The NAS Parallel Benchmarks 2.0

David Bailey[*], Tim Harris[†], William Saphir[†],
Rob van der Wijngaart[‡], Alex Woo[*], and Maurice Yarrow[§]

Report NAS-95-020, December, 1995

Abstract

We describe a set of implementations of the NAS Parallel Benchmarks based on Fortran 77 and the MPI message passing standard. These implementations, which are intended to be run with little or no tuning, approximate the performance a typical user can expect for a portable parallel program on a distributed memory computer. They complement rather than replace the original NAS Parallel Benchmarks.

We also present two additions to the original pencil and paper specification. First, we define "class C" sizes of the benchmarks to better suit the current and next generation of supercomputers. Second, we introduce changes to the reporting requirements for NAS Parallel Benchmark results.

[*] NASA Ames Research Center, Moffett Field, CA, 94035-1000.

[†] MRJ, Inc. This work is supported through NASA Contract NAS 2-14303.

[‡] MCAT, Inc. This work is supported through NASA Contract NAS 2-14109.

[§] Sterling Software, Palo Alto, California. This work is supported through NASA Contract NAS 2-13210.

Contents

1 Introduction

The Numerical Aerodynamic Simulation (NAS) program, located at NASA Ames Research Center, is a pathfinder in high-performance computing for NASA, focusing on computational fluid dynamics and related aeroscience disciplines [1]. A key goal of the NAS program is to demonstrate by the next millennium an operational computing system capable of simulating, in one to several hours, an entire aerospace vehicle system throughout its mission and life cycle. To solve this full multidisciplinary problem requires a system that can perform scientific computations at a sustained rate one to two orders of magnitude faster than current supercomputers. We expect that such a computer system will employ hundreds or thousands of powerful processors operating concurrently.

To measure objectively the performance of highly parallel computers and to compare their performance with that of conventional supercomputers, NAS developed the NAS Parallel Benchmarks (NPB 1.0) in 1991[5]. The benchmarks, which are derived from computational fluid dynamics codes, have gained wide acceptance as a standard indicator of supercomputer performance. The "pencil and paper" design of NPB 1.0 acknowledged fundamental barriers to portability among parallel computers.

While the NAS Parallel Benchmarks continue to provide an important measure of parallel performance, a number of weaknesses have become evident since their release. Implementations of the NAS Benchmarks are usually highly tuned by computer vendors, so that the performance of these implementations is difficult for scientific programmers to obtain. Moreover, these tuned implementations are generally proprietary and not publicly available, so that the techniques used by vendors to obtain high performance remain hidden.

Another sign of the age of NPB 1.0 is that the largest problems (class B) no longer reflect the largest problems being done on present-day supercomputers.

In the current work, we introduce

- source-code versions for five of the benchmarks, intended to be run with little tuning. They supplement, rather than replace, NPB 1.0.
- class "C" sizes for all benchmarks.
- new reporting requirements for NPB 1.0 results.

The bulk of this report concerns the new source code implementations.

Note that the NPB effort is distinct from the NAS High Speed Processor (HSP) benchmarks and procurements. The HSP benchmarks are used for evaluating production supercomputers for procurements in the NAS organization, whereas NPB 1.0 and 2.0 are used for the study of highly parallel processor systems in general.

1.1 NAS Parallel Benchmarks 1.0

NPB 1.0 consists of eight benchmark problems derived from important classes of aerophysics applications. The eight problems consist of five kernels and three simulated computational fluid dynamics (CFD) applications. The five kernels mimic the computational core of five numerical methods used by CFD applications. The simulated CFD applications reproduce much of the data movement and computation found in full CFD codes, and require more effort to implement than the kernels. For a detailed description and specification of NPB 1.0, see the 1991 technical report [5, 6].

The benchmarks are unique in their “pencil and paper” specification – vendors implement the detailed specifications in the NPB report using algorithms and programming models appropriate for their different machines. Within the framework of Fortran 77, Fortran 90, C and HPF, implementors are free to use language constructs, data structures, data partitioning and algorithmic details that maximize performance on a particular system. Because of this freedom, and because performance on NPB 1.0 has been used in procurements, vendors have devoted extensive effort to optimizing their implementations. While source code for these implementations is generally proprietary, several of the implementations are documented in publicly available reports [7, 8, 9]. Results submitted by vendors have been summarized in a periodically updated NAS technical report [10, 11, 12].

NAS has provided reference implementations of NPB 1.0 but these are neither portable nor optimized. The reference implementations are intended *only* as a starting point for optimized implementations.

2 Source Code Implementations - NPB 2.0

At the time NPB 1.0 was developed, the wide diversity of architectures and programming models made it impossible to write portable parallel benchmark implementations. For instance, it was not possible to write a single program that could give meaningful results for both a CM-2 (a SIMD computer programmed in a data parallel language) and an iPSC/860 (a MIMD machine programmed with message passing).

