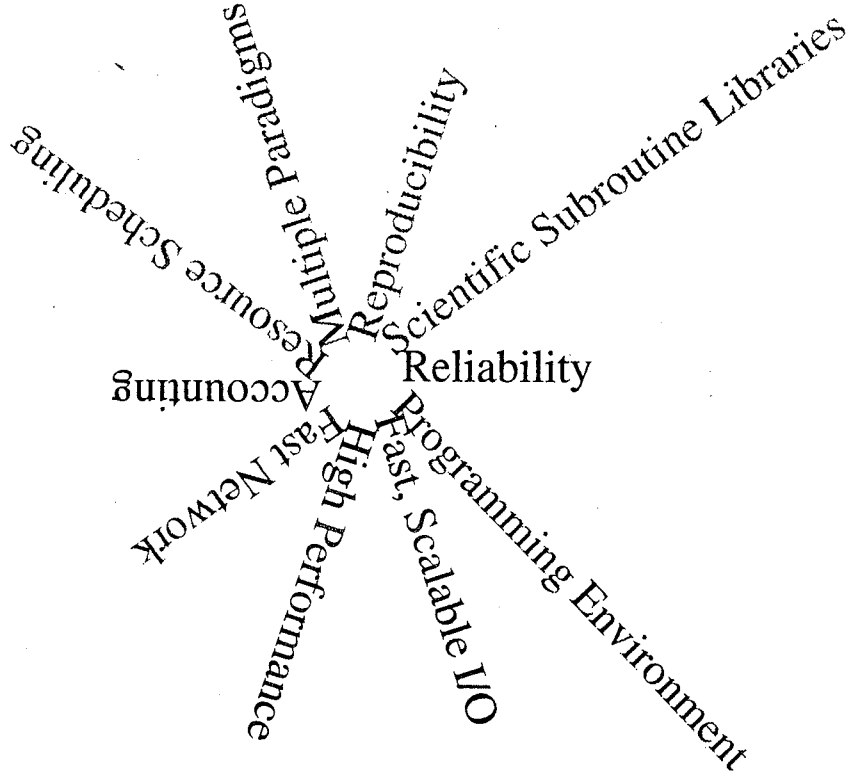
NCAR/TN-413+STR
NCAR TECHNICAL NOTE

January 1995

Beyond Machoflops: Getting MPPs into the Production Environment

Steven W. Hammond



SCIENTIFIC COMPUTING DIVISION

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO

Beyond MachoFlops: Getting MPP's into the Production Environment

Steven W. Hammond

National Center for Atmospheric Research
Scientific Computing Division
P.O. Box 3000
Boulder, CO 80307

February 24, 1995

Abstract

In the continuing pursuit of the highest performance, many parallel computer manufacturers are trying to harness the promise of "killer micros" to deliver higher and higher computational rates to supercomputer users. The emphasis is on reducing clock cycles, improving hardware architectures, and increasing the number of floating point operations per second a system can deliver on benchmarks and kernels. However, there is more to a computer *system* than hardware. What has been lost in this pursuit is the same emphasis on providing the software that makes such systems both usable and manageable. The software fabric (libraries, utilities, programming environment, etc.) is no less important than the underlying hardware. This software aspect of a computer system is important at NCAR and many other research organizations because there are many more scientists that compute than computer scientists. A robust software environment has a tremendous impact on the ease of developing, debugging, and optimizing large codes as well as the ability to administrate such a system. Here we detail a "Top 10 List" of requirements necessary to successfully integrate massively parallel processing systems into a production computing environment, a domain currently dominated by vector supercomputers.

