# Performance Results for Two of the NAS Parallel Benchmarks

David H. Bailey

NAS Applied Research Branch
NASA Ames Research Center
Moffett Field, CA 94035

Paul O. Frederickson

RIACS
NASA Ames Research Center
Moffett Field, CA 94035

**Abstract**

Two problems from the recently published "NAS Parallel Benchmarks" have been implemented on three advanced parallel computer systems. These two benchmarks are the following: (1) an "embarrassingly parallel" Monte Carlo statistical calculation and (2) a Poisson partial differential equation solver based on three-dimensional fast Fourier transforms. The first requires virtually no interprocessor communication, while the second has a substantial communication requirement.

This paper briefly describes the two problems studied, discusses the implementation schemes employed, and gives performance results on the Cray Y-MP, the Intel iPSC/860 and the Connection Machine-2.

Bailey is with the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center, Moffett Field, CA 94035. Frederickson is with the Research Institute for Advanced Computer Science (RIACS) at NASA Ames. Frederickson's work was funded by the NAS Systems Division via Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association.

## 1 Introduction

The NAS Parallel Benchmarks [3] is a new set of benchmarks that have been developed for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five "parallel kernels", and three simulated computational fluid dynamics (CFD) application benchmarks. Together they mimic the computation and data movement characteristics of many large scale aerophysics applications.

These benchmarks were developed out of the realization that none of the conventional approaches to the performance benchmarking of supercomputers is appropriate for highly parallel systems. The popular "kernel" benchmarks that have been used for traditional vector supercomputers, such as the Livermore Loops, the LINPACK benchmark and the original NAS Kernels, are clearly inappropriate for the performance evaluation of highly parallel machines. In addition to the dubious meaning of the actual performance numbers, the computation and memory requirements of these programs do not do justice to the vastly increased capabilities of the new parallel machines, particularly those systems that will be available by the mid-1990s.

On the other hand, a full scale scientific application is similarly unsuitable. First of all, porting a large application to a new parallel computer architecture requires a major effort, and it is usually hard to justify a major research task simply to obtain a benchmark number. For example, there are as yet few results for the PERFECT benchmark suite on highly parallel systems.

An alternative to conventional benchmark strategies is to specify a "paper and pencil" benchmark, i.e. to define a set of problems only algorithmically. Even the input data is specified only on paper. Naturally, the problem has to be specified in sufficient detail that a unique solution exists, and the required output has to be brief yet detailed enough to certify that the prob-

lem has been solved correctly. The person or persons implementing the benchmarks on a given system are expected to solve the various problems in the most appropriate way for the specific system. The choice of data structures, algorithms, processor allocation and memory usage are all (to the extent allowed by the specification) left open to the discretion of the implementer.

The NAS Parallel Benchmarks were constructed on this model. They consist of eight separate benchmark problems:

1. An "embarrassingly parallel" Monte-Carlo statistical computation.

2. A simplified multigrid partial differential equation (PDE) solver.

3. A conjugate gradient eigenvalue computation that involves unstructured matrices.

4. A Poisson PDE solver that employs three-dimensional fast Fourier transforms (FFTs).

5. An integer sort problem, which is used in some particle codes.

6. The LU solver of a simulated CFD application.

7. The scalar pentadiagonal solver of a simulated CFD application.

8. The block tridiagonal solver of a simulated CFD application.

There are several basic requirements on implementations: (1) 64-bit floating point arithmetic must be used, (2) programs must be coded in Fortran-77 or C, although a variety of commonly used parallel extensions are allowed, (3) except for a short list of common library functions and intrinsics, assembly language routines may not be used for any computations. Otherwise programmers are free to utilize algorithms and language constructs that give the best performance possible on the particular system being studied. The detailed definitions of the problems to be solved, as well as the specific language rules and timing procedures are given in [3].

This paper describes the implementation of problems 1 and 4 on the following three systems: (1) a Cray Y-MP with eight processors, (2) an Intel iPSC/860 system with 128 processors, and (3) a Connection Machine-2 with 32,768 processors.