Although important barriers to portability still exist, emerging standardization in parallel computing now makes it feasible to consider benchmarks specified by source code. On the software side, there is now an industry standard message passing library, MPI, and a standard data parallel programming language, HPF. On the hardware side, there has been a convergence in parallel architectures to MIMD computers based on commodity processors with cache-based memory systems, and composed of single processor or SMP nodes.

There are several motivations for introducing source code versions of the NAS Parallel Benchmarks. NPB 1.0 gives a reasonable measure of relative performance because the playing field has been leveled by the uniformity with which pre-sales marketing departments have aggressively optimized their hardware-specific implementations. NPB 1.0 is a poorer indication of real-world performance. While many codes run at NAS are highly tuned for the specific architecture on which they execute, an increasing number are written with portability in mind, especially in an environment of rapid machine turnover. Well-written but untuned benchmark codes may give a better indication of performance of these new portable codes. Finally, a comparison of source code benchmarks and NPB 1.0 results may indicate how much performance can be improved through careful tuning.

NPB 2.0 is primarily a source code release of a subset of the original benchmarks. It is an interim release of what will eventually be NPB 3.0, which will also contain some new benchmarks. The NPB 2.0 benchmarks are implemented in Fortran 77 with a few common extensions that are also part of Fortran 90. They use MPI for communication but can be run on one processor with a "dummy" MPI library that performs negligible work.

A number of NPB implementations have been produced by research projects [13, 14, 15, 16, 17, 18]. What distinguishes NPB 2.0 from other publicly available source-code implementations of NPB 1.0? First, NPB 2.0 codes were designed as benchmarks, and are not a byproduct of a related research project. They are carefully coded to use modern algorithms, avoid unnecessary computation, and generally represent real-world codes. They perform reasonably well across many platforms. Other NPB implementations (including the NPB 1.0 reference implementations from NAS) often perform more poorly, so that they don't predict performance of well-written production applications and cannot be meaningfully compared to vendors' optimized NPB 1.0 implementations. Second, NPB 2.0 is one of a few implementations to utilize MPI, which we believe is an essential component of standard parallel benchmarks. What distinguishes NPB 2.0 from other (non-NPB) source-code parallel benchmarks? Its primary advantage is that the NPB suite is already well-known – NPB 2.0 results can be readily compared to vendor-optimized NPB 1.0 results. Additionally, NPB 2.0 codes are relatively small and quite portable.

NPB 2.0 includes five of the original eight benchmark problems – FT, MG, LU, SP, BT. The other benchmarks were not implemented both because they were considered less important and because of manpower limitations.

FT contains the computational kernel of a three dimensional FFT-based spectral method. MG uses a multigrid method to compute the solution of the three-dimensional scalar Poisson equation. LU is a simulated CFD application which uses symmetric successive over-relaxation (SSOR) to solve a block lower triangular-block upper triangular system of equations resulting from an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. SP and BT are simulated CFD applications that solve systems of equations resulting from an approximately factored implicit finite-difference discretization of the Navier-Stokes equations. The BT code solves block-tridiagonal systems of 5x5 blocks; the SP code solves scalar pentadiagonal systems resulting from full diagonalization of the approximately factored scheme. More details of the implementations are described in Section 2.3.

NPB 2.0 source code benchmarks are intended to be run with little or no tuning (see Appendix A). NAS will collect results (see Appendix D) and report them along with NPB 1.0 results.

2.1 Programming Model

There are three types of programs that are portable across a wide range of distributed memory machines – message passing programs based on MPI (Message Passing Interface) or PVM (Parallel Virtual Machine), and data parallel programs based on HPF (High Performance Fortran).

We chose MPI because it is available and in wide use on all machines at NAS, because it is an industry standard, and because it is designed for and generally achieves high performance. NAS believes that MPI will become the de-facto standard message passing library. PVM, while popular, is not appropriate for NPB 2.0. It undergoes major changes frequently, so that vendors rarely have optimized implementations of the latest version and codes require revision to conform. PVM was not designed for high performance, and is more appropriate for loosely coupled, dynamic and fault-tolerant applications not represented by NPB 1.0. HPF is appropriate, but most current HPF compilers are still maturing, making development difficult and performance analysis problematic. We hope to release HPF implementations as part of NPB 3.0.

The most common programming model that we did not consider is automatic compiler parallelization (of C or Fortran) targeting shared memory architectures. We did not consider this model because it is not appropriate for non-shared memory computers and because there is no standard programming interface (e.g. compiler directives).

While the benchmarks are implemented with MPI, they are not intended to test only MPI. They are holistic benchmarks, designed to measure the overall performance of a complex system of which MPI is one part.

Although C and C++ are gaining in popularity for scientific programming, Fortran is still the overwhelming choice at NAS. We chose to implement in Fortran 77 and use only a few common extensions. These include long variable names, the “include” statement, do/enddo constructs, and similar extensions. Fortran 90 would be the forward-looking choice, but there are a several performance issues with current compilers which made a full-blown Fortran 90 implementation infeasible.