Introduction

Vendors of Massively Parallel Processors (MPPs) have put significant effort into the processors and interconnection networks of their parallel computers. They boast about their computer's latest performance on benchmarks and kernels such as the NAS Parallel Benchmarks[1] or LINPACK[4, 5]. However, such benchmark numbers are typically viewed as levels of performance that are guaranteed not to be exceeded and generally unobtainable by all but a very few programmers who tediously optimize their code for that machine and that machine alone. High computation rate is important but there is more to a computer system than its ability to execute kernels. Very few vendors will argue successfully that their MPP is as easy to program as a workstation or that a large code developed on a workstation will run on their machine, as is, by just recompiling. Other issues of concern are reliability, manageability, and scalability. When was the last time an MPP vendor was able to claim that their systems achieve 95% or more uptime like traditional vector supercomputers are expected to sustain? Some aspects of MPPs scale in a desirable fashion and others you want to be held constant. For example, what good is a window-based debugger on a parallel system with 512 processors if it requires a window per process or processor? Interfaces for tools and utilities should be independent of the number of processors.

The traditional definition of a *balanced* MPP has been a computer system with sufficient interprocessor bandwidth and memory bandwidth to supply data to the computational units and keep them busy a high percentage of the time. This type of balance is important but it only refers to the hardware. What about the software? A truly balanced system has a balanced hardware architecture *and* software that makes it usable.

Another contribution to the hardware emphasis is the way we compare MPPs. One metric that has been used to evaluate supercomputers is the purchase price per unit of delivered performance – dollars per MFlop for instance. However, this only tells part of the story since the cost of using and supporting a computer is much more than the purchase price. There are costs associated with having or not having each of the elements in the software fabric of a computer system. In some cases, the post-purchase cost associated with a computer system that fails to supply certain capabilities can exceed the initial price of the system itself. Suppose that the processors in a parallel computer are very fast but the system requires rebooting once or more daily. Or, suppose that the users are required to substantially rewrite their codes to achieve high performance on the computer. There are costs associated with each of these and they are not captured in the appealing but simplistic price/performance ratio above. A more comprehensive metric would be some sort of life cycle cost of the machine. One possibility is the sum of the initial purchase price or total lease cost, the software and hardware maintenance costs, and some estimate of the cost to rewrite codes (if

any). This sum could be divided by the performance of the machine scaled by the average percent uptime required or promised by the vendor.

The following sections detail a list of ten requirements necessary to integrate massively parallel processing systems into a production computing environment. They are listed roughly in increasing order of importance although different people might have different emphasis or requirements.

10. Accounting

In a production environment where resources are shared by a wide variety of users accurate accounting capabilities are very important. Vendors must supply accounting packages that give accurate information for each job run by each user (batch and interactive). Accounting information is necessary to make sure that all users can get their share of allotted resources and/or to prevent the resources from being monopolized by an individual or a small fraction of the users.

The accounting system should monitor the resources used by UNIX processes, interactive sessions, and batch jobs. The accounted resources should include:

1. the total wall clock time, total user CPU time, total operating system CPU time, total I/O wait time used by the process, interactive session, or batch job.

2. the maximum amount of main memory used by the processes, interactive session, or batch job.

3. sufficient information to compute a monthly total of CPU time used by each user.

4. accounting information sufficient to report the amount of disk space used by each user.

9. Resource Scheduling

In addition to accounting, fair access to system resources includes setting job priorities and time sharing of resource among multiple jobs. This includes batch processing based on a system like the Network Queuing System (NQS) or a functionally equivalent batch processing facility. Additionally, the system's ability to present a single system image to batch jobs and interactive jobs is paramount. It should distribute the submitted jobs across the processors as a balanced load. Finally, the user should be able to use directives or commands to direct batch jobs to a specific group of processors or partition. The sizes of groups or processors or partitions should not be

restricted to integer powers of two. If an application maps efficiently onto 65 processors the user should not be required to allocate 128 processors and let 63 of them sit idle.

The batch processing system should provide a checkpoint command which records the state of an executing job on disk, and a restart command which resumes the execution of a previously checkpointed batch job. These commands shall allow operators to preserve the execution of batch jobs across the MPP without requiring the use of special programming techniques or user intervention.

During different modes of operation, the hardware and software must allow for the partitioning of resources to allow users to allocate one or more computational nodes. Being able to reconfigure or repartition the processors without rebooting the system is also highly desirable.

