Domain Decomposition Strategies for Solving the Maxwell Equations on Distributed Parallel Architectures

Douglas C. Blake[†] and Thomas A. Buter[§]
Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433

Abstract

Domain decomposition strategies for solving hyperbolic systems of partial differential equations on distributed-memory parallel computing platforms are investigated. The logically-rectangular computational domain is divided either one, two, or three dimensionally into a series of computational blocks, and each block is assigned to a single processor. Theoretical predictions using standard parallel performance models indicate that higher-dimensional decompositions provide superior parallel program performance in terms of scalability. The theory is tested using a finite-volume time-domain (FVTD) Maxwell equations solver to compute the electromagnetic fields inside a rectangular waveguide using various grid sizes and processor numbers on three different parallel architectures–the Intel Paragon, the IBM SP2, and the Cray T3D. The specific performance of the FVTD algorithm on the three machines is investigated, the relation between processor connection topology and message passing performance of a typical grid-based hyperbolic equation solver are identified, and the results are used to augment the classical parallel performance model. Although clear performance trends emerge in terms of the dimensionality of the decomposition, results indicate that higher-dimensional decompositions do not always provide superior parallel performance.

1 Introduction

Numerical simulations of electromagnetic or fluid flow phenomena are most often constrained in their complexity by available computational processing capability. Although complex three-dimensional calculations were once the exclusive realm of the vector supercomputer, through recent dramatic advances in computer hardware technology–including the development of powerful Reduced Instruction Set Computing (RISC) microprocessors, high-density Dynamic Random Accesses Memory (DRAM), and very-high-speed switching networks–designers have been able to construct powerful machines comprised of hundreds to thousands of processors which are often capable of performance exceeding that of traditional vector supercomputers. Unfortunately, efficient utilization of these parallel machines requires much more effort on the part of the algorithm designer since compilers are not yet able to fully extract the parallelism inherent in a computer code and properly map that parallelism to a distributed-memory environment.

One method of potentially achieving a high degree of concurrency in typical grid-based scientific computations such as those found in time-domain computational electromagnetics (CEM) or computational fluid dynamics (CFD) algorithms is to divide the computational domain into a series of subdomains or blocks and then assign the blocks in some manner to the available processors. This approach has been applied with an emphasis toward engineering practicality by several researchers in both the CFD and CEM communities[1-6]. Furthermore, the issue has been examined from a parallel efficiency standpoint by Wong, et al.[7]

Since the practice of domain decomposition has become increasingly prevalent, it is important to investigate methods for partitioning and assigning the computational domain to the available processors. Because the problem of determining an optimal mapping is known to be in problem space NP Complete[8], no such attempt is made here. Instead three domain decomposition approaches–one, two, and three dimensional (as depicted in Figure 1)–are examined in terms of theoretical and measured parallel scalability. Furthermore, since modern parallel architectures differ widely in their processor capabilities, interprocessor connection topologies and communication bandwidths, it is reasonable to expect that a given domain mapping cannot yield identical parallel scalability across all platforms. For this reason, the domain decomposition analysis is performed on three separate parallel platforms which are discussed in the following section.

2 Target Architectures

The machines chosen for use in this study–the Intel Paragon, the IBM SP2 and the Cray T3D–arguably represent the most widely available currently produced distributed-memory computing platforms. Each of these machines is similar in that each utilizes RISC processors as its central processing units. However, the interprocessor connection topology of the machines differs substantially. The Paragon and T3D both employ very-high-performance static interconnection networks configured as a two-dimensional mesh and a three-dimensional torus, respectively. The SP2, on the other hand, utilizes an omega network of substantially lower bandwidth. In addition to the differences in connection topology, the T3D provides several programming options not available on the other machines due to its memory structure. Although the SP2 and Paragon are true distributed-memory platforms,

[†]PhD Candidate, Member ACES

[§]Assistant Professor of Aeronautical Engineering

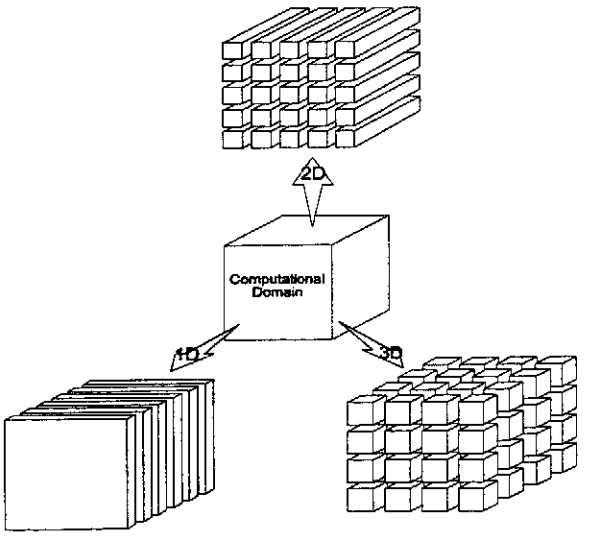

Figure 1: Domain Decomposition Techniques

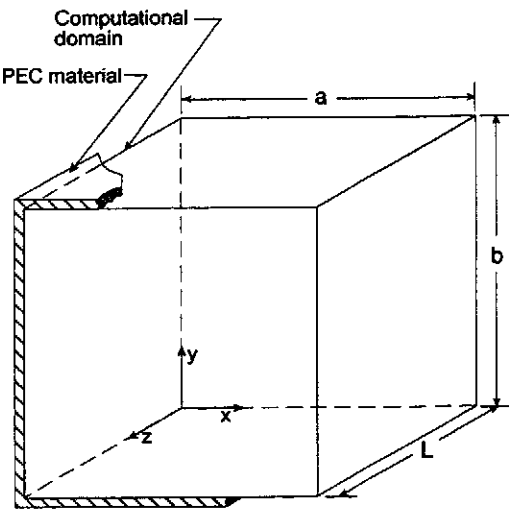

Figure 2: Rectangular Waveguide Geometry

the T3D is classified as a distributed-shared-memory machine since, although memory is physically distributed, a single global address space is available. This allows any processor to directly address memory contained on another processor. It is also possible to explicitly deliver data between processors through the use of a message-passing library such as that encapsulated in Parallel Virtual Machine (PVM)[9]. In this mode, the programming environment of the T3D is similar to that of the Paragon and SP2.

Although several message-passing libraries are available for each machine, the libraries used in this study were chosen based on vendor support, portability, and performance. At the inception of this study, PVM was actively supported by Cray Research for the T3D. Similarly, support and documentation for the Intel message passing library, NX[10], was readily available. Finally, Message Passing Interface[11] (MPI) was selected for the SP2 due to its portability and performance. By abstracting the library-specific programming calls through the use of the macro feature in the C programming language, a single computer code was used to collect performance data on all three machines. A summary of the salient features of each of the machines appears in Table 1.

3 Problem Description

3.1 Model Problem

The problem selected for examining the various domain decomposition approaches was the computation of the electromagnetic fields inside a rectangular waveguide as depicted in Figure 2. The computational domain for the problem was uniform and Cartesian, thereby facilitating a relatively straightforward partitioning of the domain. The physical dimensions a, b, and L of the waveguide were scaled to π, π, and 1 respectively, and the waveguide was excited in a TMZ11 mode. This problem has been extensively studied by Shang[4,12], and the existence of an analytical solution[13] allows for a ready verification of parallel program correctness.

