New Implementations and Results for the NAS Parallel Benchmarks 2

William Saphir * Rob Van der Wijngaart† Alex Woo‡ Maurice Yarrow§
npb@nas.nasa.gov

Abstract

We present new implementations and results for the NAS Parallel Benchmarks 2 suite. The suite currently consists of seven programs. Of these LU, SP, BT, MG and FT have previously been released. Here we describe implementations of EP and IS, as well as a rewritten version of FT that corrects some problems with the original release. Performance results are shown for several architectures: IBM SP Wide Nodes / 66 MHz, IBM SP Thin Nodes 2/ 120 MHz, SGI Origin 2000, Hitachi SR 2201, Sun Ultra Enterprise 4000, Cray J90, Cray T3D, HP Exemplar-S.

1 Introduction

The NAS Parallel Benchmarks (NPBs)[1, 2] are a widely-recognized suite of benchmarks originally designed to compare the performance of highly parallel computers with that of traditional supercomputers. The NPBs are specified algorithmically, and are implemented mainly by computer vendors, using techniques and optimizations appropriate for their products. Performance results based on these optimized proprietary implementations—referred to as NPB 1—are submitted to NAS by vendors and are reported in a periodic NAS Technical Report[4].

In late 1995, NAS announced NPB 2[3], a set of specific NPB implementations, based on Fortran 77 and MPI[5]. These codes are intended to be run with little or no tuning, in contrast with NPB 1 codes, which have been highly optimized for specific architectures. NPB 2 targets computers with hierarchical cache-based memories.

Unlike the proprietary NPB 1 codes, the source for NPB 2 implementations is freely available. This promotes the collection of data on a wide variety of machines and configurations. Another advantage is that the techniques used in NPB 2 implementations can be studied for possible use in other codes.

Because they have not been optimized by vendors, NPB 2 implementations approximate the performance a typical user can expect from a portable parallel program on a distributed memory parallel computer. Together the results presented here provide a well-calibrated comparison of the real-world performance of several parallel computers. NPB 2 results complement, rather than replace, NPB 1 results.

*National Energy Research Scientific Computing Center (NERSC), Berkeley, CA. Supported through Department of Energy contract DE-AC03-76SF00098.

†MRJ Technology Solutions, Moffett Field, CA. Supported through NASA Contract NAS 2-14303.

‡NASA Ames Research Center, Moffett Field, CA.

§Sterling Software, Palo Alto, CA. Supported through NASA Contract NAS 2-13210.

NPB 2.1, released in February 1996, contained implementations for 5 of the 8 original NAS benchmarks: the 3 pseudo-applications LU, SP and BT, and the kernels MG and FT. NPB 2.2 adds IS and EP, as well as an improved FT.

2 Methodology

NPB 2 presents a number of challenges and opportunities that were not present in NPB 1. NPB 1 implementations represent a best case scenario: they are implemented and carefully optimized by vendors; they are timed under ideal conditions in clean environments with the latest software. NPB 1 results are meaningful and can be compared because every vendor has had equal opportunity to optimize, and because the pencil-and-paper nature of the benchmarks eliminates architectural bias. NPB 1 results, while far more meaningful than the "not to be exceeded" peak performance or LINPACK[6] rates, generally represent upper limits on what can be achieved for the type of calculations performed by the NPBs.

In contrast, NPB 2 implementations are run essentially without tuning[1]. Because the codes are not proprietary and sources are readily available and easy to build, anyone with a sufficiently large computer, MPI, and a Fortran compiler can generate NPB 2 results. As with any scientific measurement, benchmark results are useless unless reproducible. Therefore great care must be taken to document the specific conditions under which measurements were made: compiler options, MPI and compiler versions, system software version, etc. Data qualified in this way can provide a well-calibrated measurement of the effect of small hardware and software changes, showing the improvement or regression of system performance over time.

