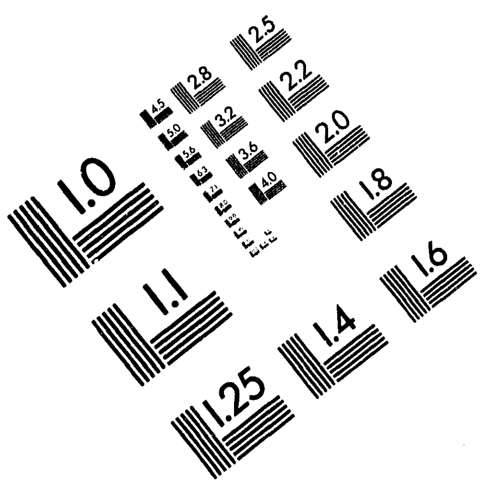
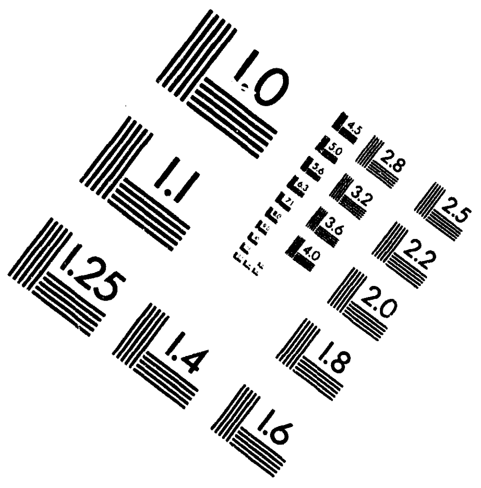


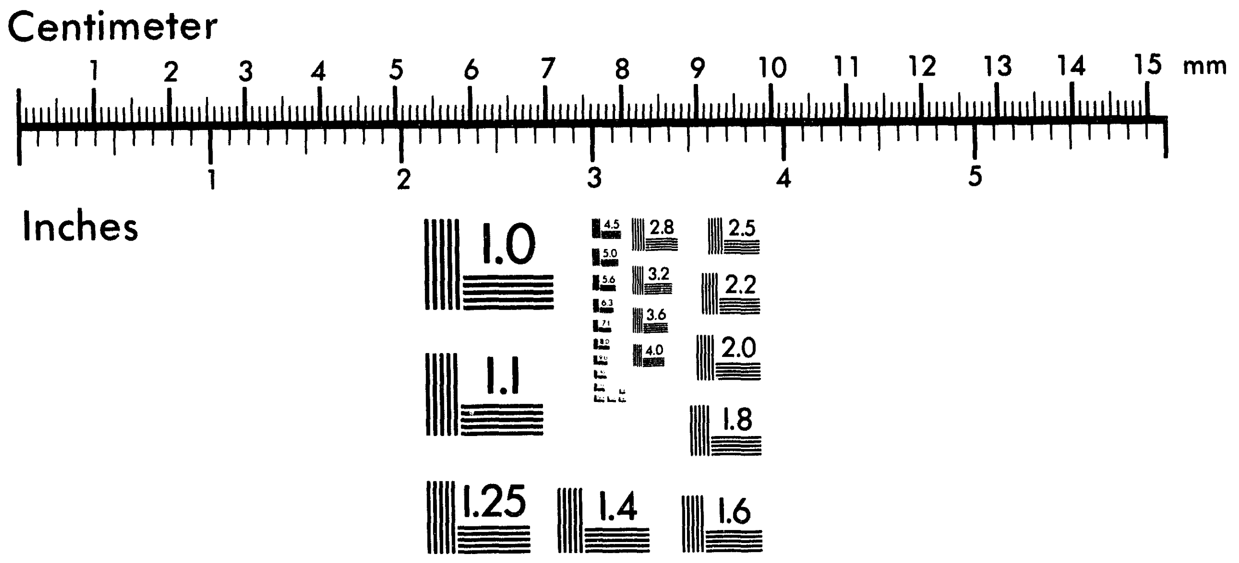


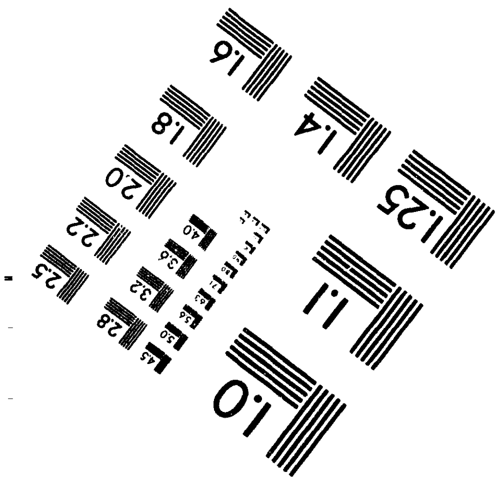
AIIM

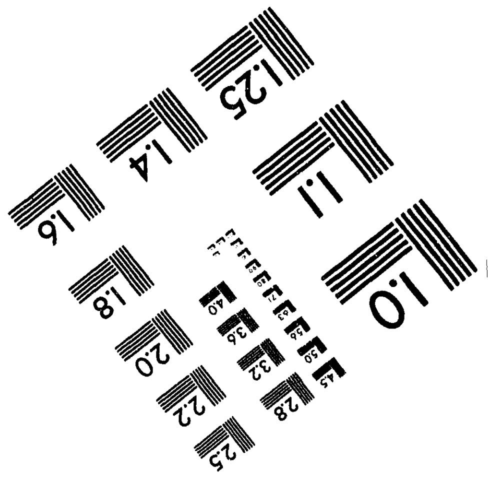
Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