3.2 Numerical Procedure

In order to compute the electromagnetic fields inside the rectangular waveguide, the differential vector forms of the two Maxwell curl equations are solved using a collocated, cell-centered, explicit FVTD scheme. For a general curvilinear (ξ, η, ζ) coordinate system, the equations can be written as

$$\frac{\partial \tilde{Q}}{\partial t} + \frac{\partial \tilde{E}}{\partial \xi} + \frac{\partial \tilde{F}}{\partial \eta} + \frac{\partial \tilde{G}}{\partial \zeta} = \tilde{J} \tag{1}$$

where $\tilde{Q}$ contains the six unknown scalar electromagnetic field components, $\tilde{J}$ contains the scalar components of the surface current, and $\tilde{E}$, $\tilde{F}$, and $\tilde{G}$ are known as the *flux vectors*. Application of equation (1) to a general hexahedral finite volume cell yields

$$\frac{\partial (\tilde{Q}V)}{\partial t} + \sum_{k=1}^{6} \tilde{R}_k \cdot \hat{n}_k A_k - \tilde{J}V = 0 \tag{2}$$

where $\tilde{R} = \tilde{E}\hat{\xi} + \tilde{F}\hat{\eta} + \tilde{G}\hat{\zeta}$, $\hat{n}_k$ and A_k are the unit surface normal and surface area of cell face k, respectively, and V is the cell volume.

In order to advance the solution in time, the FVTD scheme requires the evaluation of the flux vectors at the cell faces. To do so, the present study uses a flux-vector-splitting approach after Steger and Warming[16] which splits the flux at a cell face into positive and negative components according to the signs of the eigenvalues of the flux Jacobian matrix A,

	Cray T3D (Eglin Air Force Base, FL)	IBM SP2 (Maui High Performance Computing Center)	Intel Paragon XP/S (Wright Patterson Air Force Base, OH)
central processing unit (CPU)	DEC Alpha 21064 @ 150 megahertz	IBM RS/6000 @ 66.7 megahertz	Intel i/860XP @ 50 megahertz
single processor megaflop rating (double precision)	150	266	75
number of compute processors	128	400	368
memory per CPU (megabytes)	64	64-1024	32-64
interprocessor connection topology	Three-dimensional torus (nodes configured 8 x 4 x 4)	Omega network	Two-dimensional mesh
communication bandwidth (megabytes/sec)	300	40	200
communication library used	Parallel Virtual Machine (PVM)	Message Passing Interface (MPI)	NX
maximum processors available to a single job (without special request)	128	128	128

Table 1: Target Machine Characteristics

where for the η direction

$$A = \frac{\partial \tilde{F}}{\partial \tilde{Q}} \tag{3}$$

and for the $i+1/2$ face of cell i (see Figure 3),

$$\tilde{F}_{i+1/2} = \tilde{F}^{+}(\tilde{Q}^{L}_{i+1/2}) + \tilde{F}^{-}(\tilde{Q}^{R}_{i+1/2}) \tag{4}$$

Similar expressions hold for the fluxes at the five remaining cell faces. Note that the positive and negative fluxes in equation (4) are functions of the dependent variables at the left and right states (denoted by the superscripts L and R in the equation) of cell face $i+1/2$; however, the dependent variables are not defined at the cell faces but rather at the cell centers. It thus becomes necessary to transfer information from cell centers to cell faces. This can be accomplished in one of two ways: the fluxes can be evaluated at cell centers and then extrapolated to cell faces, or the dependent variables can be extrapolated to the cell faces and the fluxes subsequently evaluated. The latter method is known as *Monotone Upstream-centered Schemes for Conservation Laws* (MUSCL) and is the methodology used in this work. Using the MUSCL approach after van Leer[17], the extrapolation yields a scheme that is spatially third-order accurate.

With the flux evaluation complete, equation (2) is solved over each cell by applying a two-stage, second-order-accurate Runge Kutta procedure to the temporal derivative term. This yields a fully explicit numerical procedure that is ideally suited to a parallel implementation.

3.3 Parallel Implementation

The MUSCL scheme described in the previous section forms the crux of the data dependencies of the numerical scheme. Referring to Figure 3, in order to compute the total flux at cell face $i+1/2$, $\tilde{F}_{i+1/2}$, the positive flux component requires dependent variable information from cells $i-1$, i, and $i+1$. Similarly, the negative flux component requires information from cells i, $i+1$, and $i+2$. These data dependencies are known as the *computational stencil* of the scheme and are depicted in the top portion of the figure. In a parallel environment, one or more of these cells may reside on different processors, and thus a means must exist to transfer necessary information between processors. The transfer of information is facilitated through the use of buffer storage locations. These locations serve to hold dependent-variable or other information that is computed on one processor but necessary for other computations on another. A sample of the buffer locations is shown as the shaded cells in the lower portion of Figure 3. Using this approach for the simple two-processor, one-dimensional example shown in the figure, the flux calculation for a cell which falls on a boundary created by the domain decomposition begins as processors 1 and 2 independently compute the positive and negative flux component, respectively, for cell face $i+1/2$. This calculation requires data stored in the buffer locations on each processor. Processor 1 passes the positive flux component to processor 2 which then computes the total

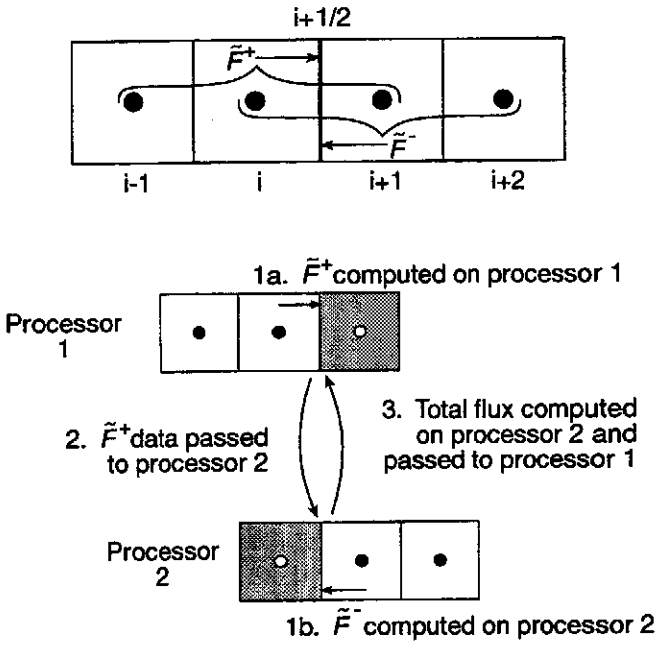

Figure 3: Flux Message Detail

flux for the cell face and returns this information to processor 1. Assuming the fluxes at the remaining cell faces have been calculated, the processors subsequently update cells i and $i+1$, respectively, to the new time level. For a higher-dimensional problem, the message-passing scenario is repeated for each of the decomposition directions. No messages are required along a coordinate direction in which the computational domain is not decomposed. Once all cells have been updated to the new time level, dependent-variable information is exchanged so the buffer storage locations contain new time level data. Since the waveguide was considered filled with a homogeneous material (free space), the wave speed was uniform throughout the computational domain. This allows a single global time step determination at the beginning of program execution, and thus no global communications are required by the algorithm after the initial time step calculation.