Most [2] NPB 2 implementations discussed in this paper run a single solver iteration and reset the initial conditions before starting the timed portion of the benchmark. This eliminates startup overhead that can severely skew results for short-running benchmarks. It also means that Mflop/s rates reported by the code will not be the same as Mflop/s rates reported by a hardware instruction counter. Care must be taken to reconcile directly-measured operation counts with the rates reported by the codes themselves. This is accomplished by directly measuring the operation count of a zero-iteration run and subtracting this from the operation count of a complete run. Time reported by the benchmarks and in this report is wall clock time. CPU time may not be meaningful on a tightly-coupled parallel system where time spent blocking for messages is not counted as CPU time.

3 Applicability and usefulness of results

VECTOR VERSUS CACHE — NPB 2 implementations are designed for cache-based architectures, and use a distributed memory software model, with communication only through MPI. While it is uncommon to run MPI codes on shared memory multiprocessors, we believe NPB 2 results reflect their performance, at least to the extent that portable MPI applications are run in shared memory environments. For vector machines such as the Cray J90, however, the results do not have an immediate interpretation as typical "real-world" application performance. Instead, they indicate opportunity for vectorization in these cache-oriented codes.

[1]NPB 2 actually permits two levels of tuning: 0% code modification and 5% code modification. This report contains results only for the 0% case

[2]All for LU, which will not have this feature until NPB 2.3.

In this report we compare results from eight machines. The IBM SP based on 66 MHz Power 2 (SP-WN) and 120 MHz P2SC nodes (SP-SC), the Hitachi SR2201 based on 150 MHz modified PA-RISC processors, and the Cray T3D based on 150 MHz Alpha processors are distributed memory machines. The Sun UltraEnterprise 4000 server based on 167 MHz Ultrasparc-I processors, and the HP Exemplar-S based on the 180 MHz PA-8000 processor are nominally symmetric multiprocessors (although memory is actually distributed), and the SGI Origin 2000 based on 190 MHz R10000 processors is a distributed shared memory multiprocessor. The Cray J90 is a shared-memory vector multiprocessor.

Because of space limitations, we are unable to describe in detail the machines on which our results were obtained. More information can be found at http://www.nas.nasa.gov/NAS/NPB/.

MFLOP/S RATES — A new feature of NPB 2 is that we report results in Mflop/s (million floating point operations per second) as well as run time. The Mflop/s measure has little meaning for NPB 1, since different implementations may have substantially different operation counts. Since NPB 2 results are based on unmodified code (0% case), the operation count is approximately the same on any machine. Optimizing compilers may be able to eliminate some operations, or may add some for certain reasons, so the reported Mflop/s rate may not be exactly what would be reported by a hardware instruction counter, but it is usually within a few percent. For the EP and IS benchmarks, in which the bulk of computation is integer arithmetic, Mflop/s rates are not meaningful. These benchmarks compute a Mop/s rate, where "op" is random-numbers-generated and keys ranked, respectively.

The nominal rate is based on a combination of hand counting, Cray and IBM hardware instruction counting, SGI "prof" and "pixie" analysis, and extrapolation. The nominal floating point count does not change with number of processors—only with problem size—so extra operations performed because of parallelization are not counted.

COMPARISON WITH NPB 1 RESULTS — When both NPB 2 and NPB 1 results are available for the "same" machine, we compare NPB 2 results with NPB 1 results. The comparison gives an indication of the difference between what an "ordinary programmer" can achieve on a portable program and what an expert or unusually careful non-expert can achieve on an architecture-optimized program. If NPB 1 and NPB 2 timings are close, one might conclude that it is relatively easy to get NPB-1-level sustained performance out of a particular machine. Different machines have substantially different results for this metric.

RELEVANCE FOR OTHER CODES — The NPB suite is based on computational fluid dynamics codes typical of those run on the supercomputers at the NAS facility. We believe that the NPB 2 results are relevant to "the average user, with an average code," with a few caveats.

First, the NPB 2 implementations do not use numerical libraries. Codes that use vendor-optimized numerical libraries may obtain substantially better performance. Some library routines that could be useful for NPB implementations are FFT routines for the FT benchmark, and banded-(block-)matrix solvers for the SP and BT benchmarks. NPB 2 implementations do not use these because there do not exist standardized, vendor-optimized and widely available versions of the necessary routines (unlike the BLAS routines).