301/587-8202







MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

2

Conf-941142--17

SAND94-0985C

PERFORMANCE ISSUES FOR ENGINEERING ANALYSIS ON MIMD PARALLEL COMPUTERS[1]

H. Eliot Fang
Computational Physics and Mechanics Department

Courtenay T. Vaughan
Parallel Computing Science Department

David R. Gardner
Parallel Computational Sciences Department
Sandia National Laboratories
Albuquerque, New Mexico

ABSTRACT

We discuss how engineering analysts can obtain greater computational resolution in a more timely manner from applications codes running on MIMD parallel computers. Both processor speed and memory capacity are important to achieving better performance than a serial vector supercomputer. To obtain good performance, a parallel applications code must be scalable. In addition, the aspect ratios of the subdomains in the decomposition of the simulation domain onto the parallel computer should be of order 1. We demonstrate these conclusions using simulations conducted with the PCTH shock wave physics code running on a Cray Y-MP, a 1024-node nCUBE 2, and an 1840-node Paragon.

INTRODUCTION

Several studies have shown that distributed-memory parallel computers can deliver higher performance than conventional vector supercomputers for complex scientific and engineering application codes (see, for example, Gustafson *et al.* (1988), Robinson *et al.* (1991), Gardner *et al.* (1992)). Few studies have considered parallel computers from the point of view of an engineering analyst, whose primary concern is solving an engineering problem with sufficient resolution to yield useful information in a reasonable time.

In this paper we consider parallel computer performance from the point of view of the engineering analyst, and explore the factors under the analyst's control for improving the performance of Multiple-Instruction, Multiple-Data (MIMD) parallel computers. We assume that the code has already been optimized and the analyst wants to run the optimized code to get its best performance on a MIMD parallel computer.

[1]This work was performed at Sandia National Laboratories supported by the U.S. Department of Energy under Contract DE-AC04-94AL85000.

In a computer with a MIMD architecture, a collection of sophisticated processors (usually tens or hundreds) execute the same or different instructions on the data to which they have access. For distributed memory MIMD computers, such as the Intel Paragon, the work performed by the processors is coordinated by explicitly passing messages from one processor to another. In shared memory MIMD computers, such as a Cray Y-MP, the work performed by the processors may also be coordinated through the shared memory. Usually each processor has its own operating system and its own copy of the instruction set. Each processor may be executing the same instruction at the same time. More commonly, however, each processor executes instructions independently of the others, and then synchronizes its execution with other processors at various times via the passing of messages, such as the global determination of a time step in a transient dynamics code. More generally each processor in a processor set could be executing an entirely different program from the other processors in the set. For example, in an eight-processor set, four processors might be devoted to performing an engineering finite element analysis, while the other four might be devoted to forming graphical images of the analysis in parallel with the computation. Networks of workstations can also be made to function as MIMD computers.

The performance of parallel computers is commonly measured using several metrics. The peak theoretical speed is often cited. The results from the LINPACK benchmarks (Dongarra, 1994) or the NAS Parallel benchmarks (Bailey *et al.*, 1991) are more indicative of the performance which may be achieved by applications codes.

The LINPACK benchmark codes perform a factorization of a dense matrix A into a lower triangular matrix L and an upper triangular matrix U, such that A = LU. This factorization, called an LU factorization, is used in solving dense linear systems of equations of the form $Ax = b$. The benchmark uses standard LINPACK (Dongarra *et al.*, 1979) routines in full-precision (64-bit) arithmetic in a Fortran environment (there is also a set of benchmarks for the C programming language) to perform an LU factorization. The benchmark consist of several tests. The first is for a matrix of order 100 using a prescribed Fortran program. The second test is for a matrix of order 1000 using any code. The third test is to factor the matrix of largest possible order using any code on a parallel computer. The full LINPACK benchmark results for a computer consist of the time required to complete each test and also include the theoretical peak speed of the computer, which represents the upper bound on machine performance. The most commonly cited LINPACK benchmark results provide an achievable upper bound for speed on problems involving the solution of dense linear systems by LU factorization; achieving the benchmark results often requires the use of special machine configurations and highly optimized assembly code or other resources not normally available to the engineering analyst.

The NAS Parallel benchmarks are a collection of eight problems designed to study the performance of parallel supercomputers. They consist of five kernels, emphasizing a particular type of numerical computation (e.g., fast Fourier transforms), and three simulated computational fluid dynamics applications. The benchmarks are specified functionally, independent of the details of implementation, with specified operation counts. While the times required to complete the NAS Parallel benchmark tests provide a more realistic assessment of the performance of a parallel computer, the tests are still highly idealized compared to real applications codes.

The intent of our work reported here is to report performance results for an application code running a realistic simulation on several high-performance computers. In this work we demonstrate that while a MIMD parallel computer may provide greater computational speed than a vector supercomputer, it must also have sufficient memory capacity to provide equal or greater resolution. A simulation with less resolution obtained more quickly may not be as useful to the analyst as a simulation with greater resolution. We also demonstrate that when a parallel computer has enough memory, simulations of greater resolution can be obtained in less time than with a serial vector supercomputer. In addition, we demonstrate the scalability of the parallel computers for the PCTH code for one through 1840 nodes. By scalability we mean that the execution speed of the code running a specific problem on a parallel computer increases (or, equivalently, the execution time per computational cell per time step decreases) linearly with the number of nodes when the computational load per node is fixed. Finally we demonstrate that naively using more nodes on a parallel computer may not always result in solving a problem faster or with greater resolution and explain how to avoid this situation.

In the following sections we describe the computing hardware we used, the performance metrics we used, the PCTH shock wave physics code, and the test problem. Then we present and discuss our performance results and their significance and finally present our conclusions.

DESCRIPTIONS OF THE COMPUTERS

In this work we compare the performance of an eight-processor Cray Y-MP vector supercomputer, a 1024-processor nCUBE 2 MIMD parallel computer, and Sandia's 1840-processor Paragon X/PS MIMD parallel computer. We briefly describe each machine below.