2.2 Architecture and Performance Considerations

The NPB 2.0 source code benchmarks can be run on on almost any MIMD parallel computer supporting Fortran and MPI. Complete portability combined with maximum performance is an impossible goal, however. We discuss here some of the architectural considerations and performance tradeoffs.

We target distributed-memory MIMD computers primarily for portability. Codes which run efficiently on distributed memory systems should be able to run well on a shared-memory architecture. The converse, however, is not necessarily true. Moreover, because of inherent scalability issues in shared memory architectures, high-end supercomputing will have to address the problems of distributed memory for the foreseeable future.

Portability was also a concern at the node level. Many highly parallel supercomputers are based on RISC or RISC-like processors with cache-based hierarchical memory systems [19]* To utilize cache effectively, the benchmarks generally access data with stride one. They are not optimized for long vector lengths, or for interleaved memory systems.

The benchmarks are not “unnaturally” optimized. There is no loop unrolling or careful ordering of operations to utilize efficiently a specific instruction set.

2.3 Benchmark Descriptions

The NPB 2.0 implementations contain the following features.

Self-Verification is contained within the code to determine if each run has completed with the correct results.

Timing is performed according to the NPB 1.0 specifications. When possible, the code is run for one time step and then reinitialized before timing begins. The purpose of this apparently gratuitous iteration is to eliminate startup costs associated with demand paging and cache loading by making sure that all code and data has been touched. A real application may run for much longer than an NPB, so that these startup costs are amortized over a long period of time.

Mflop/s rates are estimated within the code. These estimates are based on actual operation counts without compiler optimizations and were made with *pixie* on an SGI Challenge or *hpm* on a Cray Research C90. The operation counts apply only to NPB 2.0 implementations, not to NPB 1.0 implementations, and are different from the counts that can be inferred from the original NPB report. The original report contained Mflop/s rates and operation counts measured for specific CRI Y-MP implementations. Those original numbers should not be used to derive Mflop/s rates either for NPB 1.0 or for NPB 2.0 implementations.

*See also http://www.netlib.org/benchmark/top500.ps.

The benchmarks must be compiled for a specific grid size and number of processors. While some of the codes can be run successfully on a larger number of processors or smaller grid than those specified at compile time, memory access behavior may be different from that of a code compiled explicitly for that size and number of processors. Since reproducibility is an important goal in any benchmark, NPB 2.0 results are valid only for configurations identical to those specified at compile time. For instance, although the MG benchmark compiled for class A and 16 processors can be run successfully on 32 processors, the 32 processor result will not be included in the NPB 2.0 database.

2.3.1 Kernel Benchmark: FT

The implementation of the 3-D FFT PDE benchmark follows a fairly standard scheme. The 3-D array of data is distributed according to z-planes of the array – one or more planes are stored in each processor. The forward 3-D FFT is then performed as multiple 1-D FFTs in each dimension, first in the x- and y- dimensions, which can be done entirely within a single processor, with no interprocessor communication. An array transposition is then be performed, which amounts to an all-to-all exchange, wherein each processor must send parts of its data to every other processor. The final set of 1-D FFTs is then performed. A conventional Stockham-transpose-Stockham scheme is used for the 1-D complex FFTs. This procedure is reversed for inverse 3-D FFTs. FT runs on a power-of-two number of processors.

2.3.2 Kernel Benchmark: MG

The Multigrid Benchmark is based on the NX reference implementation from 1991. Four critical subroutines – the smoother, $psinv$, the residual calculation, $resid$, the residual projection, $rprj3$ and the trilinear interpolation of the correction, $interp$ – were optimized for both vector and RISC processors. This code requires a power-of-two number of processors. The partitioning of the grid onto processors occurs such that the grid is successively halved, starting with the z dimension, then the y dimension and then the x dimension, and repeating until all power-of-two processors are assigned.

2.3.3 Application Benchmark: LU

The LU benchmark is based on the NX reference implementation from 1991. This code requires a power-of-two number of processors. A 2-D partitioning of the grid onto processors occurs by halving the grid repeatedly in the first two dimensions, alternately x and then y, until all power-of-two processors are assigned, resulting in vertical pencil-like grid partitions on the individual processors. The ordering of point based operations constituting the SSOR procedure proceeds on diagonals which progressively sweep from one corner on a given z plane to the opposite corner of the same z plane, thereupon proceeding to the next z plane. Communication of partition boundary data occurs after completion of computation on all diagonals that contact an adjacent partition. This constitutes a diagonal pipelining method and is called a "wavefront" method by its authors [31]. It results in a relatively large number of small communications of 5 words each.

Although the described algorithm is not optimal for the application at hand, it is being retained as a benchmark because it is very sensitive to the small-message communication performance of an MPI implementation. It is the only benchmark in the NPB 2.0 suite that sends large numbers of very small (40 byte) messages.

