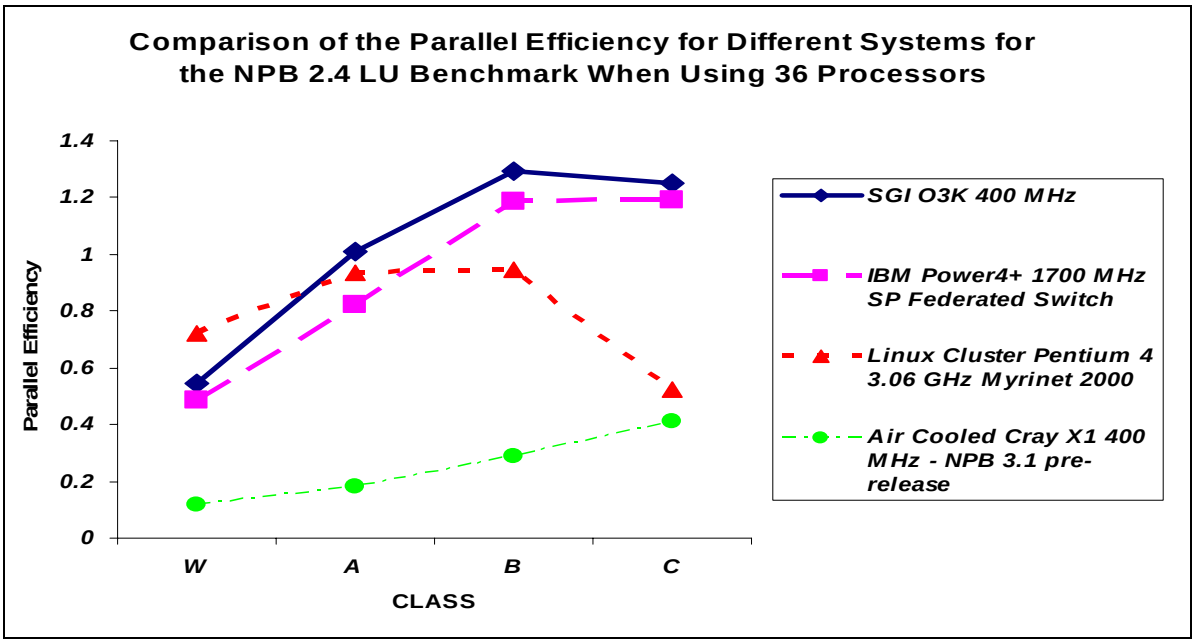

Figure 12. Comparison of the parallel efficiency for different systems for the NPB 2.4 LU benchmark when using 32 processors.

much more dependent on the latency of the message passing system. On the Cray X1, the UPC and CAF compilers are able to take advantage of the same low-latency hardware support as used by calls to SHMEM. However, many competing systems are optimized for bandwidth, not latency, and would be poor choices for use with these languages. Additionally, as the processor speeds increase, even a low-latency interface may appear to be expensive when measured in much more dependent on the latency of the message passing system. On the Cray X1, the UPC terms of missed opportunities to start floating point operations. Therefore, the long-term viability of these languages is uncertain. Additionally, since it is difficult to vectorize C code, running UPC on the Cray X1 would appear to be a questionable proposition.

For some of the benchmarks when using NPB-MPI (CG, MG, and SP), MSP mode was the best choice. For the remaining five benchmarks, SSP mode was a strong winner. We believe that, regardless of which mode one chooses to use, it is best to use the equivalent number of SSPs as the processor count when comparing performance with other systems on a per-processor basis. It should be noted that this approach to reporting performance will in no way affect efforts to report peak performance for a code, or related metrics.

As was seen when the NPB-MPI version 3.1 (pre-release) BT benchmark was run, tuning for a new class of computer architecture can significantly affect performance. It is likely that additional tuning of the BT, CG, EP, and IS benchmarks could improve their performance on the Cray X1. In some cases, this tuning might hurt the performance on nonvector systems, in which case one might need to consider the desirability of supporting alternative versions of the code. In other cases, it is likely that a single code base could be maintained, but that provisions for additional parameters in either the make.def file or one of the include files would be required to support customization of the implementation for the widest possible range of hardware. Recommendations for some of these changes have been passed on to Haoqiang Henry Jin of the NAS group at NASA Ames Research Center.

8. References

1. The NAS Benchmark (NPB) home page. http://www.nas.nasa.gov.
2. Pressel, D. M.; Jelani, C. *Benchmarking the Benchmarks*; ARL-TR-2805; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, September 2002.
3. Dongara, J. Linpack Benchmark-Parallel Table for the Linpack Benchmark. Published electronically at http://www.netlib.org.
4. McCalpin, J. Equivalent MFLOPS Table for the STREAM Benchmark. Published electronically at http://www.cs.virginia.edu/stream.
5. El-Ghazawi, T. Private correspondence dated 18 August 2005.
6. Bell, C.; et al. Evaluating Support for Global Address Space Languages on the Cray X1. Published in the conference proceedings for ICS'04, Malo, France, 26 June–1 July 2004.
7. Dunigan, T.; et al. ORNL Cary X1 Evaluation (Dunigan). Published electronically at http://www.csm.ornl.gov/~dunigan/cray, 21 April 2005.
8. El-Ghazawi, T. A.; et al. « Evaluation of UPC on the Cray X1 ». Published in the Proceedings of the 2005 Cray Users Group Meeting, May 2005.
9. Saini, S.; Bailey, D. The NAS Parallel Benchmark Results 11-96, Version l.0, Report NAS-96-018, November 1996. Published electronically at http://www.nas.nasa.gov/news/techreports/1996/PDF/nas-96-018.pdf.
10. Faulkner, S. A. Performance of the NAS Parallel Benchmarks on an Origin3000 System. Published electronically at http://people.nas.nasa.gov/~faulkner/o3k_npb_benchmarks.html.

