Experience with Parallel Computers at NASA Ames

David H. Bailey

RNR Technical Report RNR-91-007
November 14, 1991

Abstract

Beginning in 1988, the Numerical Aerodynamic Simulation (NAS) organization at NASA Ames Research Center has studied the usability of highly parallel computers on computational fluid dynamics and other aerophysics applications. Presently this organization operates a CM-2 with 32,768 nodes and an Intel iPSC/860 with 128 nodes. This note gives an overview of the experience in using these systems, highlights both strong points and weak points of each, and discusses what improvements will be required in future highly parallel systems in order that they can gain acceptance in the mainstream of scientific computing.

The author is with the NAS Applied Research Branch at NASA Ames Research Center, Moffett Field, CA 94035.

Introduction

NASA Ames Research Center has long been a trail blazer in the field of high performance scientific computation, particularly for computational fluid dynamics (CFD) and other aerophysics applications. Back in the 1970s NASA Ames was the home of the Illiac IV, which featured a peak performance of approximately 250 MFLOPS. In 1982, NASA Ames took delivery of the first two processor Cray X-MP, which featured a peak performance of nearly 400 MFLOPS. In 1986 the newly formed Numerical Aerodynamic Simulation (NAS) organization at NASA Ames installed the first full-memory Cray-2, with a peak performance of 1.6 GFLOPS. Most recently, the NAS facility added in 1988 a Cray Y-MP with a peak performance of 2.6 GFLOPS.

By the late 1980s it became clear that the "classical" supercomputer design epitomized by existing Cray systems was approaching maturity, and that highly parallel computer systems had the most potential for achieving multi-TFLOPS levels of sustained performance, which we project will be required before the end of the decade. In order to better understand these systems and to help advance this technology to the level required for mainstream supercomputing, the NAS Applied Research Branch was organized in 1988. Scientists in this organization and its affiliates port application programs to parallel computers, develop new algorithms and programming techniques, and study the performance of the resulting implementations [1, 2, 3, 4, 6, 7, 8, 9, 10 and 12]. The scope and diversity of these projects can be seen by the partial listing in Table 1.

The NAS project now operates a Connection Machine 2 (CM-2) by Thinking Machines, Inc. and an iPSC/860 by Intel Scientific Computers. There are plans to acquire significantly more powerful systems in the future. As of this date, the CM-2 has been operational for nearly three years and the iPSC/860 for over one year. This note gives a brief summary of our experience and highlights both the strengths and the weaknesses that we have observed in these two systems. Requirements and recommendations for future systems are then outlined.

The Connection Machine

The NAS CM-2, obtained in collaboration with the Defense Advanced Projects Research Association (DARPA), was installed in 1988 with 32,768 bit-serial processors. Recently it was upgraded with 1024 64-bit floating point hardware and four gigabytes of main memory, or about twice that of the NAS Cray-2. The system also now has the "slicewise" Fortran compiler, which treats the hardware as a SIMD array of 64-bit floating point processing units.

For the first year or two, most applications on the CM-2 were coded in *LISP, an extension of the LISP language for parallel processing. More recently, however, programmers have started using the CM Fortran language, which is based on Fortran-90 [5]. CM Fortran has been plagued by delays, and version 1.0 was just recently made available.

One notable achievement on the CM-2 is the porting of a variation of the NASA Ames "ARC3D" code. This is a three dimensional implicit Navier-Stokes fluid dynamics application and employs a number of state of the art techniques. On a $64 \times 64 \times 32$ grid, this