Sandia's Cray Y-MP (manufactured by Cray Research Inc.) has eight processors, each with a 6.0- nanosecond clock, and a shared-memory architecture; every processor has uniform access to all available memory. The memory is organized in eight memory banks, with a total main memory size of 64 Megawords of high-speed static RAM. The system has a Solid State Disk (SSD) with 256 Megawords of memory. The peak performance of the system is 333 Megaflop/s per processor and 2.66 Gigaflop/s for an eight-processor system.

Sandia's nCUBE 2 (manufactured by nCUBE Corporation) has 1024 proprietary nodes which integrate both communications and memory control. Each node operates at 50 MHz and has 4 Megabytes of memory. The nodes are connected via a hypercube communications topology. All coordination among nodes is performed via explicit message passing calls. The bi-directional communication channels have an asymptotic bandwidth of 4.4 Megabytes/s in full duplex mode. Each node runs the Vertex^{TM} operating system, which occupies fewer than 64 kilobytes of memory per node. The proprietary processor is rated at 2.7 Megaflop/s for double-precision (64-bit) floating point operations. Typical performance is 1.5 to 2 Megaflop/s (double precision) per node for both Fortran and C. The 1024-node system has a total memory of approximately four Gigabytes and a theoretical peak speed of 2.8 Gigaflop/s; it achieves 1.5 to 2 Gigaflop/s (double precision) on applications that scale well.

Sandia's nCUBE 2 is shared among multiple users via space sharing, in which each user gets the exclusive use of a subset of the total available nodes called a subcube. A subcube is restricted by the hypercube architecture to consist of a power-of-two number of processors. Single-processor subcubes are allowed. The nCUBE 2 represents relatively mature parallel computing technology.

Sandia's Intel Paragon X/PS L-140 parallel supercomputer (manufactured by Intel Corporation) has 1840 computational nodes, each with an i860 XP RISC processor for computation and an additional i860 XP processor devoted to inter-node communication. 512 of the computational nodes have 32 Megabytes of memory each; the remaining 1328 nodes have 16 Megabytes of memory each, for a total memory of approximately 38 Gigabytes. The processors are connected via a two-dimensional mesh communications topology. All coordination among processors is performed via explicit message passing calls. The bi-directional communication channels have an asymptotic bandwidth of 200 Megabytes/s in full duplex mode. Each node runs its own copy of the operating system. The i860 XP processor is rated at 75 Megaflop/s for double-precision (64-bit) floating point operations. Typical performance is four to eight Megaflop/s (double precision) per node for both Fortran and C. The 1840-node configuration has a total memory of approximately 38 Gigabytes and a peak theoretical speed of 138 Gigaflop/s.

Sandia's Paragon is shared among multiple users via space sharing, in which each user gets the exclusive use of a subset of the total available nodes.

The standard operating system supplied with the Paragon is the Distributed OSF/1 operating system. Release 1.2 of OSF/1 requires approximately 7 Megabytes of memory per node, and supplies far more functionality than we in general require. Sandia's Paragon is run with the Sandia/UNM Operating System (SUNMOS). SUNMOS is an early implementation of the Performance-oriented, User-managed Messaging Architecture, PUMA (Wheat *et al.* 1994). SUNMOS is designed to provide high-performance message-passing and process service on nodes of massively parallel MIMD computers such as the nCUBE 2 and the Paragon while requiring only a small amount of memory (less that 250 Kilobytes per computational node).

PARALLEL CTH

An important class of shock wave physics problems is characterized by large material deformations. These problems involve penetration, perforation, fragmentation, high-explosive initiation and detonation, and hypervelocity impact. These phenomena arise, for example, in armor/antiarmor research and development, the design of impact shielding for spacecraft, the modeling of lithotripsy for the disintegration of kidney stones, and hypervelocity impact problems. The most important of such problems are intrinsically three-dimensional and involve complex interactions of exotic materials, including alloys, ceramics and glasses, geological materials (*e.g.*, rock, sand, or soil), and energetic materials (*e.g.*, chemical high explosives).

Multidimensional computer codes with sophisticated material models are required to realistically model this class of shock wave physics problems. The codes must model the multiphase (solid-liquid-vapor), strength, fracture, and high-explosive detonation properties of materials. Three-dimensional simulations may require millions of computational cells to adequately model the physical phenomena and the interactions of complex systems of components. At Sandia we currently use Eulerian shock physics codes such as Sandia's CTH code (McGlaun and Thompson, 1990; Hertel *et al.*, 1993) to model such problems. CTH is a serial code which runs on Cray vector supercomputers and on workstations. Owing to the expense of high-speed memory, vector supercomputers do not have enough memory to model problems which require more than a few million computational cells. Many problems of interest require tens of millions of cells. Even the inadequately resolved problems often require tens or hundreds of CPU hours to complete. Traditional vector supercomputers are too slow and have too little memory to allow us to study many important weapon safety problems, or to study complex design problems, such as the effects of materials selection and design parameters on the performance of modern armor.

Parallel shock physics codes running on current-generation massively parallel computers are beginning to provide the high resolution and short turnaround time we require for these shock wave physics problems. Three years ago, work at Sandia demonstrated that massively parallel SIMD and MIMD computers running parallel versions of the CTH code were highly competitive with vector supercomputers such as a Cray Y-MP (Robinson *et al.*,. 1991; Gardner and Fang, 1992). Current-generation parallel computers, such as the Paragon X/PS, are demonstrating even better performance, both in terms of problem size and speed.