## 2 The "Embarrassingly Parallel" Benchmark

In this problem, a large number of pairs of Gaussian random deviates are generated according to a specific scheme, and the number of pairs lying in successive square annuli are tabulated. The only essential requirement for communication in a multiprocessor implementation is to collect the counts at the end. This problem represents the essence of many Monte Carlo physics calculations, and it is also typical of such diverse applications as the analysis of supercomputer memory systems [2]. Another reason for its inclusion in the NAS Parallel Benchmarks is to provide a contrast with other problems, such as the 3-D FFT PDE solver, which require substantial interprocessor communication.

The following is a brief statement of this problem. The complete statement, plus references, are given in [3].

Set $n = 2^{28}$ and $s = 271828183$. Generate the pseudorandom floating point values $r_j$ in the interval (0, 1) for $1 \le j \le 2n$ using the scheme described below. Then for $1 \le j \le n$ set $x_j = 2r_{2j-1} - 1$ and $y_j = 2r_{2j} - 1$. Thus $x_j$ and $y_j$ are uniformly distributed on the interval $(-1, 1)$.

Next set $k = 0$, and beginning with $j = 1$, test to see if $t_j = x_j^2 + y_j^2 \le 1$. If not, reject this pair and proceed to the next $j$. If this inequality holds, then set $k \leftarrow k + 1$, $X_k = x_j\sqrt{(-2\log t_j)/t_j}$ and $Y_k = y_j\sqrt{(-2\log t_j)/t_j}$, where log denotes the natural logarithm. Then $X_k$ and $Y_k$ are independent Gaussian deviates with mean zero and variance one. Approximately $n\pi/4$ pairs will be constructed in this manner.

Finally, for $0 \le l \le 9$ tabulate $Q_l$ as the count of the pairs $(X_k, Y_k)$ that lie in the square annulus $l \le \max(|X_k|, |Y_k|) < l + 1$, and output the ten $Q_l$ counts. Each of the ten $Q_l$ counts must agree exactly with reference values.

The $2n$ uniform pseudorandom numbers $r_j$ mentioned above are to be generated according to the following scheme: Set $a = 5^{13}$ and let $x_0 = s$ be the specified initial "seed". Generate the integers $x_k$ for $1 \le k \le 2n$ using the linear congruential recursion

$$x_{k+1} \quad = \quad ax_k \pmod{2^{46}}$$

and return the numbers $r_k = 2^{-46}x_k$ as the results. Observe that $0 < r_k < 1$ and the $r_k$ are very nearly uniformly distributed on the unit interval.

An important feature of this pseudorandom number generator is that any particular value $x_k$ of the sequence can be computed directly from the initial

seed $s$ by using the binary algorithm for exponentiation, taking remainders modulo $2^{46}$ after each multiplication. The importance of this property for parallel processing is that numerous separate segments of a single, reproducible sequence can be generated on separate processors of a multiprocessor system. Many other widely used schemes for pseudorandom number generation do not possess this important property.

## 3 The 3-D FFT PDE Benchmark

In this problem, a certain Poisson partial differential equation (PDE) is solved using three-dimensional fast Fourier transform (FFT) computations. In contrast to the "embarrassingly parallel" problem, this application requires substantial interprocessor communication and is thus a good test of network performance.

The following is a brief description of this benchmark. For full details, see [3].

Consider the PDE

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t)$$

where $x$ is a position in 3-dimensional space. When a Fourier transform is applied to each side, this equation becomes

$$\frac{\partial v(z,t)}{\partial t} = -4\alpha\pi^2 |z|^2 v(z,t)$$

where $v(z,t)$ is the Fourier transform of $u(x,t)$. This has the solution

$$v(z,t) = e^{-4\alpha\pi^2|z|^2 t} v(z,0)$$

Now consider the discrete version of the original PDE. Following the above, it can be solved by computing the forward 3-D discrete Fourier transform (DFT) of the original state array $u(x,0)$, multiplying the results by certain exponentials, and then performing an inverse 3-D DFT.