Project	Researchers	Y-MP	CM-2	iPSC
Multigrid (NAS benchmark)	Frederickson, Barszcz	x	x	x
Conj. gradient (NAS benchmark)	Schreiber, Simon	x	x	x
3D FFT PDE (NAS benchmark)	Bailey, Frederickson	x	x	x
Integer sort (NAS benchmark)	Dagum	x	x	x
LU solver (NAS benchmark)	Fatoohi, Venkatakrishnan	x	x	x
Scalar penta. (NAS benchmark)	Barszcz, Weeratunga	x	x	x
Block tridiag. (NAS benchmark)	Barszcz, Weeratunga	x	x	x
INS3D (incomp. Navier Stokes)	Fatoohi, Yoon	x	x	x
Isotropic turbulence simulation	Wray, Rogallo	x	x	x
PSIM3 (particle method)	McDonald	x		x
PSICM (particle method)	Dagum		x	
F3D (ARC3D multi-zone)	Barszcz, Chawala, Weeratunga		x	x
CM3D (ARC3D derivative)	Levit, Jesperson	x	x	
ARC2D	Weeratunga			x
Unstructured Euler solver	Hammond, Barth, Venkatakrishnan		x	x
Unstructured partitioning	Simon			x
High precision vortex analysis	Bailey, Krasny			x

Table 1: Overview of Parallel CFD Research at NASA Ames

code now runs at 190 MFLOPS on 16,384 processors [8]. The equivalent one processor Y-MP rate for this code is 128 MFLOPS.

The strongest positive feature of the CM-2, from our perspective, is TMC's adoption of a programming language based on Fortran-90. Some have questioned whether the Fortran-90 array constructs are sufficiently powerful to encompass a large fraction of modern scientific computation. Basically, this is the same as the question of to what extent the data parallel programming model encompasses modern scientific computation. While we cannot speak for other facilities, it seems clear that a very large fraction of our applications can be coded effectively using this model. Even some applications, such as unstructured grid computations, which appeared at first to be inappropriate for data parallel computation, have subsequently been successfully ported to the CM-2, although different data structures and implementations techniques have been required [6].

Another positive feature of the CM-2 is that its system software is relatively stable. It appears to be largely free of the frequent system "crashes" and other difficulties that plague some other highly parallel systems.

One weak point of the CM-2 is that it was not designed from the beginning for floating point computation, and the floating point processors now in the system have been incorporated in a somewhat inelegant fashion. While this has been remedied somewhat with the recent hardware upgrade and the "slicewise" software, the limited bandwidth between floating point processors and the associated main memory modules remains a problem. Related to this is the limited bandwidth between nodes. Even when using the hypercube network, data cannot be transferred between nodes at a rate commensurate with the floating point computation rate. And when the "router" (an alternate hardware facility for irregular communication) must be used, the performance of the code often drops to workstation levels.

Another weakness of the CM-2 design is that until recently no more than four users could be using the system at a given time, which is too restrictive for daytime operation when scientists are developing and debugging their codes. This restriction stems from the fact that the hardware cannot be partitioned any finer than a 8,192 node section. This difficulty has been remedied by the introduction of time sharing software that allows more than one job to be executing in a given sector of the CM-2. However, the down side of this solution is that individual jobs must occupy more processing nodes than necessary, and thus their performance is reduced accordingly.

The principal weakness with the CM-2 at the present time, however, is the CM Fortran compiler, which still has a number of bugs and frequently delivers disappointing performance. Also, the array intrinsic functions are in several cases rather inefficient. Partly this is due to the fact that writing an effective Fortran-90 compiler, including efficient array intrinsics, has proved to be much more of a challenge than originally anticipated. Other vendors attempting to write Fortran-90 compilers are reported to be having similar difficulties. In any event, clearly the CM Fortran compiler must be further improved.

The weaknesses of the current version of the CM Fortran compiler, coupled with the massively parallel design of the hardware, result in a system that is often very sensitive to the algorithm and implementation technique selected. It is not unusual for a reasonably well conceived and well written code to deliver single digit MFLOPS performance rates when first executed. Often the programmer must try quite a number of different parallel algorithms and implementation schemes before finding one that delivers respectable performance. Experienced CM-2 programmers have learned numerous "tricks" that can improve performance, but until recently TMC provided very little documentation available on these "tricks".