Second, most NPB codes use fully implicit algorithms on a single grid that is distributed over several processors. Such algorithms require more communication than explicit methods or hybrid methods that solve implicitly on a single processor with explicit updates between processors. Compared to codes using these other algorithms, the NPB suite provides a more stringent test of interprocessor communication, and may show poorer scalability. Counteracting this is the fact that the NPB 2 codes use good parallel algorithms that minimize communication.

Third, NPB problems are idealized versions of real-world problems. They are more regular than typical real-world problems (making load balancing easier) and boundary conditions and physics have been somewhat simplified. The effects on performance are complicated. For instance, load-imbalance from boundary calculations is probably smaller than in engineering applications, but having fewer operations per grid point per time step (because of simpler physics) gives a higher communication to computation ratio and lower cache reuse.

For these reasons and others, performance of real-world codes may or may not be similar to NPB performance. The main use of NPB is, instead, to compare the performance of different computers on the general class of codes represented by the NPBs.

4 Code descriptions

The NPB 2 implementations of LU, SP, BT and MG have been discussed elsewhere[3]. Here we describe the new implementations of EP and IS, and the reimplementation of FT.

EP — The NPB 2 implementation of the EP (Embarrassingly Parallel) benchmark has its origins in the NAS-provided reference code from 1991. The current implementation can run on any number of processors. Each processor independently generates pseudorandom numbers (PNs) in batches of 2^{17} and uses these to compute pairs of normally-distributed numbers. The rounded norms of these pairs are tallied before the next batch is created and processed. No communication is needed until the very end, when the tallies of all processors are combined. Performance of the code, which is reported as the number of PNs produced per second, is governed by the PN generator (see below). The calculation also contains a significant number of logarithm and square root operations.

IS — The NPB 2 implementation of the IS kernel benchmark is based on a bucket sort. The number of keys ranked, number of processors used, and number of buckets employed are all presumed to be powers of two. This simplifies the coding effort and leads to a compact program. The number of buckets is a tuning parameter. On the systems tested, best performance was obtained when the number of buckets was half that which gives best load balancing. Communication costs are dominated by an MPI_Alltoallv, wherein each processor sends to all others those keys which fall in the key range of the recipient.

FT — The FT implementation in the NPB 2.0 release was found to have some problems with scaling and performance. In NPB 2.2, it is replaced by a new version, described here. All FT results reported in this paper and previously reported by NAS are for this new version. The computational core of the benchmark is a 3-D Fast Fourier Transform (FFT) on a grid whose dimensions are $n_1 \times n_2 \times n_3$ points. On one processor, the 3D FFT is performed with three successive 1D FFTs. For a number of processors m with $m \leq n_3$, each processor owns a number of contiguous x–y planes. The 1D FFTs in the x and y directions are performed in-processor, after which the data is transposed (using MPI_Alltoall) so that the z-direction FFT can be performed in-processor. For $m > n_3$, a two-dimensional decomposition is used, leaving only the x-direction in-processor. Two transposes are performed in order to do the y- and z-direction FFTs. The new implementation of FT also contains performance improvements in the calculation of the time evolution. The basic 1D FFT routine is essentially unchanged in the new implementation.

PSEUDORANDOM NUMBER GENERATION — In FT and EP the generation of random numbers is timed, so that it is important that this operation be efficient. It is essentially an integer calculation that can be done by machines that support 64-bit integer arithmetic, but it should be possible to run the code on other machines as well. Accordingly, this release of the benchmarks includes several different modules for random number generation. The first, *randi8*, uses 8-byte integer arithmetic. It relies on overflow bits being discarded and works on most 64-bit machines. The second, *randi8_safe* uses bit manipulations on 64-bit integers and guarantees that there is no overflow. The third, *randdp*, uses 64-bit floating point arithmetic to simulate the integer operations. The fourth, *randdpvec*, is similar, but allows vectorization.

5 Results

NPB 2 data can be used to examine a number of performance issues. Here we highlight a few of the more interesting results, using a representative sample from the complete set of NPB 2 results available at http://www.nas.nasa.gov/NAS/NPB/NPB2Results/.