The specific problem to be solved in this benchmark is as follows. Set $n_1 = 256$, $n_2 = 256$, and $n_3 = 128$. Generate $2n_1n_2n_3$ 64-bit pseudorandom floating point values using the pseudorandom number generator in the previous section, starting with the initial seed 314159265. Then fill the complex array $U_{i,j,k}$, $0 \le i < n_1$, $0 \le j < n_2$, $0 \le k < n_3$, with this data, where the first dimension varies most rapidly as in the ordering of a 3-D Fortran array. A single complex number entry of $U$ consists of two consecutive pseudorandomly generated results. Compute the forward 3-D DFT of $U$, using a 3-D fast Fourier transform (FFT) routine, and call the result $V$. Set $\alpha = 10^{-6}$ and set $t = 1$. Then compute

$$W_{i,j,k} = e^{-4\alpha\pi^2(\bar{i}^2+\bar{j}^2+\bar{k}^2)t} V_{i,j,k}$$

where $\bar{i}$ is defined as $i$ for $0 \le i < n_1/2$ and $i - n_1$ for $n_1/2 \le i < n_1$. The indices $\bar{j}$ and $\bar{k}$ are similarly defined with $n_2$ and $n_3$. Then compute an inverse 3-D DFT on $W$, using a 3-D FFT routine, and call the result the array $X$. Finally, compute the complex checksum $\sum_{i=0}^{1023} X_{q,r,s}$ where $q = i \pmod{n_1}$, $r = 3i \pmod{n_2}$ and $s = 5i \pmod{n_3}$. After the checksum for this $t$ has been output, increment $t$ by one. Then repeat the above process, from the computation of $W$ through the incrementing of $t$, until the step $t = N$ has been completed. In this benchmark, $N = 6$. The $V$ array and the array of exponential terms for $t = 1$ need only be computed once. Note that the array of exponential terms for $t > 1$ can be obtained as the $t$-th power of the array for $t = 1$.

The rules of the NAS Parallel Benchmarks require that all problems be implemented in Fortran-77 or C, with some reasonable parallel extensions, and prohibit the usage of assembly language computations. However, one exception to this is that vendor-supplied, assembly-coded library routines may be employed to compute either individual 1-D FFTs or complete 3-D FFTs. Thus this benchmark can be used to contrast the performance one can obtain from a Fortran implementation to an implementation that employs certain vendor library routines.

## 4 System Descriptions

The Cray Y-MP used in this study is the system in the NAS facility at NASA Ames Research Center. It has eight processors and 128 million words of bipolar random access memory. It also has 256 million words of solid state memory, but this was not utilized here. Its clock period is 6 nanoseconds, so that its theoretical peak performance rate is approximately 2.66 GFLOPS. These codes were compiled and executed under UNICOS 6.0 system software.

The Intel iPSC/860 used in this study, which is also at the NAS facility, employs Intel's new RISC processor, the i860. The system has 128 nodes, each with eight megabytes of memory, connected with in a hypercube network. The i860 has a theoretical peak performance of 60 MFLOPS (64 bit), so that the theoretical peak performance of the system is some 7.68 GFLOPS. In practice it is hard to even approach this

rate, even from assembly code [5]. The codes were compiled using the Portland Group Fortran compiler, version 1.3a.

The NAS CM-2 has 32,768 bit-serial processors together with 1,024 Weitek 64-bit floating point processors and four gigabytes of memory. Its theoretical peak performance is some 14 GFLOPS, although again few codes can approach this rate. The 1.0 system software, which includes the 1.0 slicewise Fortran compiler and CMSSL library, was used in this study.

## 5 Implementation Techniques

Curiously, the principal algorithmic challenge in implementing the embarrassingly parallel benchmark problem was in devising a scheme suitable for high performance on a Cray supercomputer. In contrast, straightforward implementations sufficed on the Intel and CM-2 systems.