The Intel iPSC/860

The NAS Intel iPSC/860 system, which was one of the first two of these systems shipped by Intel, was installed in January 1990. It has 128 nodes, each of which contains a 40 MHz i860 processor (with a 64-bit peak performance rate of 60 MFLOPS) and 8 megabytes of memory. The complete system has one gigabyte of memory and a theoretical peak performance of roughly seven GFLOPS.

Programs for the Intel system may be coded in either Fortran or C, although most scientists have chosen Fortran. At first, the available compilers did not utilize many of the advanced features of the i860, and so single node performance was predictably poor, typically only one or two MFLOPS. More importantly, compile/link times were very long, typically 30 minutes for a 5,000 line program. Recently Intel made available some new Fortran and C compilers that were produced by the Portland Group. While the initial release of these compilers did not feature significantly faster execution speeds, at least the compile/link speeds were much better. Further, the Portland Group's i860 compilers are now available on Sun-4 and Silicon Graphics workstations, and running them off-line in this manner has further reduced compile/link times by more than a factor of ten.

The highest performance rate achieved so far on our iPSC/860 system is 1.6 GFLOPS (32-bit), which has been obtained on an isotropic turbulence simulation program. This code was written in the Vectoral language by Alan Wray and Robert Rogallo and features an assembly-coded fast Fourier transform kernel [12]. This project required approximately two person years of effort, including the time required by Wray to port his Vectoral language to the i860.

One significant advantage of the Intel system is that it is fairly easy to obtain moderately respectable performance using the system. In our experience, almost all codes that have employed reasonably well conceived algorithms have achieved at least 100 MFLOPS (64-bit) on 128 nodes, and many run at significantly higher rates. Another advantage of the Intel system is that the 128 nodes can be decomposed rather flexibly among multiple users for debugging and program development. Currently up to ten subcubes can be allocated, but there is no fundamental reason that more could not be allowed.

One weakness of the current system is that single node performance is disappointing, even with 40 MHz processors and the Portland Group compilers. Most Fortran codes only run at 4 or 5 MFLOPS. Since the peak performance of the i860 is 60 MFLOPS (64-bit), there is ample room for improvement here. However, from our analyses [7] we do not expect more than about 8 to 10 MFLOPS per node on most Fortran codes even with the best

Fortran compiler, due mainly to the somewhat limited bandwidth between main memory and the i860 processor. We have found that the 8 kilobyte on-chip data cache in the i860 is too small to significantly improve performance on most codes. A much larger cache would definitely help, but improved main memory bandwidth would be more valuable for the main body of Fortran scientific codes.

Once the Fortran compiler has been improved, the limited bandwidth of the current interprocessor network will loom as a serious performance bottleneck. Obviously Intel recognizes this problem, and the new Touchstone Delta system just installed at CalTech features a grid network with significantly higher bandwidth. Studies are currently in progress on the Delta system to determine whether this new design has alleviated this performance bottleneck.

Needless to say, programmers are not pleased that only low-level communications and synchronization primitives are available on the Intel system. Programmers who have worked on the CM-2 in particular complain about having to manually decompose arrays, synchronize operations and communicate data. While a Fortran-90 multiprocessor compiler is probably the best long-term answer, work in progress on "Fortran-D" at Rice University and elsewhere may provide programmers with help in the short term.

However, the most annoying drawback of the current system is the instability of the operating system. Even one year after installation, it is not unusual for the system to have to be rebooted three or four times in a single day. These frequent "crashes" are discouraging even to the most determined and fearless programmers. Related to this problem is the equally serious drawback of a weak front end system, which is basically a 386 personal computer. Fortunately, with the new workstation-based compilers, users do not have to compile and link on the front-end system anymore. But this system is clearly not designed to accommodate numerous interactive users, and future versions of the iPSC must include much more powerful service facilities.

NAS Parallel Benchmark Results

It is well known that existing supercomputer benchmarks and benchmarking methodologies are poorly suited for studying the performance of these new highly parallel scientific computers. For one thing, the rigid tuning requirements of many of the well-known benchmark programs pretty well rule out the usage of many widely used parallel programming constructs. One could assume the existence of automatic tools for converting "dusty deck" Fortran codes to run on highly parallel computers, but such tools simply do not now exist. Another problem with many conventional benchmarks is that the problem sizes are inappropriately small for the new highly parallel systems. Finally, most existing benchmarks employ data structures and implementation techniques that are inappropriate for parallel systems.