2.3.4 Application Benchmarks: SP and BT

The SP and BT algorithms have a similar structure: each solves three sets of uncoupled systems of equations, first in the x, then in the y, and finally in the z direction. These systems are scalar pentadiagonal in the SP code, and block tridiagonal with 5x5 blocks in the BT code.

The NPB 2.0 implementations of SP and BT solve these systems using a multi-partition scheme [20, 21]. We chose the multi-partition approach because it provides good load balance and uses coarse grained communication. Other common partitioning strategies considered were uni-partitioning, combined with Pipelined Gaussian Elimination (PGE) [28] and Dynamic Block-Cartesian Decomposition [28]. Pipelined Gaussian Elimination has asymptotically better scaling properties (for very large numbers of processors) but forces a trade-off between load balance and communication granularity. The dynamic block Cartesian decomposition requires a transpose at each step, making it appropriate only on extremely high bandwidth networks. These three strategies were compared in [28].

In the multi-partition algorithm [20] each processor is responsible for several disjoint sub-blocks of points ("cells") of the grid. The cells are arranged such that for each direction of the line solve phase the cells belonging to a certain processor will be evenly distributed along the direction of solution. This allows each processor to perform useful work throughout a line solve, instead of being forced to wait for the partial solution to a line from another processor before beginning work. Additionally, the information from a cell is not sent to the next processor until all sections of linear equation systems handled in this cell have been solved. Therefore the granularity of communications is kept large and fewer messages are sent.

Both the SP and BT codes require a square number of processors. These codes have been written so that if a given parallel platform only permits a power-of-two number of processors to be assigned to a job, then unneeded processors are deemed inactive and are ignored during computation, but are counted when determining Mflop/s rates.

2.4 Benchmark Rules

The NAS organization will report the original NPB 1.0 results as well as the NPB 2.0 results. The NPB 2.0 results will be reported in three forms

0%: (or dusty deck) where none of the source code is changed (except as required to make the code run) and

5%: where up to 5 % of the lines of source code are modified. White space changes are not counted.

$>$**5%:** where more than 5% of the lines of source code are modified.

To qualify for 0% modification, the submitter can modify or replace the makefile and build scripts provided with the NAS 2.0 distribution. The source can be *fsplit* and compiled with different compiler options or even different compilers. However the Fortran code itself cannot be modified. Automatic preprocessors which convert output Fortran are allowed. (When reporting a *fsplit* result, please report the highest level of optimization used, and which routines were *not* compiled with this level of optimization.)

NAS will also accept submissions based on greater than 5% modifications. These submissions follow the same rules as NPB 1.0, but non-vendor submissions will be reported with the NPB 2.0 results.

The fraction modification is the number of changed lines divided by the total number of lines in the original source. The number of changed lines is defined, for each file, by the number of lines produced by *sdiff* when comparing the original and new versions, or by the number of lines in the file, if an original version does not exist. For instance, the following *csh* script would print out the total number of lines changed for each file:

```
foreach f (*.f *.h *.incl)
  sdiff -s $f $f.orig |wc -l
end.
```

NAS will use its judgment to determine whether modifications which barely exceed 5% are within the intent of the minor modification criteria.

For all submittals, NAS requests the output files, source code (if changed), and auxiliary files used for building (if changed). These will be made publicly available. They are necessary to enable the results to be reproduced. The NAS organization reserves the right to verify any NPB results that are submitted to us. We may attempt to run the submitter's code on another system of the same configuration as that used by the submitter. In those instances where we are unable to reproduce the vendor's supplied results (allowing a

5% tolerance), our policy is to alert the submitter of the discrepancy and allow the submitter to resolve the discrepancy in the next release of this information. If the discrepancy is not resolved to our satisfaction, then our own observed results with NAS explicitly named as the submitter, and not the submitter's results, will be reported. This policy will apply to all results NAS receives and publishes.

For the NPB 2.0 "dusty deck" submittals, the NAS organization will entertain numbers from all sources in a procedure similar to that used for the LINPACK 100 benchmark. For these numbers, the submitter must include the hardware specification, the operating system and compiler version numbers, and the exact compiler flags used to produce the submittal. In addition, vendor submittals must include the cost in U.S. dollars of the benchmarked computer system in order for NAS to compute sustained performance per dollar. These costs should include any associated software costs such as operating systems, compilers, scientific libraries but not substantially more hardware than required for the benchmarks. Non-vendors submissions are not required to include cost. The NAS organization will verify some of these submittals.

For 5% and above 5% modification submittals, NAS requires in addition the output files and the modified source code, makefiles, and run scripts.

NAS will maintain a WWW page[*] and a periodic series of technical reports with the results from these benchmarks. NAS will make its MPI implementation freely available and encourages vendors to either allow NAS to distribute their technical reports on these NAS benchmark implementations or their source code. In addition, the Parkbench effort will incorporate these numbers in their graphical WWW pages [29].

2.5 Benchmarking Technique

Because these benchmarks measure wall clock time and are statically load balanced and tightly synchronized they have to be run on completely dedicated systems, and with as few system processes as possible. Failure to do so may cause timing results to be in error.