The principal difficulty on the Cray is that the linear congruential scheme presented above for generating uniform pseudorandom numbers is not directly vectorizable. One solution, which was employed here, is to generate these numbers in batches of 64, which is the hardware vector length of the Cray systems. This may be done by first computing the multiplier $\bar{a} = a^{64} \pmod{2^{46}}$. The first 64 elements of the sequence $x_k$ are computed by the scalar algorithm. Thereafter the remainder are computed by the recursion

$$x_{k+64} \quad = \quad \bar{a} x_k \pmod{2^{46}}$$

The acceptance-rejection operation in the generation of the Gaussian deviates, as well as the accumulation of counts, also present difficulties for vector computations, but these were resolved by reasonably straightforward means.

In the 3-D FFT PDE problem, the principal issue is the selection of an efficient means to compute 3-D FFTs. On the Cray, complex arrays of size $n_1 \times n_2 \times n_3$, where $n_1, n_2$ and $n_3$ are powers of two, were declared with physical dimensions $(n_1+1, n_2+1, n_3+1)$. Then transforms were performed in all three dimensions, with vectorization in the plane not transformed. This is efficient on the Cray since the strides of the resulting array accesses are always either one or else one greater than a power of two. Either way, no bank conflicts result. The vectorized FFTs were based on a 1-D FFT algorithm described in [1].

Multiprocessing on the Cray was achieved by inserting a few autotasking directives. The FFTs were performed by an all-Fortran routine as described above and also by calling Cray's new library routine CFFT3D. This routine permits both single processor and multiprocessor runs to be made.

On the Intel, accessing arrays in each of three dimensions is of course not practical because in at least one dimension, the accesses will be across processors. Thus the following alternate scheme was employed:

1. $n_2 n_3$ 1-D FFTs of size $n_1$ are performed, each on data vectors entirely contained in a single processing node (in fact, within the cache of the processor).

2. $n_1 n_3$ 1-D FFTs of size $n_2$ are performed. These data vectors are also contained within individual nodes, since $n_3$ never exceeds the number of nodes used.

3. The resulting complex data array is then transposed to a $n_3 \times n_1 \times n_2$ array by means of a complete exchange operation.

4. $n_1 n_2$ 1-D FFTs of size $n_3$ are performed, again on local node data.

5. The resulting complex data array is transposed back to a $n_1 \times n_2 \times n_3$ array.

Note that steps 3 and 5 require a "complete exchange" operation, i.e. the data in each processing node is partitioned into $p$ sections, where $p$ is the number of processors, and then each processor sends each of its $p$ sections to the appropriate target processor. Since this step requires substantial interprocessor communication, it is essential that it be as efficient as possible. Recently Bokhari [4] compared a number of schemes for this operation. The scheme used here is the one described in that paper as the forced, pair-wise, synchronized scheme, which is due to Seidel [6].

Recently Kuck and Associates completed a package of assembly-coded mathematical routines for i860-based systems, under contract to Intel. These routines included all of the basic linear algebra subroutines (BLAS), plus some one-dimensional FFT routines. The double precision 1-D complex-to-complex FFT routine from this package was incorporated into the benchmark code, so that results have been obtained using both this routine and an all-Fortran version.

One scheme attempted on the CM-2 is similar to that employed on the Cray, except that there is no need for array dimensions to be one greater than powers of two. Another attempted scheme is similar to

the algorithm employed for the Intel system. Unfortunately, for reasons not yet fully understood as of the date of this paper, neither scheme exhibits even remotely respectable performance — evidently significant parts of the computation are not really being performed in parallel on the CM-2. However, TMC has recently provided a rather efficient routine for one-dimensional and multi-dimensional FFTs in the CMSSL library. Thus this routine was employed in the runs cited below.

One important element of the tuning for the CM-2 implementation was the specification of the layout of the 3-D array to be transformed. This was done with the directive

```
cmf$ layout X1(:serial, 1000:send, 1:send)
```

The authors are indebted to Robert Krawitz of TMC for this suggestion.