In an attempt to remedy these difficulties, researchers in the NAS Applied Research Branch at NASA Ames, the NAS Systems Development Branch and the Research Institute for Advanced Computer Science (RIACS) have developed the NAS Parallel Benchmarks [1, 2]. These benchmarks are a collection of eight problems that are completely specified

Benchmark	Problem Size	Y-MP 8	CM-2 32K	iPSC/860 128
Embarrassingly Parallel	2^{28}	1399	659	362
Multigrid	256^3	2706	308	824
Conjugate Gradient	2×10^6	631	104	70
3-D FFT PDE	$256^2 \times 128$	1795	414	696
LU solver	64^3	1705	186	224
Scalar penta. solver	64^3	1822	109	*122
Block tridiagonal solver	64^3	1554	94	*199

Table 2: NPB Performance Results (MFLOPS)

in "pencil and paper" fashion in a technical document [1]. Even the input data is completely specified in this document. Implementations must be based on Fortran-77 or C, but a wide range of parallel constructs are allowed. With a few exceptions, assembly code and assembly language subroutines are not allowed for performing computations, although vendor-supported assembly language subroutines may be invoked to perform communication and synchronization operations. Aside from these restrictions, benchmarkers are free to select algorithms, implementation techniques and language constructs deemed to be the most advantageous for implementing the benchmarks on a given system.

The NAS Parallel Benchmarks have been implemented on the parallel computers at NAS, as well as on other systems by other researchers. While efforts are still being made to optimize these implementations, some results are available, which are shown in Table 2. These MFLOPS performance rates are based on single processor floating point operation counts. The Intel results for the last three lines noted with * denote runs made on only 64 nodes. For explanations of the various benchmarks, see [1] or [2].

When we compare the performance rates that we have achieved with these benchmarks on the parallel systems with the levels achieved on the Cray, the results are somewhat disappointing — the parallel codes typically run at the equivalent of one or two Y-MP processors. When one computes the ratio of sustained performance to peak performance, the results are even more striking: typically 1% to 5% for the CM-2 and the iPSC/860, as compared with 30% to 60% for the Y-MP. It is true that the Cray Y-MP system is significantly more expensive than the parallel systems. But when one computes sustained performance per dollar, the figures for the iPSC/860 and the CM-2 are not dramatically higher than the Cray as one might expect, but instead are roughly on a par with the Cray figures.

It is widely believed in the field of parallel computing that the key to obtaining high performance is to employ "more suitable algorithms" on the parallel computers. "More suitable algorithms" often means comparatively elementary algorithms, such as direct meth-

Solver Algorithm	Floating Point Operations	CPU Time (Secs.)	MFLOPS
Jacobi	3.82×10^{12}	2124	1800
Gauss-Seidel	1.21×10^{12}	885	1365
Least Squares	2.59×10^{11}	185	1400
Multigrid	2.13×10^{09}	6.7	318

Table 3: NCUBE-2 Performance on a Convection-Diffusion Problem

ods for solving PDEs, which require only nearest-neighbor communication and thus give higher MFLOPS performance rates than more advanced algorithms, such as implicit methods, which require long-distance communication. Often overlooked is the fact that these nearest-neighbor algorithms are typically much less numerically efficient than the more advanced algorithms. Although the work of NASA Ames researchers has amply confirmed this principle, it can most vividly be seen from some results due to Shadid and Tuminaro [11] of Sandia National Labs, which are reproduced in Table 3.

Note that while the MFLOPS rate of the Jacobi scheme is nearly six times that of the multigrid-based solver on this problem, the Jacobi scheme requires 300 times as much CPU time to complete the solution. These results and others underscore what should be obvious: the fundamental numerical efficiency of an algorithm is much more important than its appropriateness for a particular architecture. Or in other words, if one employs an algorithm on a parallel computer that has a significantly higher operation count than the best known serial algorithm for that purpose, whatever cost-performance advantage the parallel computer has might well be nullified.