PERFORMANCE PER PROCESSOR — Figure 1 shows the per-processor performance of BT

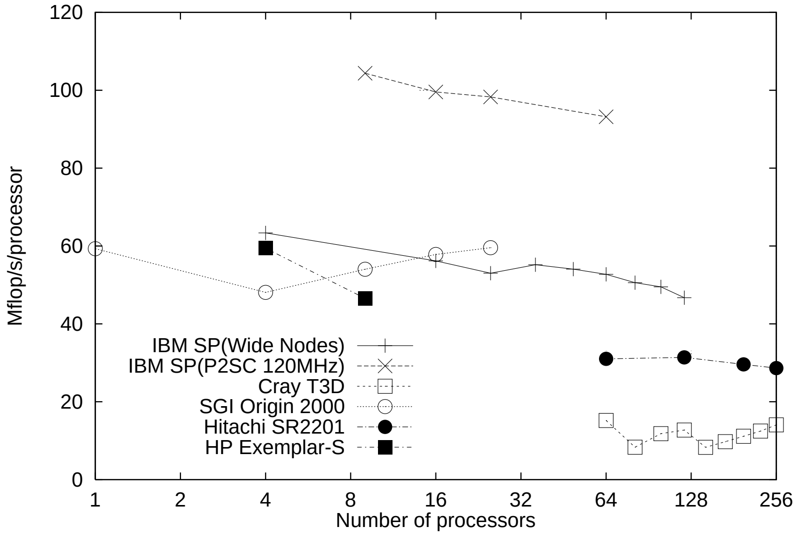

FIG. 1. *Per-processor Performance for BT Class B*

class A on all machines. It shows a large disparity in per-processor performance between the six machines. The SP-SC is the clear leader, obtaining as much as 105 Mflop/s/processor. The midrange is occupied by the Origin, Exemplar and SP-WN machines. Hitachi and T3D perform poorest, with between 15 and 30 Mflop/s/processor. This ranking is fairly consistent across the NPBs, although the Origin approaches SP-SC performance on several of the other benchmarks.

We partly attribute the dominance of the SP-SC to its high memory bandwidth. The other machines (except SP-WN, which has a slower clock than SP-SC) have significantly poorer memory bandwidth relative to peak performance.

We also note that performance of the SP-WN, SP-SC and Exemplar fall off rapidly with number of processors. The inter-node communication performance of the SP systems is relatively poor compared to their floating point performance. Fairly level lines for Hitachi and the T3D up to 256 processors indicate good relative network performance. [3]

Total performance — Figure 2 shows total machine performance for LU class B. The

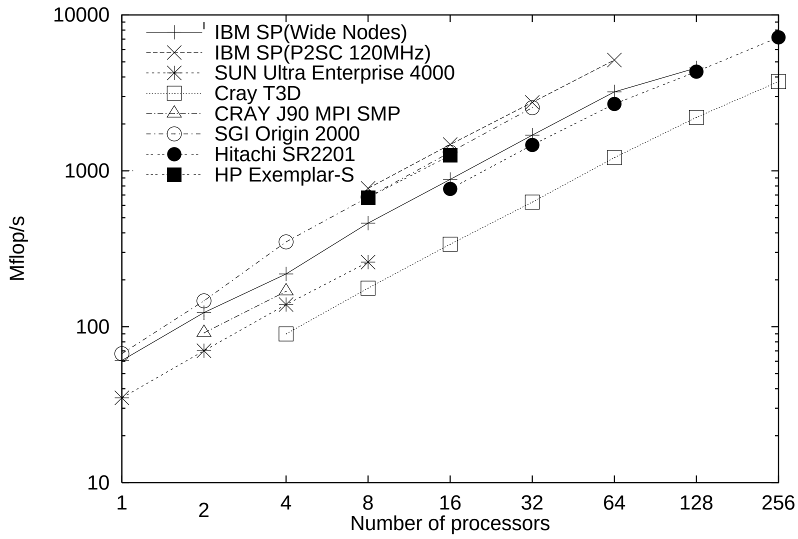

Fig. 2. *Total Machine Performance for LU Class B*

information content is similar to that in Figure 1, except that it is easier to judge total machine performance. Among the machines for which we have results, the Hitachi has the highest performance, even though its performance per node is fairly poor. Good scalability and large system size make up for that.