All of the implementations for this benchmark employ ordered FFTs. Some improvement in performance may be possible by using unordered (i.e. bit reversed) FFTs. Future benchmark efforts will explore this possibility.

## 6 Performance Results

The embarrassingly parallel benchmark was run for three problem sizes: $n = 2^{24}, 2^{28}$ and $2^{30}$. All three systems, including the Cray, which is known for difficulties with numerical inaccuracies, obtained precisely correct counts, apparently confirming that billions of floating operations were performed correctly (within acceptable tolerance limits) and that the library routines SQRT and LOG returned satisfactorily accurate results.

Performance results for this benchmark are listed in Table 1. In accordance with the rules for the NAS Parallel Benchmarks, run times are the elapsed time of day from start of execution to completion, and MFLOPS figures are for 64-bit data, with operation counts as given in [3], which in turn are based on some standard figures for the SQRT and LOG functions. In cases where two or more runs were possible with the same system on the same size problem, but with different numbers of processors, "speedup" figures are shown in the sixth column. These numbers are normalized based on the run with the smallest number of processors.

Not surprisingly, these runs exhibit almost perfectly linear speedups. What also scales linearly is the run time versus the size of the problem — timings for equivalent systems with $n = 2^{30}$ are almost exactly four times the timings for $n = 2^{28}$. The results indicate that on this type of problem, the full Intel system is roughly equivalent to 2.5 Y-MP processors, and that the full CM-2 is roughly equivalent to three Y-MP processors.

It is interesting to note that these results are 41%, 4.7% and 3.1% of the peak performance rates of the Y-MP, the Intel and the CM-2, respectively. The low percentages on the Intel and CM-2 are in spite of a minimal communication requirement. These figures thus underscore that there is considerable room for improvement in the performance of the Intel and CM-2 Fortran compilers and library intrinsics.

The 3-D FFT PDE benchmark was run with the problem sizes $64 \times 64 \times 64$, $128 \times 128 \times 128$ and $256 \times 256 \times 128$. All three systems produced checksums at the end of each iteration that were correct to the required 12 significant figures. Timing results are shown in Table 2. "F" or "L" in the Code column denotes an all-Fortran or a library FFT (a 1-D FFT for the Intel, and a 3-D FFT for the Cray and the CM-2).

Timings on the CM-2 were problematic for this benchmark. The CM timing routines output two run times: "CM elapsed time" and "CM busy time". The former is the elapsed time of day between start of execution and end, and corresponds to the definition of run time for the NAS Parallel Benchmarks. The latter excludes time when the CM is not actually busy with computation, such as when it is exchanging data with the front end system.

In the first benchmark, as well as in a majority of applications, these times are not greatly different. However, in the 3-D FFT PDE benchmark they are greatly different, especially in the $64 \times 64 \times 64$ problem size. This difference is apparently due to time spent in transferring data and program code from the front end to the CM-2 as part of initialization for the library FFT routine. Thinking Machines plans to greatly reduce this initialization time in future software releases. For this reason, and the fact that the CM busy timings scale more predictably with problem size and number of processors, the CM busy timings were also included in the table. The CM elapsed and CM busy timings are distinguished by "E" or "B" in the Code column.