2.6 Source Distribution

NAS has received a waiver which allows unlimited, world wide distribution of the NAS Parallel Benchmarks.

[*] See also http://www.nas.nasa.gov/NAS/NPB/.

3 Class C Benchmarks

Since the 1991 specifications of NPB 1.0, computer speed and memory sizes have grown and correspondingly so have representative problem sizes. Many computational aeroscience applications routinely use millions of grid points. NPB 1.0 specifies two problem sizes for each benchmark – class "A" and a larger class "B". The class A benchmarks can now be run on a moderately powerful workstation, and class B benchmarks on high-end workstations or small parallel systems. To retain the focus on high-end supercomputing, we now add a class "C" for all of the NAS benchmarks.

The class C problem sizes are given in Table 1. Iteration counts and validation tolerances are the same as for class B. MG class C should use the class B smoothing operator. Validation numbers for all codes which are also part of NPB 2.0 are included in the NPB 2.0 implementations. For a complete explanation of the manner in which the computed values must be calculated and of the agreement tolerances for other classes, see [5].

Benchmark code	Class A	Class B	Class C
Embarrassingly parallel (EP)	2^{28}	2^{30}	2^{32}
Multigrid (MG)	256^3	256^3	512^3
Conjugate gradient (CG)	14000	75000	150000
3-D FFT PDE (FT)	$256^2 \times 128$	512×256^2	512^3
Integer sort (IS)	2^{23}	2^{25}	2^{27}
LU solver (LU)	64^3	102^3	162^3
Pentadiagonal solver (SP)	64^3	102^3	162^3
Block tridiagonal solver (BT)	64^3	102^3	162^3

Table 1: NAS Parallel Benchmarks Problem Sizes

4 New rules for reporting NPB 1.0 results

NPB 1.0 results have been based, with few exceptions, on proprietary implementations of the benchmarks. NAS has reported performance results, but has not been allowed to release the proprietary codes on which they are based. Vendors have been reluctant to release source code.

While we respect the right of vendors to keep their codes confidential, we no longer believe it serves the interests of the high performance computing (HPC) community for them to remain proprietary. The algorithms needed to write efficient implementations of the NAS Parallel Benchmarks are now well known and well understood.

To balance the right to confidentiality with the needs of the HPC community, we have therefore adopted the following policy on NPB 1.0 results. Vendors are encouraged to make source code publicly available. NAS will place publicly available source code on the world wide web (WWW). If a vendor does not wish to make source code publicly available, NAS will require that a technical report describing the implementations be made publicly available instead. NAS will make this report available on the WWW. This policy applies to all NPB 1.0 results submitted after Supercomputing '96, as a prerequisite for inclusion in the periodic NAS report. NAS encourages vendors to release source code as soon as possible, however.

5 Discussion

NPB 2.0 is the first release of source “portable” NAS Parallel benchmarks. The final three benchmarks will be released some time next year. To track the performance differences between FORTRAN and other popular languages, most notably HPF, parallel-C and C++, NAS also solicits results of “portable” implementations of the NAS Parallel Benchmarks in other languages.

The NAS Parallel Benchmarks were designed to reflect Computational Fluid Dynamics Applications run at NAS in 1991. Five years later, additional algorithms and classes of applications are run at NAS. Therefore NAS intends to release additional benchmarks which represent current trends in Aerophysics computations at NASA. In particular, new benchmarks will stress the I/O and load balancing of new systems.

Acknowledgment

The authors would like to thank and acknowledge the original group of NAS researchers who designed and wrote the NX implementations of these five benchmark codes from which the current ones are derived [5].

References

[1] Numerical Aerodynamic Simulation Program Plan. NAS Systems Division, Ames Research Center, October 1988.

[2] 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.

[3] High Performance Fortran Forum: High Performance Fortran (HPF) Language Specification, Version 1.0. Center for Research in Parallel Computing, P.O. 1892, Rice University, Houston, TX 77251, January 1993, http://www.erc.msstate.edu/hpff/home.html.

[4] Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Mancheck, R.; and Sunderam, V.: PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Scientific and Engineering Computation, Kowalik, J., ed. 1994. http://www.epm.ornl.gov/pvm/pvm_home.html.

[5] Bailey, D. H.; Barszcz, E.; Barton, J. T.; Browning, D. S.; Carter, R. L.; Dagum, L.; Fatoohi, R. A.; Frederickson, P. O.; Lasinski, T. A.; Schreiber, R. S.; Simon, H. D.; Venkatakrishnam, V.; and Weeratunga, S. K.: The NAS Parallel Benchmarks, *International Journal of Supercomputer Applications*, Vol. 5, No. 3, (Fall 1991), pp. 63-73.

[6] 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.

[7] Wang, J.C.H.; Lung, H.; Katsumata, Y.; and Ishigai, T.: "Using Domain Decomposition in the Multigrid NAS Parallel Benchmark on the Fujitsu VPP500," *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, Bailey, D. et al. editors.