Sandia scientists have developed a parallel version of the CTH shock physics code. The parallel code, called PCTH, is a multidimensional, multimaterial, finite-difference shock physics code which models large deformation and shocks, and the multiphase behavior, strength, and fracture of materials. In PCTH, the equations governing the conservation of mass, momentum and energy are integrated explicitly in time using a two-step Eulerian scheme. The first step is a Lagrangian step in which the computational cells distort to follow the material motion, using an algorithm which is second-order accurate in space and time. The second step is an advection or remapping step in which the distorted cells are mapped back to the Eulerian mesh using a second-order van Leer advection scheme. The algorithms are implemented using modern object-oriented numerics techniques in the C++ programming language. PCTH is designed to be easily portable to message-passing parallel computers, and currently runs on the nCUBE 2, the Intel iPSC/860, the Paragon, and networks of workstations (Budge *et al.*, 1992; Fang and Robinson, 1993; Wong and Fang, 1993).

A three-dimensional Cartesian computational mesh is implemented in PCTH. Within the simulation domain, each coordinate axis is divided into mesh regions, with a constant or regularly varying interval size within each region. The global mesh is mapped to the nodes in a parallel computer by dividing it into blocks in such a way that each node has nearly the same number of mesh points (the maximum number of cells that any node has is minimized). When a block of the global mesh is mapped to a node, it is surrounded by a layer of ghost cells. These cells are used for enforcing physical boundary conditions when the block boundary corresponds to a physical boundary or for communicating results from logically adjacent nodes when the block boundary falls in the interior of the simulation domain. This is illustrated for a two-dimensional domain in Figure 1.

PERFORMANCE EVALUATIONS FOR MIMD PARALLEL COMPUTERS

In this paper we consider computational rate, memory size, and scalability as metrics of parallel computer performance.

In reporting the performance of a computer, a computational rate is most commonly cited, whether it be theoretical peak computational rate, benchmark results or simulation computational rate for a custom application. Yet computational rate and size are distinct though related aspects of performance. Analysts usually want to perform simulations as quickly as possible—for example,

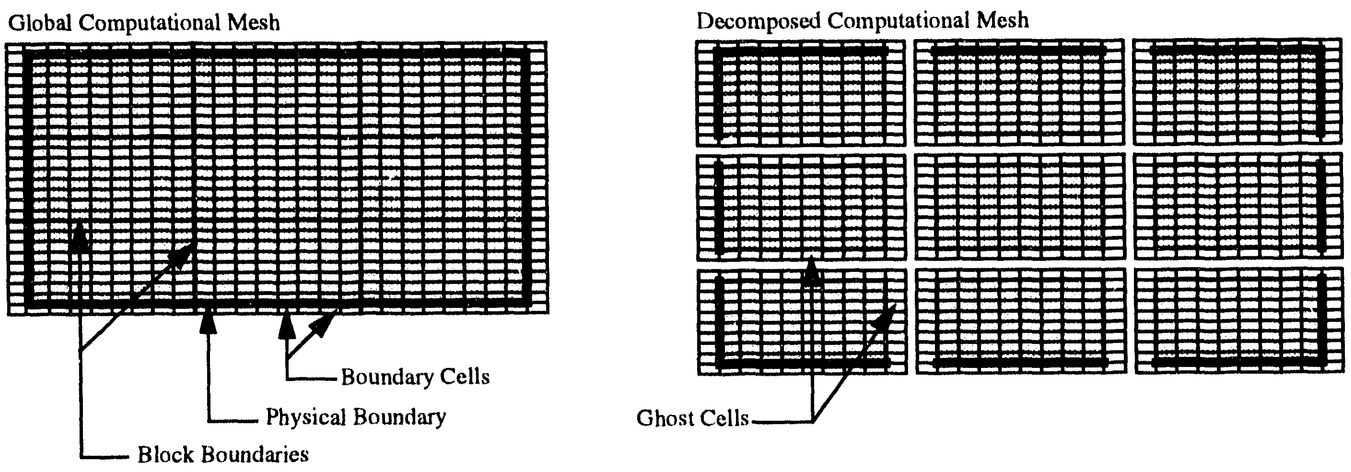


FIGURE 1: ILLUSTRATION OF THE MAPPING OF PCTH MESH BLOCKS TO PARALLEL COMPUTER NODES.

when conducting a parameter study—and distributing a problem on a parallel computer allows the net speed of some massively parallel computers with individually slower processors to match or exceed that of the fastest vector supercomputers on sufficiently large simulations. Vector supercomputers must use expensive high-speed memory to achieve high speed, and the cost of that memory places a practical limit on the memory size of the machine, and hence on the size of the simulations which can be performed with it. Distributed memory parallel computers use slower, less expensive memory, and hence, for the same cost as a vector supercomputer, a parallel computer with much larger total memory can be acquired. Thus it is practical for a distributed memory parallel computer to run much larger simulations than can be performed on existing vector supercomputers. From this point of view, the issue is not so much speed as memory size: if a computer does not have enough memory to perform the simulation, it does not matter how fast it is. Both computational rate and size should be considered when measuring the performance of a computer.

The grind time is a useful measure of the computational rate of a mesh-based, time-marching computer code, such as PCTH. The grind time, t_{grind}, is the execution time for the code calculating a given problem divided by the product of the number of time steps and the number of computational cells:

$$t_{grind} = \frac{T_N(m)}{mn}$$

where $T_N(m)$ is the execution time on N nodes for a problem of m computational cells run for n time steps. The grind time depends on the number of cells and the number of nodes, and also on the specific simulation.

Here we say a code is scalable if the execution speed (or, equivalently, grind time) of the code running a specific problem on a parallel computer increases linearly with the number of nodes when the computational load per node is fixed.