| System | Problem Size | No. Proc. | Time (sec.) | MFLOPS | Speedup |
|---|---|---|---|---|---|
| Y-MP | $2^{24}$ | 1 | 9.41 | 147.40 | 1.00 |
| | | 8 | 1.25 | 1109.60 | 7.53 |
| Intel | | 4 | 122.58 | 11.32 | 1.00 |
| | | 8 | 61.30 | 22.63 | 2.00 |
| | | 16 | 31.60 | 43.89 | 3.88 |
| | | 32 | 15.77 | 87.95 | 7.77 |
| | | 64 | 7.67 | 180.83 | 15.98 |
| | | 128 | 3.84 | 361.20 | 31.92 |
| CM-2 | | 8K | 13.11 | 105.80 | 1.00 |
| | | 16K | 6.55 | 211.76 | 2.00 |
| Y-MP | $2^{28}$ | 1 | 150.10 | 147.90 | 1.00 |
| | | 8 | 20.10 | 1104.48 | 7.47 |
| Intel | | 32 | 245.20 | 90.54 | 1.00 |
| | | 64 | 122.61 | 181.06 | 2.00 |
| | | 128 | 61.32 | 362.03 | 4.00 |
| CM-2 | | 8K | 201.04 | 110.44 | 1.00 |
| | | 16K | 100.84 | 220.15 | 2.00 |
| | | 32K | 50.92 | 435.98 | 3.95 |
| Y-MP | $2^{30}$ | 1 | 603.50 | 147.14 | 1.00 |
| | | 8 | 80.63 | 1101.33 | 7.48 |
| Intel | | 128 | 245.21 | 362.14 | |
| CM-2 | | 32K | 202.94 | 437.57 | |

Table 1: Embarrassingly Parallel Benchmark Performance Results

| System | Code | Problem Size | No. Proc. | Time (sec.) | MFLOPS | Speedup |
|---|---|---|---|---|---|---|
| Y-MP | F | $64 \times 64 \times 64$ | 1 | 1.21 | 156.85 | |
| | L | | 1 | 0.82 | 231.46 | |
| Intel | F | | 4 | 17.08 | 11.11 | 1.00 |
| | F | | 8 | 9.01 | 21.06 | 1.90 |
| | F | | 16 | 4.77 | 39.78 | 3.58 |
| | F | | 32 | 2.51 | 75.68 | 6.81 |
| | L | | 4 | 11.87 | 15.99 | 1.00 |
| | L | | 8 | 6.39 | 29.70 | 1.86 |
| | L | | 16 | 3.41 | 55.68 | 3.48 |
| | L | | 32 | 1.84 | 103.29 | 6.46 |
| CM-2 | LE | | 8K | 10.76 | 17.64 | |
| | LB | | 8K | 3.16 | 60.06 | 1.00 |
| | LE | | 16K | 8.24 | 23.03 | |
| | LB | | 16K | 1.82 | 104.29 | 1.74 |
| Y-MP | F | $128 \times 128 \times 128$ | 1 | 10.27 | 169.33 | |
| | L | | 1 | 7.14 | 243.56 | 1.00 |
| Intel | F | | 32 | 19.32 | 90.02 | 1.00 |
| | F | | 64 | 9.95 | 174.82 | 1.94 |
| | F | | 128 | 5.45 | 319.34 | 3.54 |
| | L | | 32 | 13.70 | 126.89 | 1.00 |
| | L | | 64 | 7.14 | 243.67 | 1.92 |
| | L | | 128 | 4.05 | 429.77 | 3.39 |
| CM-2 | LE | | 8K | 30.79 | 56.48 | |
| | LB | | 8K | 24.67 | 70.49 | 1.00 |
| | LE | | 16K | 17.11 | 101.64 | |
| | LB | | 16K | 12.07 | 144.08 | 2.04 |
| | LE | | 32K | 13.06 | 133.15 | |
| | LB | | 32K | 6.13 | 283.69 | 4.02 |
| Y-MP | F | $256^2 \times 128$ | 1 | 39.23 | 192.23 | 1.00 |
| | F | | 8 | 5.17 | 1458.61 | 7.59 |
| | L | | 1 | 29.31 | 257.28 | 1.00 |
| | L | | 8 | 6.15 | 1226.18 | 4.77 |
| Intel | F | | 128 | 22.22 | 339.44 | |
| | L | | 128 | 15.13 | 498.45 | |
| CM-2 | LE | | 16K | 51.20 | 147.29 | |
| | LB | | 16K | 47.06 | 160.24 | 1.00 |
| | LE | | 32K | 28.59 | 263.76 | |
| | LB | | 32K | 24.50 | 307.80 | 1.92 |