Efficiency — Figure 3 shows the machine efficiency on MG class B. Efficiency is total

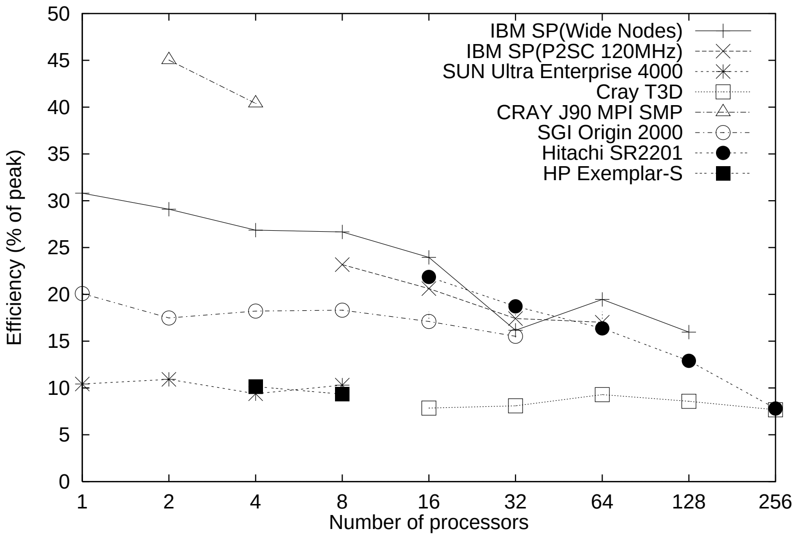

Fig. 3. *Machine Efficiency for MG Class B*

[3]The jagedness of the T3D line is an artifact of power-of-two processor allocation on the T3D, and does not reflect network performance.

Mflop/s divided by the peak Mflop/s, expressed as a percentage.

The most interesting point of this graph is that it shows that the SP-SC and Origin obtain their high per-node performance not only through a high peak performance, but by also attaining a relatively large fraction of it. This result emphasizes the inappropriateness of relying on peak performance to judge the capabilities of RISC-based machines. Notice that vector machines (such as the Cray J90) typically attain a much higher percentage of peak performance on appropriate, well-written codes than do cache-based machines.

COMPARISON OF NPB 2 AND NPB 1 PERFORMANCE — It can be insightful to compare NPB 2 and NPB 1 results. While NPB 2 results can be reported in Mflop/s/processor, NPB 1 implementations may have different operation counts. Instead of Mflop/s/processor, we plot a the ratio of NPB 2 run time to the NPB 1 run time. Because NPB 1 and NPB 2 data are not always available on the same numbers of processors, we calculate a ratio based on a curve fit to the NPB 2 results. Figure 4 shows the NPB 2 to NPB 1 run time ratio

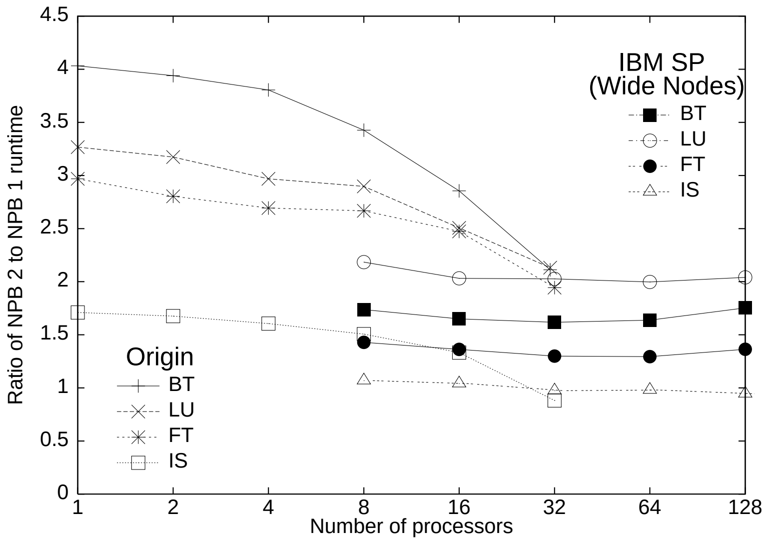

FIG. 4. *NPB 2/1 run time ratio for Origin and SP-WN*

for the BT, LU, FT and IS benchmarks on the Origin and SP-WN. Higher numbers mean that NPB 1 performs better relative to NPB 2. Positive slopes show NPB 1 scaling better, and negative slopes show NPB 2 scaling better.