THE TEST PROBLEM AND CONDITIONS

For our test problem we used a projectile impacting a water-filled canister at a velocity of 1.75 km/s along the axis of the canister (Figure 2). This problem is representative of a variety of multimaterial shock wave physics problems. The problem has five materials: 1080 and 4130 stainless steels, 2024 T-4 aluminum, polyproplux (a plastic), and water. Mie-Grüneisen equations of state were used for all the materials. Elastic-plastic constitutive models and fracture models based on tensile pressure with void insertion were used for the aluminum, the polyproplux, and the stainless steels. These models are described in detail in Wong and Fang (1993). All the simulations we used for this work were fully three-dimensional and were conducted in double-precision (64-bit) arithmetic with standard optimized versions of PCTH for each machine (Table 1). In this problem an analyst might be interested in the global evolution of the impact, or in examining details such as the response of the canister wall during the impact.

The problems run on the Cray Y-MP were run with the serial Fortran-77 version of CTH, under UNICOS. The problems run on the nCUBE 2 and the Paragon were run with PCTH under Vertex^{TM} and SUNMOS, respectively; the specific operating system versions are given in Table 1. The problems run were the largest which would fit on each machine or on the number of nodes used for the calculation.

PERFORMANCE RESULTS AND DISCUSSION

Peak processor speed may be an inadequate measure of computer performance. For example, for double-precision (64-bit) arithmetic, the Cray Y-MP processor is rated at 333 Megaflop/s per CPU, the nCUBE 2 processor is rated at 2.5 Megaflop/s per node, and the i860 XP processor is rated at 75 Megaflop/s per node. The peak theoretical speed of a 1024-node nCUBE 2 is thus 2.56 Gigaflop/s and the peak theoretical speed of an 1840-node Paragon is 138 Gigaflop/s. Both the nCUBE 2 and the Paragon are rated at higher theoretical peak speeds and both can attain higher actual computational rates than the Cray Y-MP (Gardner *et al.*, 1992). However, this is only true if the problem is sufficiently

Table 1: Compiler Versions and Optimization Levels for PCTH

Computer	Operating System	Compiler and Version	Compiler Optimization Level
Cray Y-MP	UNICOS 7.0.4	cft 6.0	Fully Optimized
nCUBE 2	Vertex^{TM} f5.41	ncc 3.2	-O
Paragon X/PS	OSF/1 R1.2 with SUNMOS 1.4.7	icc/Paragon Sun4 Version R4.5	(None)

large. Based on the theoretical peak speeds, 256 nodes of the nCUBE 2 and five nodes of the Paragon are equivalent in computational rate to a single-processor Cray Y-MP.

Available processor memory is an important measure of supercomputer performance. For many computers, the memory available for data, M_{data}, can be represented by the equation

$$M_{data} = \sum_{1}^{N_p} (m_{total} - m_{code})$$

where m_{total} is the available memory for code and data per node, m_{code} is the memory required by the operating system and the applications code, and N_p is the number of nodes. Sandia's Cray Y-MP, using its full Solid State Disk, has 200 Megawords or 1.6 Gigabytes of memory, minus the memory required for the operating system and applications code, available for data. Sandia's 1024-node nCUBE 2 has a total memory of 4.096 Megabytes (4 Megabytes per node times 1024 nodes); but the memory available for data is reduced by the memory occupied by the 1024 copies of the operating system and applications code which must also be stored. Similarly, Sandia's 1840-node Paragon has a total memory of 37.6 Gigabytes (32 Megabytes per node for 512 nodes and 16 Megabytes per node for 1328 nodes) minus the memory required by the 1840 copies of the operating system and applications code which must also be stored. Thus for application codes, for example, the nCUBE 2 may have less available memory for running simulations than the Cray Y-MP. In particular, this is true of PCTH, which has an executable size of approximately 1.6 Megabytes on the nCUBE 2.

We noted above that 256 nodes of an nCUBE 2 and five nodes of a Paragon have the same peak theoretical computational rate as a single-processor Cray Y-MP. However, they have less memory than a Cray Y-MP. 512 nCUBE 2 nodes or 100 Paragon nodes are required to match the total memory of the Cray Y-MP. Both these configurations will provide greater computational speed than the Cray Y-MP as well.

We turn now to specific simulations conducted with PCTH. We put the largest computational mesh possible on each computer in order to provide the greatest possible resolution to an analyst, and measured the computational time required to reach a simulation time of 15.0 μ s. The results are summarized in Table 2.

Table 2: Maximum Size and Computational Time for the Projectile-Canister Problem

Computer	Maximum Mesh Size	CPU Time hours	Grind Time μ s/cell/ timestep
Single-CPU Cray Y-MP	$80 \times 80 \times 640$	31.8	58.5
1024-node nCUBE 2	$64 \times 64 \times 512$	11.5	27.6
1840-node Paragon	$150 \times 150 \times 1200$	——*	~4.2

* Owing to machine availability, this calculation was not run to completion.

Simulation on the Cray Y-MP

The largest problem size for the test problem described above which would fit in our SSD is 4,096,000 computational cells (a $80 \times 80 \times 640$ -cell mesh), which uses 181444608 words of memory and ran for 31.8 CPU hours. The grind time was 58.5 μ s/cell/timestep. The resolution afforded by this mesh was sufficient to provide a global picture of the evolution of the impact but not sufficient to allow an analyst to examine the response of the canister wall.

Simulation on the nCUBE 2

The largest problem size for the test problem described above which would fit on Sandia's 1024-node nCUBE 2 is 2,097,152 cells (a $64 \times 64 \times 512$ -cell mesh) and ran for 11.5 CPU hours. The grind time was 27.6 μ s/cell/timestep. The nCUBE 2 demonstrated greater computational speed than the Cray Y-MP but the more limited memory results in poorer resolution. In other words, although the simulation runs faster on nCUBE 2 than Cray Y-MP, the calculated results are less useful to an analyst.