[8] Agarwal, R.C.; Alpern, B.; Carter, L.; Gustavson, F. G.; Klepacki, D.J.; Lawrence, R.; and Zubair, M.: "High-performance parallel implementations of the NAS kernel benchmarks on the IBM SP2," *IBM Systems Journal*, Vol. 34, No. 2, 1995.

[9] Naik, V. K.: "A scalable implementation of the NAS Parallel Benchmark BT on distributed memory systems,"*IBM Systems Journal*, Vol. 34, No. 2, 1995.

[10] Bailey, D. H.; Barszcz, E.; Dagum, L.; and Simon, H. D.: "The NAS Parallel Benchmarks Results," *Technical Report NAS 94-001*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, October 1994.

[11] Saini, S.; and Bailey, D. H: “NAS Parallel Benchmarks Results 3-95,”*Technical Report NAS 95-011*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, April 1995.

[12] Saini, S.; and Bailey, D. H: “NAS Parallel Benchmarks Results 10-95,”*Technical Report NAS 95-019*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, October 1995.

[13] Lewis, J.G; and van de Geijn, R. A: “Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithms,” *Supercomputing ’93*, ACM 0-8186-4340-4/93/0011.

[14] Bader, D. A.; and Ja’Ja’, J.: “Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection,” Institute for Advanced Computer Studies (UMIACS), Univ. of Maryland, CS-TR-3493, July, 1995. ftp://ftp.cs.umd.edu/pub/papers/papers/3494/3493.ps.Z.

[15] Sukup, F.; and Fritscher, J.; “Efficiency Evaluation of Some Parallelization Tools on a Workstation Cluster Using the NAS Parallel Benchmarks,’ Vienna Univ. of Technology, 1994.

[16] White, S.; Alund, A.; and Sunderam, V. S.: “Performance of the NAS Parallel Benchmarks on PVM Based Networks,” NASA Ames Research Center, RNR-94-008, May, 1994. http://www.nas.nasa.gov/NAS/reports/RNRreports/otherpeople/RNR-94-008/RNR-94-008.html.

[17] Fatoohi, R.; and Weeratunga, S.: “Performance Evaluation of Three Distributed Computing Environments for Scientific Applications,” *Supercomputing ’94*, IEEE 1063-9535/94.

[18] Ferrari, A.; Filipi-Martin, A.; and Viswanathan, S.: “The NAS Parallel Benchmark Kernels in MPL,” Dept. of Computer Science, Univ. of Virginia, CS-95-39, Sept. 1995.

[19] Dongarra, J.J.; Meuer, H.W.; and Strohmaier, E.: “TOP500 Supercomputer Sites,” RUM 40/94, Nov. 9, 1994.

[20] Bruno, J; Cappello, P.R.: Implementing the Beam and Warming Method on the Hypercube. Proceedings of 3rd Conference on Hypercube Concurrent Computers and Applications, Pasadena, CA, Jan 19-20, 1988.

[21] Naik, N.H.; Naik, V.K.; Nicoules, M.: “Parallelization of a Class of Implicit Finite Difference Schemes in Computational Fluid Dynamics,” *International Journal of High Speed Computing*, Vol. 5, No. 1, pp.1-50.

[22] Breit, S. R.; and Shah, G.: “Implementing the NAS Parallel Benchmarks on the KSR1,” Parallel CFD’93, 1993.

[23] Breit, S.; Celmaster, W.; Coney, W.; Foster, R.; Gaiman, B.; Montry, G.; and Selvidge, C.: "The Role of Computational Balance in the Implementation of the NAS Parallel Benchmarks on the BBN TC2000 Computer," submitted to Concurrency, April 1991.

[24] Imamura, T.; Nogi, T.; Sakai, K.; and Obayashi, Y.: "NAS Parallel Benchmarks on ADENART," Parallel Computational Fluid Dynamics, P-CFD'94, Kyoto, May, 1994.

[25] Sur, S.; and Bohm, W.: "Sorting in an implicitly parallel programming environment: NAS benchmark IS on Monsoon," High Performance Functional Computing Proceedings, ed. Bohm, A. P. W. and Feo, J. T., April, 1995.

[26] Hayashi, K.; Doi, T.; Horie, T.; Koyanagi, Y.; Shiraki, O.; Imamura, N.; Shimizu, T.; Ishihata, H.; and Shindo, T.: "AP1000+: Architectural Support for Parallelizing Compilers," Transactions of Information Processing Society of Japan, Vol. 36, No. 7, July, 1995.

[27] Gibson, T. L: "NAS Parallel Conjugate Gradient Benchmark on the Cray T3D," SRC-TR-94-129, Supercomputing Research Center, Bowie, MD, Jan., 1995.

[28] Van der Wijngaart, R.F.: Efficient Implementation of a 3-dimensional ADI Method on the iPSC/860. Supercomputing '93, Portland, OR, November 1993, IEEE Press.