It is possible to double the size of the buffers and thereby preclude the requirement for either processor to send any flux data to the other; however, this approach was examined and discarded since the storage penalty exacted was not offset by any significant performance gain.

4 Parallel Analysis

4.1 Theoretical Parallel Performance

Given the algorithm data dependencies described in the previous section, the theoretical parallel performance can be determined. Parallel run time, T_{par}, can be assumed to consist of contributions from calculations, T_{calc}, and from parallel overhead, $T_{overhead}$, i.e.

$$T_{par} = T_{calc} + T_{overhead} \tag{5}$$

The overhead consists of several factors including communication and any extra calculations necessary to implement the code on a parallel machine. For this analysis, the parallel overhead is assumed to be dominated by the communication time, T_{comm}. If the classical cut-through-routing communication cost model of Kumar, et al.[19] is used, then the communication time required for a single message to travel between processors a and b, t_{comm}, is given by

$$t_{comm} = t_s + mt_w + lt_h \tag{6}$$

where t_s is the message start-up time, t_w is the per-word transfer time, t_h is the latency associated with a hop between two processors, m represents the number of words transferred, and l represents the number of hops the message must make in order to travel from processor a to processor b. In most modern parallel architectures, the per-hop time is extremely small and all processors can be considered computationally close. This allows the communication cost model to be simplified[8], viz.

$$t_{comm} = t_s + mt_w \tag{7}$$

For this analysis, the computational domain is assumed to be three dimensional and to consist of n cells with $n^{1/3} \equiv R$ cells distributed along each of the three coordinate directions. Furthermore, the domain is assumed to be evenly divided among the number of available processors, p. Thus, the number of grid cells residing on each processor, n_p, is given by

$$n_p = R^3/p \tag{8}$$

With the exception of processors that contain the grid cells on the boundaries of the computational domain, each processor must send one message to each neighboring processor along each of the decomposition directions during the numerical flux computations. Furthermore, once the flux computations are complete, the updated dependent-variable information stored in the buffer arrays must be exchanged between neighboring processors. Since the time integration portion of the algorithm is a two-step procedure, this scenario is repeated twice in order to advance the solution in time. Consequently, the number of messages sent and received per time step is $16i$, and the length of each message, m, is

$$m = DR^2 p^{(1-i)/i} \tag{9}$$

where D represents the number of storage bytes required per grid cell for the dependent-variable data and i represents the dimensionality of the decomposition, $0 \le i \le 3$. Note that for $i = 0$, the computational domain is not decomposed, and thus $i = 0 \Leftrightarrow p = 1$. In all cases, the term $p^{(1-i)/i}$ is restricted to integral values. Since the total communication

time is the product of the single message communication time and the total number of messages, then

$$T_{comm} = 16i(t_s + DR^2 p^{(1-i)/i} t_w) \quad (10)$$

which leads to the theoretical parallel run times for the various decompositions, namely

$$T_{par} = t_c(R^3/p) + 16i(t_s + DR^2 p^{(1-i)/i} t_w) \quad (11)$$

where t_c is the single-processor computation time per grid cell per time step.

With parallel run time determined, theoretical absolute speedup, S_a, and absolute efficiency, E_a–two of the most common metrics for parallel performance–can be determined using the standard definitions [19]

$$S_a = T_1/T_{par} \quad (12)$$

$$E_a = S_a/p \quad (13)$$

where T_1 is the run time for the best known serial algorithm to solve the problem in question. It is often not practical to compare the chosen parallel algorithm against the best known serial algorithm, and consequently, relative speedup, S, and efficiency, E, are often used whereby the parallel algorithm run time is compared against the run time of the algorithm on a single processor. Substituting $i = 0, p = 1$ into equation (11) to determine T_1, the theoretical relative performance is thus given by

$$S = \frac{1}{\frac{1}{p} + \frac{16i}{R}[R^{-2}(t_s/t_c) + Dp^{(1-i)/i}(t_w/t_c)]} \quad (14)$$

$$E = \frac{1}{1 + \frac{16i}{R}[pR^{-2}(t_s/t_c) + Dp^{1/i}(t_w/t_c)]} \quad (15)$$

At this point, the applicability of the analysis to other implementations can be extended by treating the terms t_c, t_s, t_w, and $16i$ (the number of message passes) as constants. An asymptotic analysis is then performed in which p and R are assumed large to yield[19]

$$E = \frac{1}{1 + \Theta(p^{1/i}/R)} \quad (16)$$

where Θ indicates a tight upper bound.

It is important to note that the asymptotic analysis encapsulated in (16) is taken for p and R appropriately large when in fact, for a practical implementation, it may not be possible to let either variable grow suitably large due to memory or machine architecture restrictions. Consequently, a consideration of the constants appearing in (14) and (15) may be necessary when assessing true performance of the algorithm on the machine of implementation.

Since parallel machines provide a means of solving relatively large problems over a fairly large number of processors, it is desirable to predict how an algorithm will perform as both the problem size and the number of processors is increased. *Isoefficiency* measures the scalability of an algorithm in such an instance and is determined by setting the computational work equal to the parallel overhead function[19]. In the present analysis, the isoefficiency is derived directly from equation (16) to yield

$$R = n^{1/3} = \Theta(p^{1/i}) \Rightarrow n = \Theta(p^{3/i}) \quad (17)$$

Thus, in order to maintain a constant parallel efficiency, a computational domain that is decomposed three dimensionally need only be increased in size by an amount linearly proportional to increasing processor number. One- and two-dimensional decompositions require larger increases in the size of the computational domain as increasing numbers of processors are applied to the problem if the efficiency is to be maintained at a constant value.

4.2 Machine Performance Characterization

Determination of theoretical performance as embodied in equations (14) and (15) requires that the values of t_c, t_s, and t_w be ascertained. In general, t_s and t_w are machine specific, while t_c depends both on the target architecture as well as the algorithm. In order to determine t_c, the FVTD algorithm was run for 100 time steps on a single processor on each of the three machines using a variety of grid sizes up to the maximum size containable in the machine's core memory. The times for these execution runs are contained in Figure 4. Although not presented here, a formal analysis of the FVTD algorithm run-time complexity reveals that the method is in time space $\Theta(n)$ where n is the number of cells in the computational domain. This analysis is corroborated by the data exhibited in the figure which show the run times to be a linearly increasing function of n. Any slight deviations from linearity can be explained by noting that boundary condition cells require less computational work, and as the grid size decreases, the boundary condition points have an increased affect on the algorithm run time. Using a simple least-squares fit of the data, t_c was determined from the slope of each plot.