Conclusions

While we are still basically optimistic that highly parallel computers are the wave of the future, it is clear that some improvements must be made from the current designs. First of all, parallel vendors need to recognize that most real scientific floating point computation, and certainly scientific computation at our facility, is characterized by a ratio of floating point operations to main memory references of approximately unity. What this means is that systems cannot rely solely on large register or cache utilization ratios to obtain respectable performance rates, and that improved bandwidth between processors and main memory is essential. It also means that floating point processors must deliver good performance on computations with nonunit strides.

Secondly, it is an unpleasant fact of the state of the art in computational fluid dynamics, as well as in many other numerical applications, that the demanding problems that have the most research interest involve implicit numerical solvers or other schemes that require substantial long distance communication. Also, arrays often must be accessed in each of three dimensions. In short, the data communication patterns of advanced algorithms, both within a single node and between nodes, are typically nonlocal. The message for parallel computer designers is clear: these systems must be designed with sufficient internode bandwidth, among other things, so that advanced implicit algorithms, as well as other numerically efficient algorithms, can be effectively executed.

Thirdly, we are now convinced that for a majority of all scientific applications, and for a large majority of our applications, the Fortran-90 language [4] (particularly the Fortran-90 array constructs) will be the language of choice, and vendors must support this language for multiprocessor computation, not just on a single node. Many scientists have told us that their chief reservation of porting codes to parallel computers is the prospect that their code will no longer run on other systems. Once Fortran-90 is officially adopted, which is now expected soon, scientists should be much more open to recasting their codes into this language. Already Cray, for one, has enhanced its Fortran compiler to accept many of the Fortran-90 array constructs, and scientists at our facility now use the Crays in their efforts to port codes to the CM-2.

It may be too much to expect that future parallel computer systems will deliver respectable performance from arbitrary Fortran-90 programs. For example, it is questionable that interconnection networks of future systems will have the high bandwidth and low latency required to support intensive, long distance computation along each dimension of a three dimensional array code. However, in our opinion it is reasonable to expect that a parallel computer system should deliver respectable performance on codes where an honest effort has been made to reduce interprocessor data traffic. As an illustration, consider the following design for one iteration of a three dimensional scientific computation:

1. Perform numerical computations, using array constructs, along the first dimension, which is mapped to be within local nodes.

2. Perform an array transposition, using the Fortran-90 RESHAPE operator, so that the second dimension is now the first dimension.

3. Perform numerical computations along the first (i.e. the second) dimension.

4. Perform an array transposition so that the original third dimension is now the first dimension.

5. Perform numerical computations along the first (i.e. the third) dimension.

6. Perform an array transposition back to the original array ordering.

One feature of the above design that is common to many three dimensional scientific programs is that it features two dimensional parallelism. For a $100 \times 100 \times 100$ problem, this means 10,000 way parallelism. In our opinion, if a parallel computer system cannot deliver respectable performance on a problem of this size with two dimensional parallelism, but instead requires the programmer to also find parallelism in the third dimension, then its usability for a broad range of real scientific problems will be severely limited.

We certainly do not claim that the above scheme is most efficient for all three dimensional scientific applications. Even for our CFD programs, other schemes are often more efficient. Nonetheless, it seems clear that if a parallel computer system cannot deliver respectable performance on an application coded according to this design, then it is doubtful that it will be able to deliver very much better performance on that application no matter how it is coded. In short, we regard this as a minimum requirement of a usable parallel computer.

While Fortran-90 appears to be a satisfactory means of expressing the main body of data parallel, single program multiple data (SPMD) computations, it should be emphasized that some important algorithms and applications do not fit well into this model. Some of these that we have identified include "chimera" schemes for unstructured grid computations, computations over complicated geometries (such as complete aircraft configurations), and domain decomposition schemes. Additional research is needed to find the best language constructs for handling truly independent, asynchronous computations such as these.