8. Reproducibility

The primary mission of NCAR is to perform scientific research using large production geoscience programs such as the Community Climate Model (CCM2) [2, 8]. Long term climate simulations have a critical need for reproducibility of results, requiring reproducible computations independent of the number of processors that the code is run on. Put another way, on the same MPP, one should be able to run same program on different numbers of processors and get the same results, bit for bit. This is a must for validation of a new model against an existing model [13] and for debugging. Often a user detects an error on a large partition but for debugging purposes will request a smaller number of processors. If a program produces different results on different numbers of processors it complicates the debugging process to the point of it being impossible at times.

7. Reliability

Some aspects of an MPP scale in a desirable fashion; reliability is not one of them. The mean time between total system failures should be a constant. System availability should be at or above 95%, independent of the number of processors in your system.

Computer manufacturers need to start delivering systems with redundancy in their hardware configurations or provide for the elimination of single point failures that render the whole system unusable. The ability to detect and correct error conditions and the ability to operate in a degraded mode is increasingly important as supercomputers increase in memory size, numbers of processors, etc. Systems should be able to operate with one or more computational nodes disabled, and to do so

with minimal performance degradation. The same holds true for interprocessor communication components. The system should be able to tolerate failures by graceful dynamic degradation of communication and computation performance.

Fault tolerance is still a very difficult problem. It has to be designed into a system from the very beginning. The solution requires both hardware and software working in concert and different levels of solution have different costs. Redundant hardware, each doing the same computation, is perhaps the most expensive. Another possibility is for a system to have hot spares ready to be swapped in upon detection of a fault. The system may have to abort all running jobs, reconfigure, send out notification of the event, and then start computing anew on pending jobs. Either of these two solutions or something between is better than having the whole system unavailable due to a single point failure. Plus, it is not sufficient to simply be able to detect and recover from minor faults, it should be done rapidly as well.

Another often overlooked aspect of reliability is the time to diagnose a hardware error and reboot an MPP with a large number of processors. There are numerous anecdotal stories about vendors assembling their first large system only to realize that their diagnostics and boot sequence are strictly sequential and take a long time to execute because they were developed on small, single processor test systems.

6. Robust Scientific Subroutine Library

Software libraries are important tools in the use of computers in general and even more so in the use of MPPs. They must work on data local to a single processor as well as data distributed across processors. Therefore the libraries must be available as parallel routines and not just offered as serial entities.

Libraries enhance portability and reliability of computer programs. MPPs might consist of thousands of processing elements each with its own local memory. The processors are typically interconnected with a network that has characteristics such as throughput and latency unique to each vendor and to each generation of machines from the same vendor. The performance optimization for each different system requires special attention to data allocation, data motion, memory hierarchies, and load balancing. Additionally, fundamental changes to classical algorithms may be required to take advantage of unique architectural features. All of this machine specific programming can be captured in a robust scientific subroutine library thus insulating the details from the user.

Libraries for scientific and engineering applications embody expert knowledge of many disciplines including algorithms, computer architecture, applied mathematics, and numerical analysis. Additionally, they enhance productivity not only by providing optimized preprogrammed functions, but, by providing functionality with well

known and documented storage requirements, execution time, and numerical behavior. Libraries make the architecture of computers more transparent to a user than programming languages by defining functions at a sufficiently high level for optimization beyond the capabilities of compilers. The portability of user programs is greatly enhanced with respect to both performance and numerical behavior. And finally, good libraries substantially lower the cost of computation and improve user productivity and program efficiency.

All vendors who cater to scientific markets should provide a scientific subroutine library (SSL) which offers a palette of optimized scientific subroutines, containing at least BLAS and LAPACK, and hopefully FFTs and others. Even vendors who do not offer turnkey parallel systems should offer a comprehensive SSL, because of the potential for clustering, as has been evidenced in the workstation industry. A good example of a rich SSL for an MPP is the Connection Machine Scientific Subroutine Library (CMSSL) supplied by Thinking Machines Corporation [3]. It contains routines for solving linear systems of equations – dense, sparse, and banded matrices, using direct and iterative methods. It also provides utilities for computing fast Fourier transforms, solving ordinary and partial differential equations, and routines for linear programming and random number generation. Finally, it includes functions for communication primitives such as polyshift, all-to-all rotation, matrix transpose, and gather/scatter. The vendor has optimized this software for its computer systems. Similar capabilities should be offered by other vendors.