Table 2: 3-D FFT PDE Benchmark Performance Rates

The results for this benchmark definitely indicate sub-linear speedup with increasing numbers of processors on the Intel. On the $64^3$ problem, the Intel system (with the library 1-D FFT routine) is only 6.5 times faster with 32 processors than with 4. The CM-2 is better in this regard: on the $128^3$ problem, the CM-2 is actually 4.02 times faster with 32K processors than with 8K, although a fall-off in performance is seen with the other two problem sizes. Curiously, the all-Fortran Cray program exhibited more nearly linear speedups and actually out-performed the library FFT-based program with eight processors on the $256 \times 256 \times 128$ problem.

However, it should be emphasized that speedup figures can be easily misinterpreted. For example, one reason that the speedup figures are not worse on the Intel is that the current Intel Fortran compiler is still not achieving nearly the performance that can be obtained through assembly coding. The single node performance rate for the all-Fortran code on the Intel system is only about 3 MFLOPS, whereas the peak 64-bit performance of the i860 is 60 MFLOPS. When a more powerful compiler becomes available, then these speedup figures can be expected to fall off more rapidly, because the performance will be dominated to a greater extent by communication costs.

The performance rates on the largest problem size indicate that the full Intel system is roughly equivalent to 1.9 Y-MP processors (comparing library FFT timings), and that the CM-2 is roughly equivalent to 1.2 Y-MP processor (comparing the CM busy timings with the Y-MP library rates). The fact that these ratios are not as favorable to the highly parallel systems on the 3-D FFT PDE benchmark as they were on the embarrassingly parallel benchmark is clearly due to the more demanding communication requirement in the 3-D FFT.

## 7 Conclusions

With some algorithmic experimentation and implementation tuning, all three of the tested systems were able to obtain "respectable" performance rates on these two benchmark problems. However, the Intel and CM-2 systems, while showing promise, are not yet demonstrating on these problems the consistently high level of sustained performance that users of Cray systems have come to expect (comparing full systems to full systems).

It is true that the performance of highly parallel computers on a particular application can in some cases be significantly improved by employing alternate algorithms that feature less costly communication. However, in many cases, including the 3-D FFT PDE benchmark and other problems that require intensive computations in each dimension of three dimensional arrays, only modest improvement can be achieved in this manner, since a certain amount of communication is unavoidable. Also, many CFD applications and other hyperbolic PDE problems must employ implicit algorithms in order to obtain a solution in a reasonable amount of time, and these implicit algorithms require substantial long-distance communication. What this means is that the usability of a highly parallel system will be quite limited if it cannot perform well on communication intensive problems. Thus it is hoped that the next generation of highly parallel supercomputers will feature greatly increased communication performance.

## References


[1] Bailey, D. H., "A High-Performance FFT Algorithm for Vector Supercomputers, *International Journal of Supercomputer Applications*, vol. 2 (1988), p. 82 – 87.

[2] Bailey, D. H., "Vector Computer Memory Bank Contention", *IEEE Transactions on Computers*, vol. C-36, no. 3 (Mar. 1987), p. 293 - 298.

[3] Bailey, D. H., Barton, J. T., Lasinski, T. A, and Simon, H. D., eds., "The NAS Parallel Benchmarks", RNR Technical Report RNR-91-002, NASA Ames Research Center, January 1991.

[4] Bokhari, S. H., "Complete Exchange on the iPSC-860", ICASE Report No. 91-4, NASA Langley Research Center, January 1991.

[5] Lee, K., "On the Floating Point Performance of the i860 Microprocessor", RNR Technical Report RNR-90-019, NASA Ames Research Center, October 1990.

[6] Seidel, S., Lee, M-H., and Fotedar, S., "Concurrent Bidirectional Communication on the Intel iPSC/860 and iPSC/2", Technical Report CS-TR 9006, Dept. of Computer Science, Michigan Technical University, Nov. 1990.