Simulation on the Intel Paragon

The largest problem size for the test problem described above which would fit on Sandia's 1840-node Paragon running the SUNMOS operating system is 27,000,000 cells (a $150 \times 150 \times 1200$ -cell mesh); owing to machine availability, the simulation was not run to completion. The grind time was 4.2 μ s/cell/timestep for the standard production version of PCTH (a more highly optimized version achieved a grind time of 3.3 μ s/cell/timestep). Both higher computational speed and higher resolution were obtained on the Paragon than on the Cray Y-MP or the nCUBE 2.

Scalability

A parallel code will be scalable if the communications overhead is small compared to the total execution time. As shown in the next subsection, a parallel code may be scalable for one problem decomposition, but not for another. The scalability of PCTH on the Paragon is demonstrated by the data in Figure 3, where we

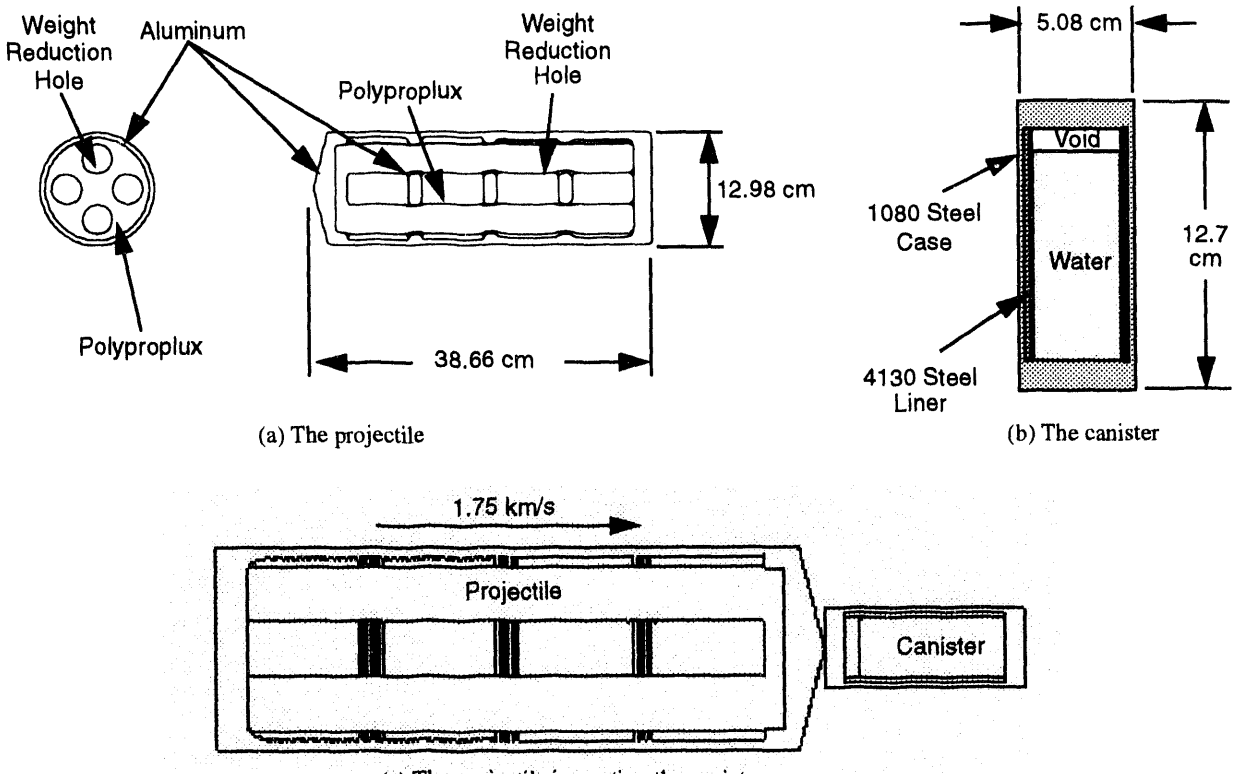


FIGURE 2: THE INITIAL CONFIGURATION FOR THE TEST PROBLEM.

have plotted grind time and the scaled speedup as a function of the number of nodes for a simulation of the explosive welding of a copper tube to a steel plate (Gardner and Fang, 1994). Scaled speedup is the ratio of the time required to solve a problem of size Nm on a single node, $T_1(Nm)$, to the time required to solve the same problem on N nodes with a subdomain of size m on each node, $T_N(Nm)$, when the work per node is fixed (Gustafson *et al.*, 1988). Since $T_1(Nm)$ cannot be measured directly, we estimate it by the $NT_1(m)$, the CPU time that would be required for a single node to solve the problem serially, assuming that no time is required to swap the subdomains in memory and that there is sufficient memory to store all the subdomains. Thus, the scaled speedup, S_N, is given by,

$$S_N = \frac{NT_1(m)}{T_N(Nm)}.$$

For the calculations used to produce Figure 3, the computational load per node remained fixed at the largest cubical subdomain which would fit in node memory. The scalability of PCTH on the Paragon is shown by the linear relationship between the number of nodes and the scaled speedup and grind time in Figure 3. To appreciate the significance of this, consider running a simulation on eight nodes of the Paragon. If we increase the resolution by a factor of two in every coordinate direction and rerun the simulation on 64 nodes, the execution time increases by only a factor of approximately two. Since PCTH is an explicit code and the time step is limited by the computational cell size, increasing the resolution by a factor of two will result in requiring approximately twice as many time steps to reach the same physical time. In contrast, to increase the resolution by the same factor using CTH on a serial computer would require eight times the memory and approximately 16 times the execution time of the original problem because there would be eight times as many computational cells and twice as many time steps. This illustrates that an applications code must be scalable in order to produce more precise results in a more timely manner.