The message-passing performance of each of the three machines was determined by configuring eight processors as a logical ring and circulating messages of varying size 1000 times around the ring. Run time data for this experiment appears in Figure 5. By performing a linear curve fit of the data, the message start-up time and per-word transfer time can be computed directly from the y-intercept and line slope. Although the linear curve-fitting process generated excellent results (goodness of fit > 0.99 in all cases), the SP2 is characterized by marked and somewhat random variations in execution time. This variation is most likely due to message contention over the omega switching network. Table 2 con-

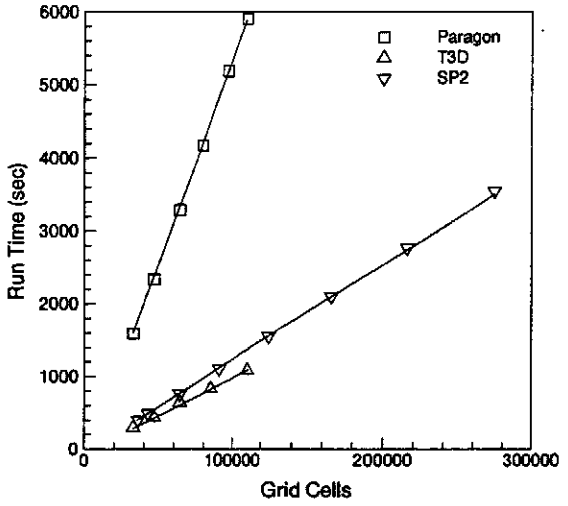

Figure 4: Single Node Algorithm Run Times

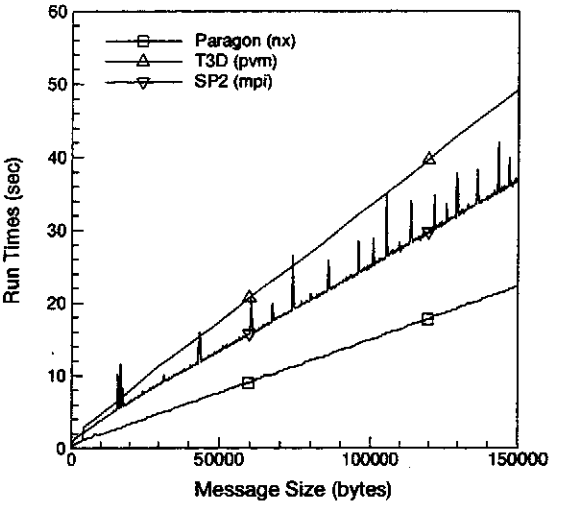

Figure 5: Ring Message Passing Times
(1000 rings on 8 processors)

tains a summary of the findings of this portion of the study. The values agree in general with those of Foster[8]. Any discrepancies can be explained by differences in operating systems and message-passing libraries used.

5 Domain Decomposition Study

5.1 Test Procedure

As is evident in the theoretical derivations, parallel speedup and efficiency are dependent on a variety of parameters including the problem size and the number of processors on which the algorithm is executed. In order to test these parameters, the FVTD algorithm was run for 100 iterations on grid sizes ranging from $R = 32$ to $R = 128$ using up to 128 processors. One-hundred iterations was deemed an acceptable number so that run times would not be excessive yet any transient parallel-environment effects such as interprocess-message contentions would be suitably minimized. In every decomposition, the number of finite-volume cells residing on each processor was identical. This reduced, but did not completely eliminate, load imbalance since boundary condition cells require less computational effort than interior cells. The choice of domain decompositions and grid sizes was constrained primarily by the amount of memory available on each processor. This was especially true in the case of the Paragon which, with the exception of 16 "MP" nodes, possesses only 32 megabytes per processor. In addition to the memory constraint, additional restrictions were imposed by the processor-allocation scheme of the T3D. The current version of the operating system on this machine allows processors to be allocated only in powers of two. This required that the computational domain be partitioned in powers of two along each decomposition direction.

In addition to the dimensionality of the decomposition, the directional dependence of the decompositions was assessed by permuting the decompositions for each coordinate direction. The permutations correspond to a re-orientation of the planes or lines of grid points as depicted in Figure 1. Examination of the decomposition along each direction allows for an assessment of machine memory-accessing performance and uncovers potential bus contentions which are the inevitable result of mapping a higher-dimensional physical problem onto a lower-dimensional interconnection network. Table 3 contains a summary of the decompositions examined. Rather than list each grid size, processor count, and decomposition separately, they are combined into a single listing whenever possible with the understanding that all possible combinations on a given table row were examined.

The FVTD code used to conduct the study was written in C to take advantage of that language's dynamic memory allocation routines. Memory for each grid decomposition was allocated at run time. This facilitated the examination of

	t_c (µsec)	t_s (µsec)	t_w (µsec)
T3D	102	25	.030
Paragon	546	36	.018
SP2	132	76	.040

Table 2: Machine and FVTD Algorithm Performance Parameters

Dimension of Partition	Grid Size (R)	Processors (P)	Decomposition*
1D	32, 64, 128	4, 8, 16, 32, 64, 128	$P \times 1 \times 1$ such that $P \le R$
2D	32, 64, 96	4, 16, 64	$\sqrt{P} \times \sqrt{P} \times 1$
2D	32, 64, 96	8, 32, 128	$\sqrt{\frac{P}{2}} \times 2 \sqrt{\frac{P}{2}} \times 1$
3D	32, 64, 96	8, 64	$\sqrt[3]{P} \times \sqrt[3]{P} \times \sqrt[3]{P}$
3D	32, 64, 96	16, 128	$\sqrt[3]{\frac{P}{2}} \times \sqrt[3]{\frac{P}{2}} \times 2\left(\sqrt[3]{\frac{P}{2}}\right)$
3D	32, 64, 96	32	$2 \times 4 \times 4$

*All permutations of the tabulated decompositions were performed. For example, in addition to the stated $P \times 1 \times 1$ one-dimensional partitioning, $1 \times P \times 1$ and $1 \times 1 \times P$ partitions were also examined. The decomposition nomenclature refers to the number of blocks by which the computational domain was partitioned in each coordinate direction.

Table 3: Summary of Examined Decompositions

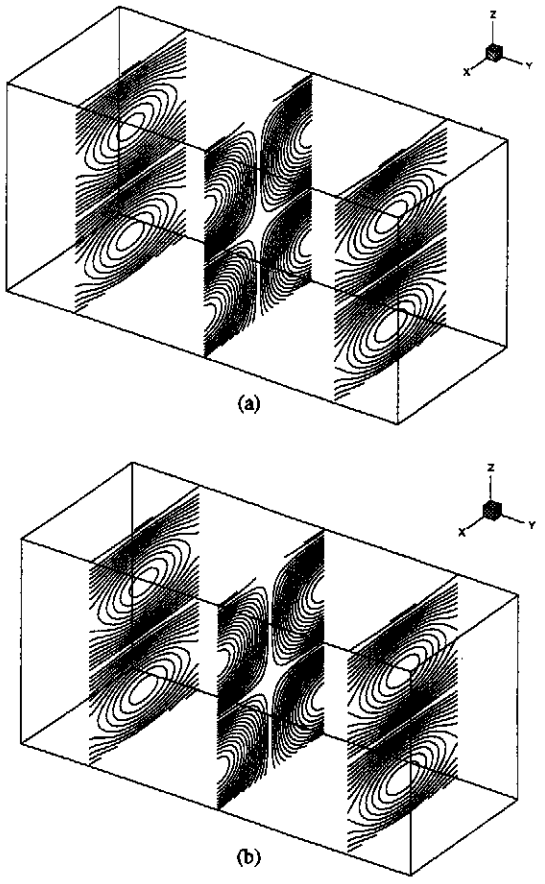

Figure 6: Total Magnetic Field Contours
(a) exact, (b) computed

a large number of decompositions during a single program run with no memory wasted due to static array dimensioning. All message passing was performed using non-blocking communication primitives with special care taken to interleave communication and computation wherever possible.