[29] Papiani, M.; Hey, A.J.G.; and Hockney, R.W.: "The Graphical Benchmark information Service," *Scientific Programming*, Vol. 4, 1995, http://hpcc.soton.ac.uk/RandD/gbis/papiani-new-gbis-top.html.

[30] Simon, H. D.; and Strohmaier, E.: "Amdahl's Law and the Statistical Content of the NAS Parallel Benchmarks," Supercomputing, Nov. 1995.

[31] Barszcz, E.; Fatoohi, R.; Venkatakrishnan, V.; and Weeratunga, S.: "Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors"*Technical Report NAS RNR-93-007*, NASA Ames Research Center, Moffett Field, CA, 94035-1000, April 1993.

A Explanation of Makefiles and Run Procedures

The authors of these codes and Makefiles have attempted to create an environment that allows a user to generate the executables and obtain benchmark numbers with little effort.

The basic file structure of this environment under UNIX is

```
NPB/config !configurations files which must be edited or chosen
NPB/sys !utility programs for the environment
NPB/bin !location of executables
NPB/FT !FT benchmark source
NPB/MG !MG source
NPB/LU !LU-CFD source
NPB/BT !BT-CFD source
NPB/SP !SP-CFD source
NPB/MPI_dummy !dummy MPI library for SMP or single node runs
NPB/samples !sample NQS, PBS and Interactive scripts
```

A.1 Edit the configuration file

Edit the site- and machine-specific data in `config/make.def`. Several sample versions are included in this directory so you may be able to copy one. A clean version is in `config/make.def.template`. Sample `make.def` files are in the config directory. Here is a sample one for an INTEL PARAGON running OSF:

```
# Site-specific definitions.
# The following must be defined:
# MPI_LIB  - any -L and -l arguments required for linking MPI programs
# MPI_INC  - any -I arguments required for compiling MPI programs
# F77      - fortran compiler
# LOAD     - loader
# FFLAGS   - compilation arguments
# LDFLAGS   - loader arguments
# BINDIR   - destination directory for executables
#
# compilations are done with $(F77) $(MPI_INC) $(FFLAGS) or
#                            $(F77) $(FFLAGS)
# linking is done with       $(LOAD) $(MPI_LIB) $(LDFLAGS)
# flags must be such that DOUBLE PRECISION data is 64 bits
# In particular, you need -dp on a Cray

F77     =       /mpi/mpich/lib/paragon/ch_nx/mpif77
LOAD    =       /mpi/mpich/lib/paragon/ch_nx/mpif77
```

```
MPI_LIB =       -L/app/mpi/mpich/lib/paragon/ch_nx
MPI_INC =       -I/app/mpi/mpich/include
FFLAGS  =       -O4 -Mvect -Mnostride0 -Knoieee -Mnodebug -Mextend
LDFLAGS  =       -nx -O4 -Knoieee -Mnodebug -Mextend
CC      =        cc
BINDIR  =       ../bin

# The following variables are set correctly for most
# architectures. They need to be set if the compiler
# flags used above change the size of any of the
# Fortran data-types INTEGER, DOUBLE PRECISION or DOUBLE COMPLEX
# If the default sizes have been changed, MPI will send
# the wrong amount of data. You need to tell MPI which of
# its builtin data-types to use to send this resized data.
# For instance, if you use -dp on a Cray to demote
# double precision data to 64 bits, DP_TYPE should
# be MPI_REAL, which is the standard 64-bit REAL type.

# DP_TYPE      - correct MPI type to send DOUBLE PRECISION data
# INTEGER_TYPE - correct MPI type to send INTEGER data
# DC_TYPE      - correct MPI type to send DOUBLE COMPLEX data

DP_TYPE         =       MPI_DOUBLE_PRECISION
INTEGER_TYPE    =       MPI_INTEGER
DC_TYPE         =       MPI_DOUBLE_COMPLEX
```

The compile and load strings that are defined in `make.def` are included in `NPB/sys/setparams.h` and appear in the output from the benchmark runs.

A.2 Making One executable

Each benchmark comes in 4 sizes (classes): A, B, C and S(ample). Since Fortran 77 doesn't have dynamic memory allocation, both the class and the number of processes must be specified at compile time. Some benchmarks (FT, MG, LU) run on a power-of-2 number of processes. Others (SP, BT) run on a square number of processes (1, 4, 9, ...)

To compile a given benchmark for specific class and number of processes, type

```
make benchmark-name CLASS={A,B,C,S} NPROCS=#
```

For instance, to create a class B version of the SP benchmark that runs on 16 processes, type:

```
        make sp CLASS=B NPROCS=16
```

If you specify an illegal number of processes for a given benchmark or an unknown class, the compilation aborts.

The executable is placed in the subdirectory “bin" of the distribution (or in the directory BINDIR specified in make.def, if you’ve defined it). The name of the executable is “benchmark-name.CLASS.NPROCS", e.g., “sp.A.16".