5. Programming Paradigms

NCAR is aware that a number of programming models exist for supercomputers. Unfortunately there has been no significant convergence of programming paradigms for parallel machines. Therefore, in the short term MPP vendors should support a variety of them. The most common models are shared memory programming (a paradigm in which performance is mostly independent of data layout due to roughly uniform memory access times), data parallel programming (defined by High Performance Fortran [11] as a model characterized by single thread execution, a global name space, and is loosely synchronous), and explicit message passing (using constructs such as those available in PVM or MPI). The programming models should be supported in both FORTRAN and C and should be interoperable, meaning that they can be mixed within a single source program.

The programmers and scientists at NCAR who provide production simulation codes also wish to minimize the time spent converting large FORTRAN codes from existing Cray Research Y-MP and Thinking Machines Corporation CM-5 machines to future machines. Upward-compatible software that supports reusability and avoids

costly reprogramming and retraining unless there is significant gain in user/application productivity is highly valued.

At NCAR we currently maintain three different versions of CCM2 – a version that is parallelized using multitasking on the Cray Y-MP, a message passing version (PVM) that is decomposed by latitudes for parallel computing, and a data parallel version for the CM-5. The holy grail for us with respect to this large production code (approximately 40000 lines of Fortran) is to have one version that runs efficiently on workstations, on traditional vector supercomputers, and on MPPs. All that should be required is for the user to recompile one code specifying the target machine. Although we recognize that HPF is not a panacea, it seems to offer the best hope for maintaining a single unified version of the code. Until vendors are able to offer optimizing HPF compilers we are forced to expend considerable time and effort in maintaining the three CCM2 versions required to run on the variety of machines we have in house. Explicit message passing is not viewed as a long term solution to our programming needs.

4. Fast Interconnection Network

A high bandwidth, low latency network is critical to a scalable MPP. But, the network needs more than just good point-to-point speed. An MPP must also have a good *bisection bandwidth* which scales with the number of processors. The bisection bandwidth is the minimum number of interprocessor connections that must be cut to bipartition the processors into two equal size groups, for all possible bipartitions. This is important for fast matrix transposes and other all-to-all data permutations.

An example of why this is important comes from our experience with porting NCAR's Community Climate Model (CCM2) to the CM-5 [9, 10, 12]. For efficiency of the data parallel code we dynamically map data to processors to minimize interprocessor communication at different points during each time step. The choice of the mapping of data to processors is critical for efficient execution on any MPP system. In the adiabatic dynamics section of the climate model, CCM2 makes use of the spectral transform method which requires that state information be transformed from grid point space to spectral space and back again each time step. We use a forward FFT to convert to Fourier space and then a forward Legendre transform to spectral space. We find it most efficient to transpose the data so that each transform is computed in-processor, in parallel, rather than using distributed transform algorithms. The Fourier transform is performed in parallel in the longitudinal direction and then the Legendre transform is parallelized in the latitudinal direction. We pay the communication penalty all at once in the transpose and then use efficient serial algorithms (running in parallel on each processor) for each transform. We have found

that on the CM-5 the transposition overhead is small compared to the time savings realized during the transforms. Additionally, transposes have a regular communication pattern and thus can be optimized on most MPP's [6, 7]. Networks such as the 3D torus, the hypercube, and the fat tree have relatively large bisection bandwidths compared to rings and 2D grids.

Finally, as mentioned in Section 7, the system should be able to tolerate failures in its communication network by graceful dynamic degradation of communication performance.

3. Fast Per-node Performance

The focus of this paper thus far has been on software and not on high performance. The importance of raw speed should be kept in perspective but it cannot be denied. If one takes the time to get a code to run well on an MPP there must be some advantage in performance over what can be achieved with relatively little pain on traditional vector supercomputers. Currently we achieve approximately 7 Mflops per processor for our climate code on a 32 processor CM-5 and on a 64 processor Cray T3D. The sad reality is that this is about 5% of peak performance on these machines. It is hardly motivation for users to move their codes off of our vector machines.