As timing data was collected, several runs were reaccomplished to assess the repeatability of the timing data. In the case of the T3D and the Paragon, results were found to be repeatable to well within one percent. However, such was not the case for the SP2 where timing data varied much more dramatically (most likely due to the effects observed in Figure 5). In order to ensure accurate results, all runs on the SP2 were accomplished at least five times and the minimum time observed was used as the execution time. This procedure is similar to that recommended by Thinking Machines in the timing studies conducted by Blosch[1].

5.2 Results

5.2.1 Electromagnetic Field Computations

Figure 6 contains a comparison of the magnitude of the computed and exact magnetic fields inside the waveguide. The computed solution was obtained from a 32-node Paragon run using approximately 110,000 grid points with the computational domain partitioned three-dimensionally in a $4 \times 2 \times 4$ configuration. In the figure, the y and z axes have been scaled so that the cutting planes located at $y = 0.6$, 1.6, and 2.5 are unobstructed. The computed and exact solutions are in excellent agreement; in fact, the plots are indistinguishable. While not a formal proof of correctness, it does indicate that the parallel algorithm functioned as intended.

5.2.2 Parallel Scalability

The results of the domain-decomposition studies appear in Figures 7-12. Figures 7-9 contain parallel speedup results for the one-, two-, and three-dimensional decompositions, respectively, while Figures 10-12 contain efficiency results. A comparison of the relative performance of the FVTD algorithm for a given dimensionality of decomposition on each of the three machines can be conducted by examining sub-figures *a*, *b*, and *c* of a single figure. On the other hand, a comparison of the sub-figures in a given column facilitates an examination of the effects of the dimensionality of the decomposition for a given platform. It is apparent from the figures that in nearly every case, the two- and three-dimensional decompositions produced performance superior to that of the one-dimensional decompositions. Furthermore,

three-dimensional decompositions exhibited slightly better performance in several instances on the T3D while two-dimensional decompositions were slightly superior in general on the Paragon and noticeably superior on the SP2. This is especially evident in an examination of the efficiency curves of Figures 10-12. These trends in parallel performance correspond quite closely to the topologies of the three machines. The three-dimensional torus structure of the T3D allows each processor six neighboring processors while the two-dimensional mesh structure of the Paragon translates to at most four neighbors for a given computational node. In contrast, the omega network of the SP2 requires that any communication between two processors traverse at least one switch hop and two communication lines. It appears that superior parallel performance is achievable when there is a close agreement between the physical dimensionality of the domain decomposition and the physical topology of the architecture onto which it is mapped. It should be remembered that the speedup and efficiency curves of Figures 7-12 provide one measure of the scalability of the algorithm and do not reflect actual program execution times. For example, although the curves show the scalability of the algorithm on the SP2 to be decidedly less than that on the other two platforms, the superior performance of the RS/6000 when compared to the i/860XP yielded substantially faster run times. In other cross-platform comparisons, the T3D was able to significantly outperform the other two machines both in terms of parallel scalability and in terms of absolute execution speed. This is despite the comparatively poor performance of the message passing (as shown in Figure 5) which is most likely due to the use of PVM.

5.2.3 Comparison with Theoretical Model

A comparison of the theoretical and measured parallel speedups for a two-dimensional decomposition ($R = 64$) appears in Figure 13. The theoretical curve was generated by substituting the measured values contained in Table 2 into equation (14). The measured speedup is somewhat over predicted, but this is not surprising since the theoretical model neglects such issues as load imbalance and message contention. It is therefore expected that this model provides a best-case performance prediction. The behavior of the model with respect to variations in the ratio $t_w/t_c \equiv \chi$ is also shown in the figure. Using the value for t_w and t_c found in Table 2 yields the ratio $\chi = 0.0003$. A value of $\chi = 0.0009$ is also shown for reference purposes. It is not unreasonable to expect fairly large variations in χ for different decompositions due to differences in memory accessing patterns. In fact, these variations are demonstrated in section 5.2.4.

Although the theoretical model was able to predict the performance of certain decompositions on the T3D to an acceptable manner, such was not the case for the Paragon and the SP2. In these cases, the model drastically over predicted parallel performance. Since it is known that χ plays a primary role in parallel performance, a more realistic test problem was conceived to measure t_w. Instead of utilizing a ring structure in which a single message is in transit at any given instant, processors were configured as a logical three-dimensional mesh of dimension $2 \times 2 \times 2$ and $4 \times 4 \times 4$. Nearest neighbors in the mesh simultaneously exchanged messages of varying lengths along each coordinate direction and the times for 1000 exchanges in each direction were recorded. The results of this experiment are contained in Figure 14. In the absence of message contention, the message-passing times are expected to be identical regardless of processor number or direction of message exchange since no two messages simultaneously transit the same logical connection between processors. In reality, however, the mapping of the logical three-dimensional structure to a lower-dimensionality architecture results in the mapping of more than one logical connection to the same physical connection. Although all three machines exhibit some degree of variation in message-passing times as the number of processors is increased, the effect is much less dramatic on the T3D. The deviation in the general trend is also small for the SP2, but the variations exhibited in Figure 5 become much more pronounced as the number of processors is increased. In addition to the variation with processor number, the Paragon also exhibits a directional bias in message transfer times which becomes more pronounced as the number of processors increases. In the figure, the i, j, and k notation simply identifies a direction along which the message exchanges occurred. The magnitude of the variations in message-passing times relates directly to the quality of the theoretical parallel performance model, and large variations in message-passing times are expected (and observed) to adversely impact the theoretical predictions.