A.3 Making a set of executables

The procedure in item 2 allows you to build one benchmark at a time. To build a whole suite, you can type

```
        make suite
```

Make will look in file config/suite.def for a list of executables to build. The file contains one line per specification, with comments preceded by “#". Each line contains the name of a benchmark, the class, and the number of processors, separated by spaces or tabs. For instance, the file could contain:

```
# This is a sample suite file to build several executables
sp      A    16
sp      A    25
sp      A    36
ft      B    1
ft      B    2
ft      B    4
```

The config directory also contains several sample suites.

B The MPI_Dummy Library

The benchmarks have been designed so that they can be run on a single processor without an MPI library, if desired. A few “dummy" MPI routines are still required for linking. For convenience, such a library is supplied in the “MPI_DUMMY" subdirectory of the distribution. It contains a mpif.h include file which must be used as well. Typing “make" in the mpi_dummy directory should build the dummy library, libmpi.a. You must modify make.def to use this dummy library and include file.

This is far from a complete implementation of MPI and only contains those routines needed by the NPB 2.0 Benchmarks. Copies are used in the “Reduce” routines and the library will exit if any communication is actually attempted.

C Sample Scripts

Several sample scripts are provided to either run the executables interactively, submit an NQS batch job, or a PBS batch job. For instance, here is a simple NQS script which will email the user at the beginning and end of the job and run a CLASS A, FT benchmark on 128 nodes of a PARAGON.

```
#
# QSUB -mb -me
cd $HOME/NPB/bin
date
ft.A.128 -plk -sz 128 >../ft.A.128.out
date
```

Field	Type	Size	Key	Optional?
hardware_key	string	80	PRIMARY_KEY	NOT_NULL
manufacturer	string	80		NOT_NULL
machine_type	string	60		NOT_NULL
model	string	60		
cpu	string	60		
cpu_clock_mhz	int	4		
L1_cache_KB	int	4		
L2_cache_KB	int	4		
other_cache_KB	int	4		
min_memory_MB	int	4		
disk	string	80		
interconnect_type	string	80		
cost_in_US_dollars	real	8		
other_hardware	string	80		
h_comments	string	80		

Table 2: Table Name: hardware

D Submittal of Results

NAS requests to receive the output files, all changed source, makefiles and scripts used to generate the submittal.

An mSQL database has been created as a repository for the NPB 2.0 results. The format of a submittal consists of four SQL tables, ‘hardware’, ‘software’,‘submitter’, and ‘result.’ There will be one ‘result’ table for each benchmark and possibly several ‘software’ and ‘submitter’ entry for each ‘hardware’ table. Unique keys should be provided in the ‘result’ table to link the tables. Examples are illustrated below. >From the output files, NAS will create SQL input for the “results” table below. However, the submitter should include the information for the other three tables. Entries below marked “NOT_NULL” must be completed.

In addition, a WWW server to submit results will be accessible through the NPB web page already mentioned. NAS reserves the right to review all submissions before inclusion into the SQL database.

Field	Type	Size	Key	Optional?
software_key	string	80	PRIMARY_KEY	NOT_NULL
operating_system	string	80		NOT_NULL
OS_version	string	12		
compiler_name	string	15		
compiler_version	string	80		
compiler_switches	string	80		
MPI_name	string	10		
MPI_version	string	10		
s_comment	string	80		

Table 3: Table Name: software

Field	Type	Size	Key	Optional?
submitter_key	string	80	PRIMARY_KEY	NOT_NULL
name	string	80		NOT_NULL
organization	string	80		NOT_NULL
email	string	80		NOT_NULL
telephone	string	20		
date_sent	string	9		

Table 4: Table Name: submitter

Field	Type	Size	Key	Optional?
hardware_key	string	80		
software_key	string	80		
submitter_key	string	80		
benchmark	string	20		NOT_NULL
version	real	8		NOT_NULL
percent_modified	int	4		
patch_level	string	2		
problem_size	string	1		NOT_NULL
number_nodes	int	4		NOT_NULL
time_secs	real	8		NOT_NULL
Mflop/s	real	8		NOT_NULL
date_benchmarked	string	9		NOT_NULL
r_comments	string	80		

Table 5: Table Name: result

E Further Information

NAS will maintain two mailing lists and a WEB page concerning these benchmarks.

Permission to use, copy, distribute and modify this software for any purpose with or without fee is hereby granted. We request, however, that all derived work reference the NAS Parallel Benchmarks 2.0. This software is provided "as is" without express or implied warranty.

Information on NPB 2.0, including the technical report, the original specifications, source code, results and information on how to submit new results, is available at:

http://www.nas.nasa.gov/NAS/NPB/

```
Send comments or suggestions to  npb@nas.nasa.gov
Send bug reports to              npb-bugs@nas.nasa.gov

      NAS Parallel Benchmarks Group
      NASA Ames Research Center
      Mail Stop: T27A-1
      Moffett Field, CA   94035-1000

      Fax:     (415) 604-3957
```

E-mail: npb@nas.nasa.gov