NO. OF
COPIES ORGANIZATION

1 (PDF ONLY) DEFENSE TECHNICAL INFORMATION CTR
DTIC OCA
8725 JOHN J KINGMAN RD
STE 0944
FORT BELVOIR VA 22060-6218

1 US ARMY RSRCH DEV & ENGRG CMD
SYSTEMS OF SYSTEMS INTEGRATION
AMSRD SS T
6000 6TH ST STE 100
FORT BELVOIR VA 22060-5608

1 DIRECTOR
US ARMY RESEARCH LAB
IMNE ALC IMS
2800 POWDER MILL RD
ADELPHI MD 20783-1197

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRD ARL CI OK TL
2800 POWDER MILL RD
ADELPHI MD 20783-1197

ABERDEEN PROVING GROUND

1 DIR USARL
AMSRD ARL CI OK TP (BLDG 4600)

NO. OF COPIES ORGANIZATION

1 PROG DIR
C HENRY
1010 N GLEBE RD
STE 510
ARLINGTON VA 22201

1 DEP PROG DIR
L DAVIS
1010 N GLEBE RD
STE 510
ARLINGTON VA 22201

1 DEP PROG DIR
S SCHNELLER
1010 N GLEBE RD
STE 510
ARLINGTON VA 22201

1 HPC CTRS PROJ MGR
J BAIRD
1010 N GLEBE RD
STE 510
ARLINGTON VA 22201

1 CHSSI PROJ MGR
L PERKINS
1010 N GLEBE RD
STE 510
ARLINGTON VA 22201

1 NAVAL RSRCH LAB
J OSBURN
CODE 5594
BLDG A49 RM 15
WASHINGTON DC 20375-5340

1 NAVAL RSRCH LAB
R RAMAMURTI
CODE 6410
OVERLOOK AVE SW
WASHINGTON DC 20375-5344

1 NAVAL RSRCH LAB
J MCCAFFREY
HEAD OCEAN DYNAMICS
PREDICTION BR
CODE 7320
STENNIS SPACE CTR MS 39529

NO. OF COPIES ORGANIZATION

2 US AIR FORCE WRIGHT LAB
J SHANG
WL FIM
2645 FIFTH ST
STE 6
WRIGHT PATTERSON AFB OH
45433-7912

1 USAF PHILIPS LAB
S WIERSCHKE
OLAC PL/RKFE
10 E SATURN BLVD
EDWARDS AFB CA 93524-7680

1 NVL RSRCH LAB
D PAPCONSTANTOPOULOS
WASHINGTON DC 20375-5000

1 AIR FORCE RSRCH LAB/DEHE
R PETERKIN
3550 ABERDEEN AVE SE
KIRTLAND AFB NM 87117-5776

1 NAVAL RSRCH LAB
G HEBURN
RSCH OCEANOGRAPHER CNMOC
BLDG A491020 RM 178
STENNIS SPACE CTR MS 39529

1 AIR FORCE RSRCH LAB
INFORMATION DIR
R LINDERMAN
26 ELECTRONIC PKWY
ROME NY 13441-4514

1 SPAWARSYSCEN (D4402)
R WASILAUSKY
BLDG 33 RM 0071A
53560 HULL ST
SAN DIEGO CA 92152-5001

1 USAE WATERWAYS
EXPERIMENT ST
J HOLLAND
CEWES HV C
3909 HALLS FERRY RD
VICKSBURG MS 39180-6199

NO. OF
COPIES ORGANIZATION

1 US ARMY CRD&ED
B PERLMAN
AMSEL RD C2
FT MONMOUTH NJ 07703

1 SPACE & NVL WRFR SYS CTR
K BROMLEY
CODE D7305
BLDG 606 RM 325
53140 SYSTEMS ST
SAN DIEGO CA 92152-5001

1 US ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
B BRYAN
1200 WASHINGTON AVE
S MINNEAPOLIS MN 55415

1 NAVAL CMD CNTRL &
OCEAN SURVEILLANCE CTR
L PARNELL
HPC COORDINATOR 7 DIR
NCCOSC RDTE DIV D3603
49590 LASSING RD
SAN DIEGO CA 92152-6148

1 ASSOCIATE DIR
S MOORE
INNOVATIVE CMPTG LAB
CMPTR SCI DEPT
1122 VOLUNTEER BLVD STE 203
KNOXVILLE TN 37996-3450

ABERDEEN PROVING GROUND

11 DIR USARL
AMSRD ARL CI HC
J CLARKE
P CHUNG
J GOWENS
B HENZ
R NAMBURU
C NIETUBICZ
D PRESSEL
D SHIRES
R VALISETTY
C ZOLTANI
S BOGGAN