5.2.4 Variations in Decomposition Times

The results contained in Figures 7-12 represent the best observed times for a given dimensionality of decomposition, processor number, and grid size. As noted in Table 3, all permutations of a given decomposition were performed for each case. Although each permutation yielded the same number of grid cells on a given processor and the same amount of message traffic, certain decompositions were observed to yield substantially better performance than others. The performance differences were quantified by constructing the ratio

$$\zeta = T_{max}/T_{min}$$

where T_{min} and T_{max} represent the minimum and maximum observed run times for each combination of processor number, grid size, and decomposition dimensionality, Figure 15 depicts this ratio for the two-dimensional decompositions on the SP2. Although the trends differed depending on the machine and decomposition approach, variations of similar magnitude were observed for the Paragon and T3D. The sometimes marked variations indicate that memory accessing issues are as important as the dimensionality of the decomposition in achieving good performance. This is to be expected since the RISC processors employed in all machines require a high degree of data locality in order to achieve near-advertised megaflop ratings. Should the domains be decomposed in a manner that precludes locality-

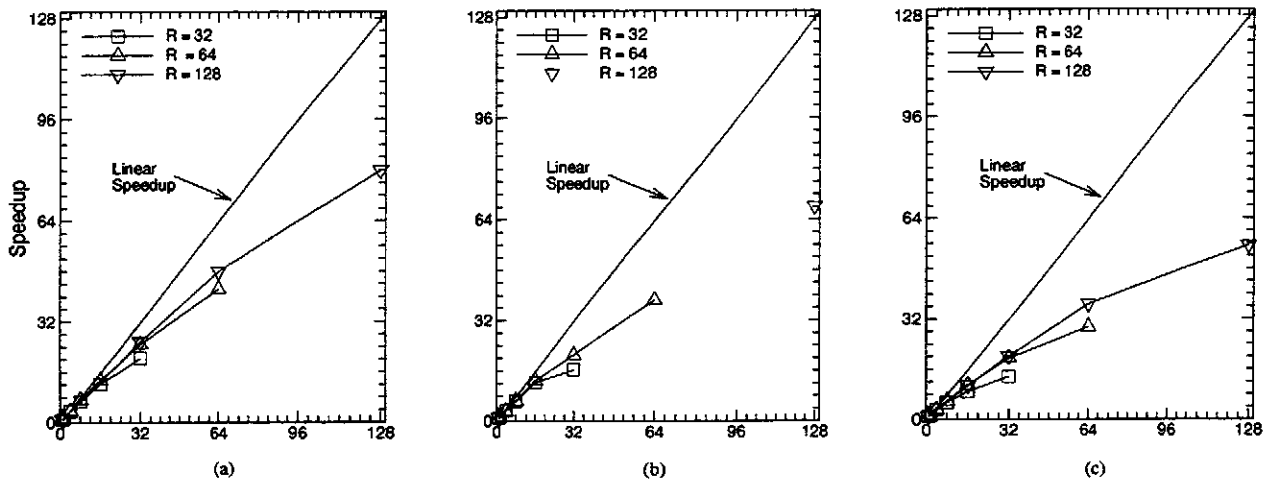

Figure 7: One-Dimensional Decomposition Speedups a) T3D, b) Paragon, c) SP2

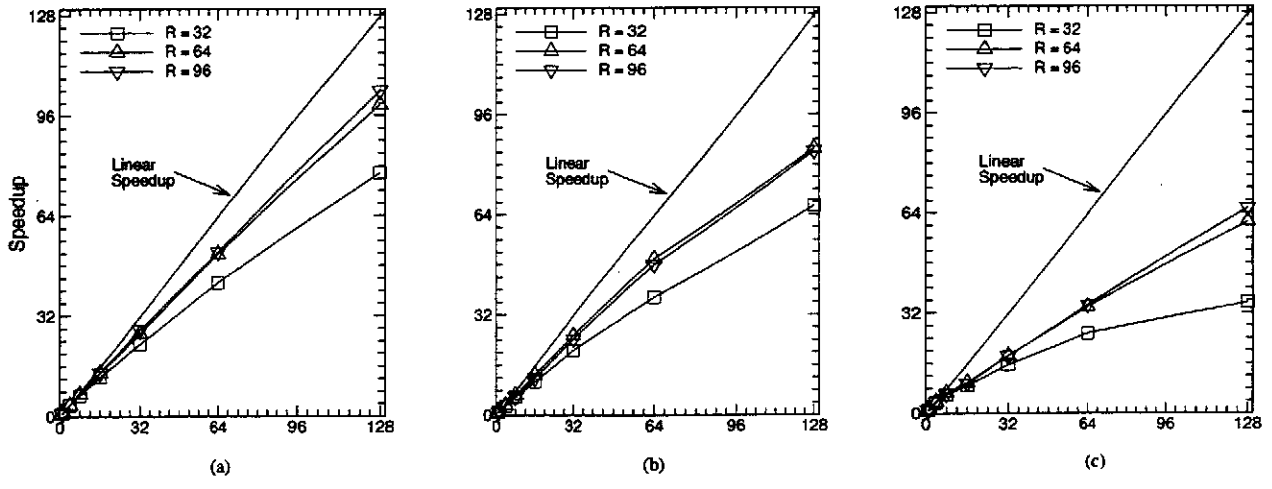

Figure 8: Two-Dimensional Decomposition Speedups a) T3D, b) Paragon, c) SP2

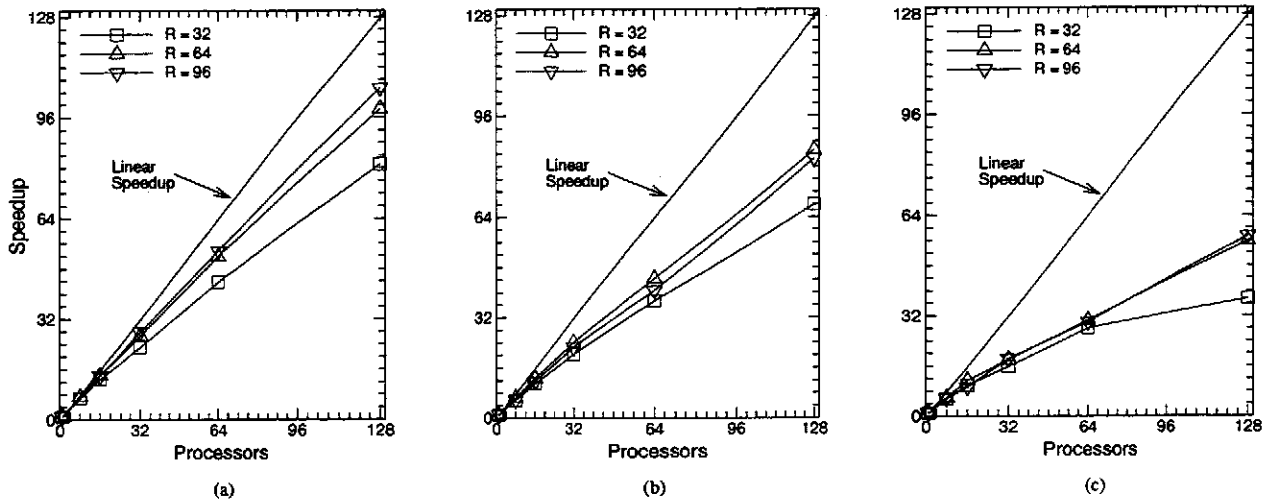

Figure 9: Three-Dimensional Decomposition Speedups a) T3D, b) Paragon, c) SP2

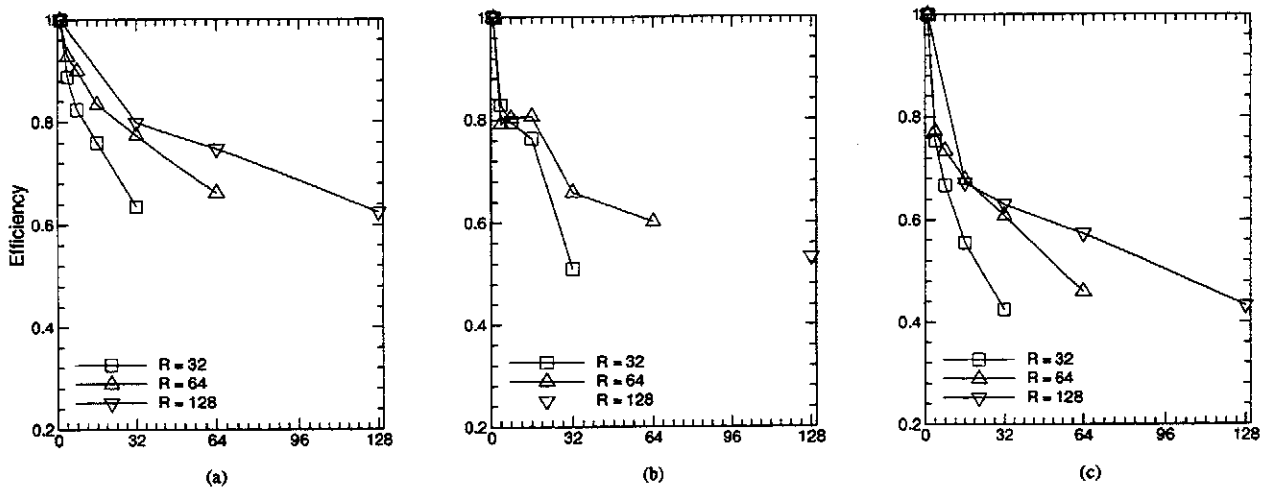

Figure 10: One-Dimensional Decomposition Efficiencies a) T3D, b) Paragon, c) SP2

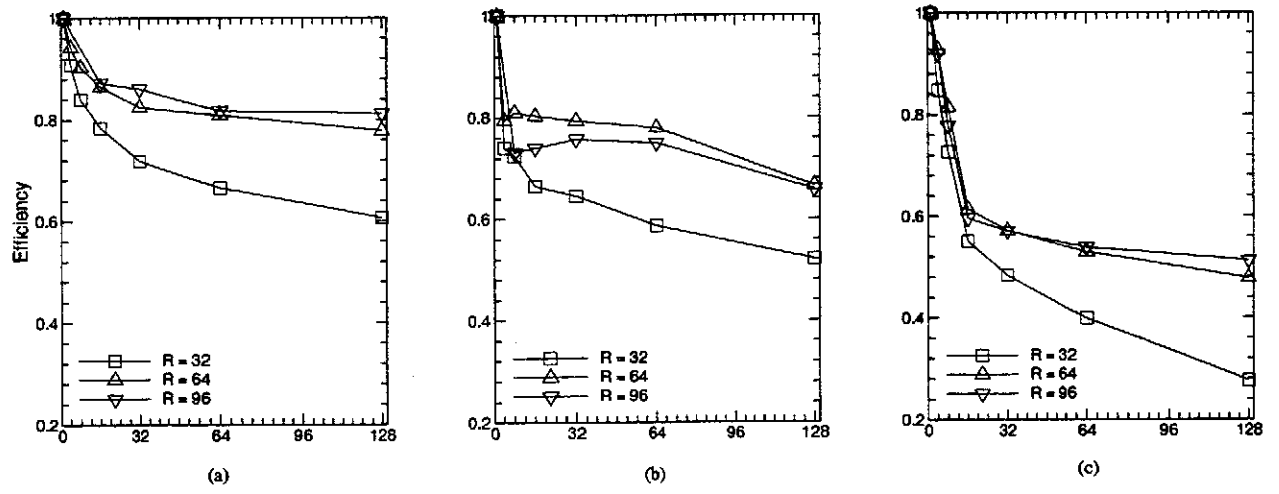

Figure 11: Two-Dimensional Decomposition Efficiencies a) T3D, b) Paragon, c) SP2

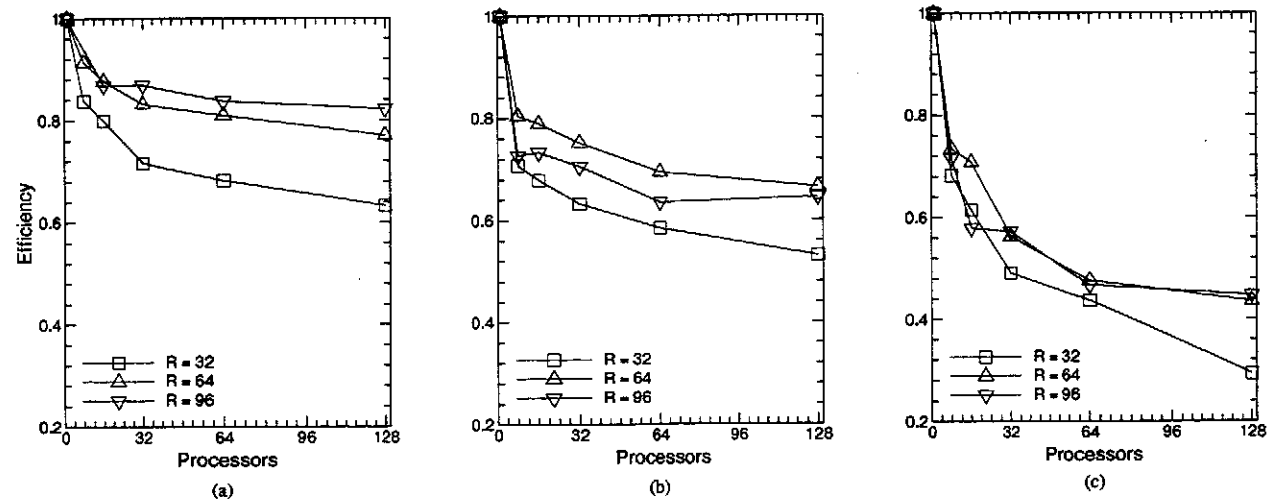

Figure 12: Three-Dimensional Decomposition Efficiencies a) T3D, b) Paragon, c) SP2

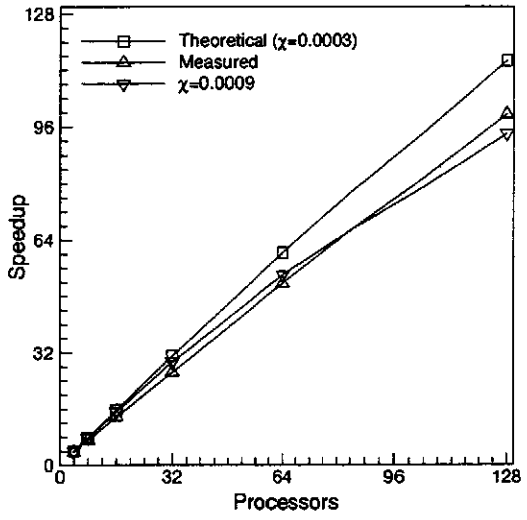

Figure 13: Theoretical and Measured Performance of the T3D
(R = 64 , 2D decomposition)

of-reference, then performance suffers dramatically.