Spatial Decomposition

To run a problem, the analyst may be tempted to use all of the available nodes in a parallel computer, thinking that the more nodes used, the larger the problem which can be solved and the faster it can be solved. However, under some circumstances a larger problem may require fewer processors. To motivate this discussion, consider running a simple two-material problem requiring 240 x 240 x 240 computational cells with PCTH on the Paragon. With the SUNMOS operating system and 1264 processors there are 14 x 10^6 bytes of memory available per computational node for the code and its data. To run this problem on 1264 nodes, the domain decomposition will be 4 x 4 x 79 (4 nodes in the x-coordinate direction, 4 nodes in the y-coordinate direction, and 79 nodes in the z coordinate direction). This implies that there are 60 x 60 x 3computational cells per node (60 cells in the x-coordinate direction, 60 cells in the y-coordinate direction, and 3 cells in the z-coordinate direction) for all but three layers of nodes (which have 60

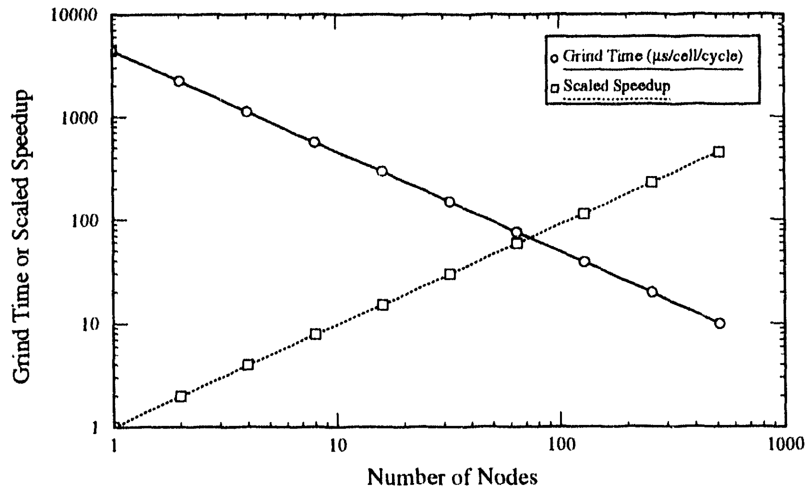


FIGURE 1: PCTH PERFORMANCE ON THE Paragon AS THE COMPUTATIONAL MESH IS REFINED.

x 60 x 4 computational cells), or 62 x 62 x 5 or 62 x 62 x 6 total cells per node (computational cells plus ghost cells), requiring a total of 14.4 x 10^6 bytes per node for some of the nodes. The same problem decomposed for 1024 processors results in a domain decomposition of 8 x 8 x 16, with 30 x 30 x 15 computational cells per processor and 32 x 32 x 17 total cells per node, requiring a total of 10.8 x 10^6 bytes per processor. The net result is that the 1264-processor decomposition will not run, while the 1024-processor decomposition will.

The difficulty lies in the aspect ratio of the decomposition. If the prime factorization of the number of processors includes a large prime number (such as 79 in the example above), then one dimension of the problem domain will be divided by that large prime in allocating subdomains to processors. While this may be appropriate for problem domains which are long and narrow (i.e., have a large aspect ratio), more commonly problems have aspect ratios of order one, i.e., they resemble cubes. For the more cubical domains, one domain dimension will be divided by the large prime creating subdomains which have a large aspect ratio. This subdivision causes two problems, both of which degrade the performance of the code. One problem is that the number of ghost cells increases dramatically. A second problem is that the communication overhead also increases.

To understand the problem of increased numbers of ghost cells, recall that each subdomain is surrounded by a layer of ghost cells, which are used in storing and communicating intermediate results (Figure 1). These cells require memory, and consequently we want to use as few ghost cells as possible. For the problem described above decomposed on 1264 processors, most nodes have10,800 computational cells and 8420 ghost cells, a ratio of 0.80 ghost cells per computational cell. For the same problem decomposed on 1024 processors, there are 13,500 computational cells and 3908 ghost cells per node, a ratio of 0.29 ghost cells per computational cell. Thus, in this example, in decomposing the problem onto a larger number of nodes, we are incurring a greatly increased memory overhead due to the ghost cells resulting from the aspect ratio.

In addition, since the primary purpose of the ghost cells is to store communicated results from other processors, an increased number of ghost cells also indicates an increase in the communications overhead. For a fixed problem, the number of communication calls per node will be fixed, but the communication traffic will depend on the decomposition: the more ghost cells, the greater the communication traffic and hence the greater the communication overhead.

Thus, in some circumstances, using a larger number of nodes will not guarantee being able to solve a larger problem and it may also increase the run time owing to the increased communication overhead. To avoid this problem on a machine like the Paragon, the analyst should avoid using a number of nodes whose prime factorization includes large primes.

SUMMARY AND CONCLUSIONS

In this paper we have discussed the performance of multiple-instruction, multiple-data (MIMD) distributed memory parallel computers from the point of view of an engineer or scientist seeking to model complex physical systems using a standard version of an analysis code. To provide illustrative data for our discussion, we modeled the hypervelocity impact of a projectile with a water-filled canister using the CTH shock wave shock wave physics code running on the Cray Y-MP, and the parallel version of CTH, PCTH, running on a 1024-node nCUBE 2 and an 1840-node Intel Paragon. As the basis for our comparison, we modeled the projectile-canister problem using the highest resolution uniform grid possible on each computer, as limited by memory. Using these simulations, we demonstrated that while a parallel computer may provide greater computational speed than a vector supercomputer, it must also have sufficient memory capacity to provide equal or greater resolution. In particular, for our simulations, the 1840-node Paragon provided both higher resolution and greater computational speed than the 1024-node nCUBE 2 or the Cray Y-MP for problems which essentially filled the memory of each computer. In contrast, owing to memory constraints, the nCUBE 2 provided greater computational speed but lower resolution than the Cray Y-

MP. Thus both computational speed and the memory available for applications code data must be considered in evaluating parallel computer performance. On a distributed memory parallel computer the memory available for applications data is significantly reduced because the operating system and the applications code must be stored on each node.