Although a high level of sustained performance on real scientific applications, coded with reasonable effort, is the most important consideration in a parallel computer, it is clear from our experience that other aspects of these systems also need to be improved. For example, if parallel computers are ever to be widely used in scientific computation centers, then they must be capable of handling a fairly large number of interactive users (say 50), especially during daytime hours when users are debugging and upgrading their codes.

While it is possible to support multiple running jobs by means of time sharing, as TMC has attempted to do, it still seems preferable for performance reasons (as mentioned above) to be able to assign individual jobs to separate hardware subdomains. Such a hardware design may also be an advantage in the future, when scientists tackle large multidisciplinary applications. Whichever scheme is used, the system must provide reasonable security against users storing data into nodes or memory locations that they do not own.

A related issue is mass storage. While both the current CM-2 and iPSC/860 systems have moderately fast and high capacity mass storage systems, it is clearly important that these features greatly improve with future systems. Also, it is essential that future parallel computer systems provide a high performance network interface (such as the HiPPI interface) to allow high speed data communication to workstations and archival storage.

We recognize that the development of high performance, highly usable parallel computer systems will be a challenging task. But we feel that without major improvements in both hardware and software, there is a risk that scientists will eventually tire of struggling with these systems and will return to conventional workstations and vector computers, and the development of parallel computing technology will be greatly retarded.

Acknowledgment

The author wishes to acknowledge helpful comments and suggestions by Eric Barszcz, Leo Dagum and Tom Lasinski of the NAS Applied Research Branch, by Rod Fatoohi, Horst Simon, V. Venkatakrishnan and Sisira Weeratunga of Computer Sciences Corporation, and by P. Frederickson and R. Schreiber of the Research Institute for Advanced Computer Science (RIACS).

References

1. Bailey, D. H., Barton, J., Lasinski, T., and Simon, H., "The NAS Parallel Benchmarks", Technical Report RNR-91-002, NAS Applied Research Branch, NASA Ames Research Center, January 1991.

2. Bailey, D. H., et al., "The NAS Parallel Benchmarks", *International Journal of Supercomputer Applications*, 1991, to appear.

3. Bailey, D. H., et al., "Performance Results on the Intel Touchstone Gamma Prototype", *Proceedings of the Fifth Distributed Memory Computing Conference*, April 1990, p. 1236 – 1245.

4. Barszcz, E., "One Year with an iPSC/860", Technical Report RNR-91-001, NAS Applied Research Branch, NASA Ames Research Center, January 1991.

5. *Fortran 90*, Draft International Standard, American National Standards Institute, June 1990.

6. Hammond, S., and Barth, T. J., "An Efficient Massively Parallel Euler Solver for Unstructured Grids", RIACS Technical Report 90-47, NASA Ames Research Center, October 1990. Also published as AIAA paper 91-0441, 29th Aerospace Sciences Meeting, January 1991.

7. Lee, K., "On the Floating Point Performance of the i860 Microprocessor", Technical Report RNR-90-019, NAS Applied Research Branch, NASA Ames Research Center, October 1990.

8. Levit, C., and Jespersen, D., "Numerical Simulation of Flow Past a Tapered Cylinder", Technical Report RNR-90-021, NAS Applied Research Branch, NASA Ames Research Center, October 1990. Also published as AIAA paper 91-0751, 29th Aerospace Sciences Meeting, January 1991.

9. McDonald, J. D., "Particle Simulation in a Multiprocessor Environment", Technical Report RNR-91-003, NAS Applied Research Branch, NASA Ames Research Center, January 1991.

10. Schreiber, R., "An Assessment of the Connection Machine", RIACS Technical Report 90-47, NASA Ames Research Center, June 1990.

11. Shadid, J. N., and Tuminaro, R. S., "Iterative Methods for Nonsymmetric Systems on MIMD Machines", *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, 1991, to appear.

12. Wray, A., personal communication.