Our experience is not unique and can be attributed to a number of different factors. Many MPPs are often built from cache-based commodity micro processors. When one tries to optimize a computational kernel for these systems one finds that there is a very small "sweet spot" within which the code performs well and outside of which it does not. The micro processors are very sensitive to loop order and memory access patterns. As Cray Research has learned from the T3D, an 8Kb data cache is clearly inadequate. Also, getting good performance from a whole code is much more difficult than focusing on kernels. Stepping through large arrays tends to limit the amount of cache reuse. In effect, each memory reference ends up being a cache miss.

Another problem is slow floating point divide. The T3D processor can perform a floating point addition each clock cycle but it takes at least 60 clock cycles, non-pipelined, to compute a floating point divide. (There is no malice toward Cray Research; the T3D just happens to be a familiar machine and is only used for illustration.)

Finally, there is the power-of-two restriction. Although this is not truly a hardware problem, many MPPs operate most efficiently when array axes are dimensioned to be a power of two. Some programming paradigms/vendors have even required arrays to be declared in this fashion. Any deviation from this and the code suffers a factor of two performance reduction per deviant axis. Vendors need to remove this shackle from the users

2. Scalable, Fast I/O

As mentioned in Section 9, the hardware and software must allow for the partitioning of resources to allow users to allocate one or more computational nodes. The vendors should provide a single logical file system which can be accessed from these partitions. The entire disk system should be accessible to every computational node, and be available to every running process and collection of processes for expeditious storage and retrieval. It should include hardware supported by software to minimize error rates and to support logging of errors for performance and error analysis.

Additionally, throughput should scale with the number of processors in the system. As a programmer uses more processors the rate at which he is able to compute increases and the rate at which results are produced increases. Therefore, there is the need for higher I/O rates that scale with the number of processors.

Independent of the programming paradigm, the user should be able to read from or write to a single image. Output should be machine size independent. A file written from a program running on one size partition should be transparently read by a program running on a different size partition.

1. Code Development Environment

Perhaps the most critical aspect (from a programmers standpoint at least) and certainly the most neglected aspect of MPPs is the code development environment. This includes debuggers, performance monitoring, and documentation. Ideally, the environment for executing on a workstation, on an individual node of an MPP, or on the entire system of an MPP should be functionally identical.

The vendors should provide an interactive symbolic debugger. It is most helpful if it is based on dbx since most programmers know dbx and this familiarity reduces the learning curve. The debugger should support the following features:

1. The ability to reference and manipulate all the data of an application distributed across multiple computation nodes.

2. The ability to set global conditional and unconditional break points at source-code lines. When breakpoints are reached, it should be possible to symbolically examine memory and to continue execution with a minimal degradation to performance.

3. The debugger should include both line mode and a graphical user interface.

4. One should be able to represent data in graphical form as well as ASCII.

5. The user should be able to specify the number of significant digits displayed in floating point numbers.

6. The ability to report the minimum and maximum value in an array or array section. This feature is very useful is when comparing runs on two different machines for code validation.

Finally, requiring the user to manage a window per processor on a large MPP is clearly a losing proposition.

The vendor should supply performance monitors that document time spent in computation and interprocessor communication. This is often difficult because commodity microprocessors are not typically built with hardware monitors. The monitor should give relative information as well as absolute performance. In many instances it is sufficient to know where a code is spending most of its time rather than knowing where it spends all of its time. This allows the programmer to focus on critical sections. Additionally, the performance monitor should be a low overhead process and not significantly reduce the performance of the code being studied. The final performance tuning is often done on a full code and if the overhead is too high (a factor of 5 or more) then it may not be feasible to run a code and collect the statistics necessary. Another problem is that performance monitors often collect data on a per processor basis and dump it into files. This is another feature that should not scale. It is easy to overwhelm a file system with performance traces collected from a large MPP when there is a file dumped per processor.