Interpreting NPB 1 performance as the "peak achievable" performance on a particular machine, and NPB 2 performance as "normal" performance of a portable code, the data suggest that the SP-WN design makes it relatively easy to obtain a high percentage of peak performance. On the Origin they suggest that is much harder to achieve good performance, but careful architecture-specific tuning can improve the situation considerably. A confounding factor is that the NPB 1 Origin implementations use a shared memory model with automatic compiler parallelization.

It is also clear from Figure 4 that NPB 2 codes scale about as well as the IBM implementations on the SP-WN, but that NPB 2 codes scale much better than Cray/SGI implementations on the Origin.

6 Analysis/Conclusions

NPB 2 results show that there has been a recent improvement of the quality of RISC processors for scientific computing. Whereas actual attainable performance used to be about 10% of the peak speed of the processor, the IBM RS/6000 and SGI R10000 routinely obtain more than 20% of peak speed.

The combined results of NPB 1 and NPB 2 provide insight in the attainable performance of modern computer architectures on significant scientific applications, both by the general, careful user, and the expert vendor programmer. The different benchmarks in the NPB 2 suite stress different aspects of the hardware and software in the computer system. For example, IS and FT require efficient implementation of the all-to-all communication routines, and need large network bandwidth to accommodate large data transfers. SP has poor data reuse, unlike BT and LU which spend a major fraction of their execution time inverting dense 5×5 matrices, so it requires high memory bandwidth. BT and LU, on the other hand, stress instruction cache and register use, due to the sizeable loops in which most of the work is performed. Potential buyers can now rate machines on these different performance aspects in an objective fashion using the fixed-source-code format of NPB 2.

The current release of the benchmark codes, NPB 2.2, targets distributed-memory machine models with explicit message-passing communications. This design does not fully test shared-memory or vector architectures. We plan to address these limitations in future work.

Acknowledgments

The NAS parallel systems group was very helpful in many stages of collecting the NPB 2 results. Lou Zechtzer, Archie Deguzman, James Jones, and Mary Hultquist went beyond the call of duty to help out when there were problems or special requests. We thank Barbara Horner-Miller for providing access to the JPL CRI T3D machine and allowing us to greatly exceed our CPU quota. The Sun numbers were obtained on the COMPS cluster at NERSC.

References

[1] Bailey, D. H., *et al.*: The NAS Parallel Benchmarks, *International Journal of Supercomputer Applications*, Vol. 5, No. 3, (Fall 1991), pp. 63-73.

[2] Bailey, D. H.; Barton, J.T.; Lasinski, T. A.; and Simon, H. D., eds: "The NAS Parallel Benchmarks," *NASA Technical Memorandum 103863*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, July 1993 http://www.nas.nasa.gov/NAS/NPB/.

[3] Bailey, D. H.; Harris, T.; Saphir, W.; van der Wijngaart, R.; Woo, A.; and Yarrow, M., "The NAS Parallel Benchmarks 2.0," *NASA Technical Report NAS-95-020*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, December 1995 http://www.nas.nasa.gov/NAS/NPB/.

[4] Bailey, D. H.; and Saini, S., "The NAS Parallel Benchmarks Results 12-95," *NASA Technical Report NAS-95-021*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, December 1995 http://www.nas.nasa.gov/NAS/NPB/.

[5] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 1.1, July, 1995, http://www.mcs.anl.gov/mpi/mpi-report-1.1/mpi-report.html.

[6] Dongarra, J. J.: The LINPACK Benchmark: An Explanation. SuperComputing, Spring 1988, pp. 10-14.