We also demonstrated the scalability of the PCTH code on the Paragon as the simulation size is increased with a fixed computational load per node. An applications code must be scalable in order to increase the resolution without unreasonably increasing the execution time as more nodes are used and so provide an analyst with more precise and, presumably, more useful information in a more timely manner. Scalability depends on the domain decomposition used, as well as on the efficiency with which inter-processor communications are effected within the code.

Finally we demonstrated that naively using more nodes on a parallel computer may not always result in solving a problem faster or with greater resolution. For applications codes which use ghost cells for communicating results among nodes, the aspect ratio of the individual subdomains is also an important consideration. In some cases, increasing the number of subdomains also increases the number of ghost cells and decreases the memory available for real computational cells. Increasing the number of ghost cells also corresponds to increasing the communications overhead. The number of ghost cells may be increased unnecessarily either through placing fewer real computational cells on more processors or through subdividing the computational domain in such a way as to produce subdomains with large aspect ratios. The latter situation can arise when the number of nodes used contains a large prime factor. In general, the analyst should use numbers of nodes without large prime factors to avoid incurring unnecessarily high memory and communication overhead costs.

ACKNOWLEDGMENTS

PCTH has been developed by a team of Sandia scientists, including A. C. Robinson, J. S. Peery, C. J. Pavlakos, M. K. Wong, A. L. Ames, D. A. Verner, and the authors.

The work described in this paper was supported under the Joint DoD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Office of the Secretary of Defense.

REFERENCES

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., Venkatakrishnan, V., and Weeratunga, S. K., 1991, "The NAS Parallel Benchmarks," *International. Journal of Supercomputer Applications*, Vol. 5, pp. 63-73.

Budge, K. G.; J. S. Peery, and A. C. Robinson, 1992, "High-Performance Scientific Computing Using C++," *Proceedings, USENIX C++ Technical Conference*, USENIX Association, Berkeley, CA, pp. 131-150.

Dongarra, J. J., Bunch, J., Moler, C., Stewart, G. W., *LINPACK User's Guide*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1979.

Dongarra, J. J., 1994, "Performance of Various Computers Using Standard Linear Equations Software," Report CS-89-85 (March 23, 1994), Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301.

Fang, H. E. and Robinson, A. C., 1993, "3-D Massively Parallel Impact Simulations Using PCTH," *Proceedings, 1993 Summer Computer Simulation Conference*, J. Shoen, ed., The Society for Computer Simulation, San Diego, CA, pp. 385-390.

Gardner, D. R., Cline, D. D., Vaughan, C. T., Krall, R., and Lewitt, M., 1992, "The Performance of PAGOSA on Several SIMD and MIMD Parallel Computers," Technical Report SAND92-1452, Sandia National Laboratories, Albuquerque, New Mexico.

Gardner, D. R. and Fang, H. E., 1992, "Three-Dimensional Shock Wave Simulations on Massively Parallel Supercomputers," *Proceedings, 1992 Summer Computer Simulation Conference*, P. Luker, ed., The Society for Computer Simulation, San Diego, CA, pp. 537-541.

Gardner, D. R. and Fang, H.E., 1994, "Three-dimensional Shock Wave Physics Simulations With PCTH on the Paragon Parallel Computer," *Proceedings, 1994 Simulation Multiconference, High Performance Computing Symposium 1994—Grand Challenges in Computer Simulation*, A. M. Tentner, ed., The Society for Computer Simulation, San Diego, CA, pp. 132-137.

Gustafson, J. L., Montry, G. R., and Benner, R. E., 1988, "Development of Parallel Methods for a 1024-Processor Hypercube," *SIAM Journal on Scientific and Statistical Computing*, Vol. 9, pp. 609-638.

Hertel, E. S., Jr.; Bell, R. L., Elrick, M., G., Farnsworth, A. V., Kerley, G. I., McGlaun, J. M., Petney, S. V., Silling, S. A., Taylor, P. A., and Yarrington, L, 1993, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," *Proceedings, 19th International Symposium on Shock Waves*, Université de Provence, Provence, France, Vol 1., p. 274

McGlaun, J. M. and Thompson, S. L., 1990, "CTH: A Three-Dimensional Shock Wave Physics Code," *International Journal of Impact Engineering*, Vol. 10, pp. 351-360.

Robinson, A. C., Vaughan, C. T., Fang, H. E., Diegert, C. F., Cho, K.-S., 1991, "Hydrocode Development on the nCUBE and the Connection Machine Hypercubes," *Shock Compression of Condensed Matter—1991, Proceedings of the American Physical Society Topical Conference*, S. C. Schmidt et al., ed., Elsevier Science Publishers B.V., Amsterdam, The Netherlands, pp. 289-292.

Wheat, S. R., Maccabe, A. B., Riesen, R., van Dresser, D. W., and Stallcup, T. M., 1994, "PUMA: An Operating System for Massively Parallel Systems," *Proceedings, The 27th Annual Hawaii International Conference on System Sciences*, H. El-Rewini and B. D. Shriver, ed., IEEE Computer Society Press, Los Alamitos, CA, Vol. 2, pp. 56-65.

Wong, M. K, and Fang, H.E., 1993, "MatResLib: A Reusable, Object-Oriented Material Response Library," *Proceedings, First Annual Object-Oriented Numerics Conference (OON-SKI'93)*, T. Keffer, ed., Rogue Wave Software, Inc., Corvalis, OR, pp. 364-372.