The vendor should supply on line hypertext documentation for all manuals. Many programmers and scientists work at home and do not have all the documentation they need with them. Another necessary feature is general keyword search capability. On line documentation is great for browsing but only if you know where to look. The documentation system should permit the user to enter a key word or phrase and search all documentation or a subset of the documentation for its instances and summarize what was found. The user should then be able to "jump" to the part of interest using the cursor and mouse to make the selection.

Finally, the speed of compiling, linking, and loading a large code is critical to the code development effort. For example, compiling the data parallel version of CCM2 from source on the Sun compile server (a Sparc2 with 64Mb of memory) to the CM-5 at NCAR takes approximately three hours using CMAX version 2.0 final and CM Fortran version 2.1.1-2. Any significant changes to the code were made only after careful consideration because of this slow procedure. Recently we upgraded to a faster compile server (Sparc20 with 256Mb) and it now takes only 24 minutes.

Conclusions

We have given a list of requirements that we feel, vendors must meet to get MPPs into the production environment. The list ranges from accounting software and programming paradigms to I/O and program development environment. For MPPs to get beyond the point where they are viewed predominantly as "research machines" and be placed into the production environment, the vendors must start providing the necessary software fabric that makes an MPP a computer *system.* Otherwise, parallel computing will continue to be the technology of the future and not the technology of today.

Acknowledgements

The author wishes to thank Rich Loft and Dick Valent for their insightful comments and suggestions. This work was sponsored in part by the National Science Foundation.

References

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. *The NAS Parallel Benchmarks* RNR Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, March 1994

[2] L. M. Bath, J. R. Rosinski, and J. Olson. User's guide to (CCM2). Technical Report NCAR Technical Note/TN-382+IA, Climate and Global Dynamics Division, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307, 1992.

[3] *CMSSL for CM Fortran: CM-5 Edition, Volumes I and II,* Thinking Machines Corporation Manual, Version 3.1, June 1993.

[4] J. J. Dongarra, *The LINPACK Benchmarks: An Explanation* Supercomputing, Spring 1988, pp. 10-14.

[5] J. J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment.* TR MCSRD 23, Argonne National Laboratory, March 1988.

[6] A. Edelman, S. Heller, and S. L. Johnson, *Index Transformation Algorithms in a Linear Algebra Framework.* Thinking Machines Technical Report TMC-223, April 1992.

[7] P. M. Flanders. A Unified Approach to a Class of Data Movements on an Array Processor, *IEEE Trans. of Comp.* Vol. C-31, No. 9, September 1982, pp 809-819.

[8] J.J. Hack, B.A. Boville, B. P. Briegleb, J.T. Kiehl, P. J. Rasch, and D. L. Williamson. Description of the NCAR Community Climate Model (CCM2). Technical Report NCAR Technical Note/TN-382+STR, Climate and Global Dynamics Division, National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307, 1993.

[9] S. W. Hammond, R. D. Loft, J. M. Dennis, and R. K.Sato. A Data Parallel Implementation of the NCAR Community Climate Model (CCM2) *to appear in Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, (SIAM, 1995).

[10] S. W. Hammond, R. D. Loft, J. M. Dennis, and R. K.Sato. Massively Parallel Atmospheric Modeling at NCAR *to appear in Parallel Supercomputing in Atmospheric Science*, (World Scientific, 1995).

[11] High Performance Fortran (HPF) Language Specification, Version 1.1. Center for Research in Parallel Computing, PO Box 1892, Rice University, Houston, TX 77251, November 10, 1994.

[12] R. D. Loft and R. K.Sato. Implementation of the NCAR CCM2 on the Connection Machine, *Parallel Supercomputing in Atmospheric Science*, G. R. Hoffman and T. Kauranne, ed., (World Scientific, 1993), 371-393.

[13] J. M. Rosinski and D.L. Williamson. On the Accumulation of Rounding Errors in a Global Atmospheric Model, *submitted to Monthly Weather Review*, NCAR Ms. 0301/94-05, June 1994.