6 Conclusions

The relation between the dimensionality of the domain decomposition and parallel performance has been assessed on three modern distributed memory parallel computing platforms using a FVTD algorithm and a rectangular waveguide geometry. Higher dimensionality (two- and three-dimensional) decompositions were found to nearly always outperform one-dimensional decompositions. The performance of the decompositions was found to relate very closely to the topology of the machine on which the algorithm was implemented. In general, machines with higher processor connectivity favored higher-dimensional decompositions.

Despite the fact the classical parallel performance model used in this study accurately predicted performance trends, it occasionally dramatically over predicted the actual performance of the algorithm. This is due to the fact that the model does not account for issues such as message contention which occurs when more than one logical message path is mapped to the same physical connection. The adverse affect of message contention was observed to increase with increasing processor count on all architectures.

Although caution must be exercised when attempting to extend program performance characteristics to other algorithms, because many algorithms designed for solving hyperbolic systems of partial differential equations possess similar data dependencies, the results presented here can be generalized at least to some degree to a large number of explicit hyperbolic (and perhaps even some parabolic and elliptic) partial differential equations solution schemes.

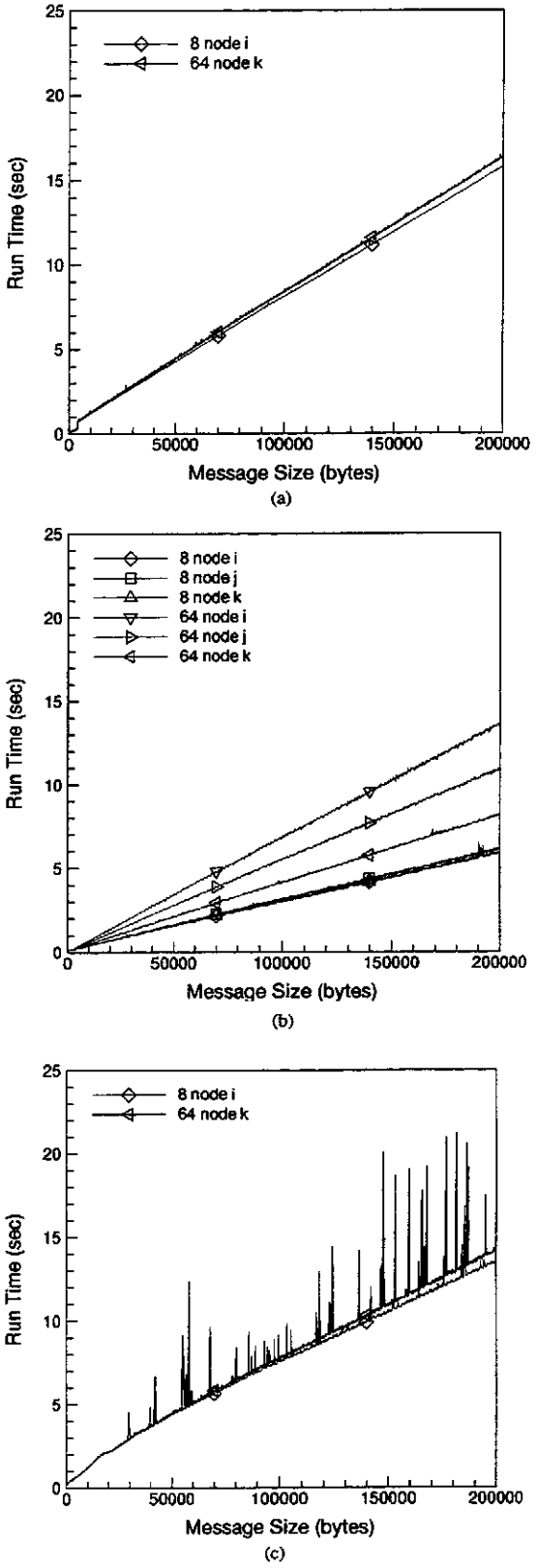

Figure 14: Nearest-Neighbor Message Passing Performance a) T3D, b) Paragon, c) SP2

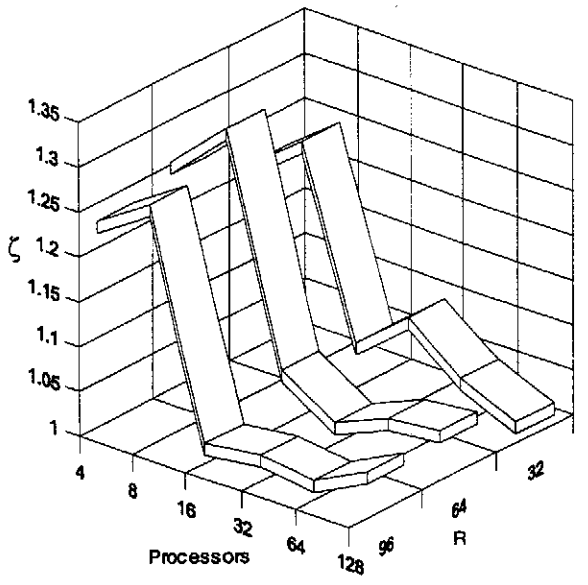

Figure 15: Example of Maximum to Minimum Execution Time Ratio
(2D decompositions on SP2)

Acknowledgments

The authors wish to thank Dr. Joseph Shang of Wright Laboratories for sponsoring this work and providing the original sequential FVTD computer code.

References

[1] Blosch, E. L. and Shyy, W., "Parallel Efficiency of Sequential Pressure-Based Navier-Stokes Algorithms on the CM-2 and MP-1 SIMD Computers," AIAA Paper 94-0409, January 1994.

[2] Hauser, J., and Williams, R., "Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines," *International Journal for Numerical Methods in Fluids*," Vol. 15, pp. 51-58, 1992.

[3] Scherr, S. J., "Implementation of an Explicit Navier-Stokes Algorithm on a Distributed Memory Parallel Computer," AIAA Paper 93-0063, January 1993.

[4] Shang, J. S., Hill, K. C., and Calahan, D., "Performance of a Characteristic-Based, 3-D, Time-Domain Maxwell Equations Solver on a Massively Parallel Computer," AIAA Paper 93-3179, July 1993.

[5] Shankar, V., Hall, W. F., Mohammadian, A. and Rowell, C., "Computational Electromagnetics: Development of a Finite-Volume, Time-Domain Solver for Maxwell's Equations," Final Contract Report of contract N62269-90-C-0257, May 1993.

[6] Stagg, A. K., Cline, D. D. and Carey, G. F., "Parallel, Scalable Parabolized Navier-Stokes Solver for Large-Scale Simulations," *AIAA Journal*, Vol. 33, No. 1, Jan. 1995.

[7] Wong, C. C., Blottner, F. G., Payne, J. L. and Soetrisno, M., "A Domain Decomposition Study of Massively Parallel Computing in Compressible Gas Dynamics," AIAA Paper 95-0572, January 1995.

[8] Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley Publishing Company, 1995.

[9] PVM and HeNCE Programmer's Manual, SR-2501 5.0, Cray Research, Inc., 1994.

[10] Paragon C System Calls Reference Manual, Intel Corporation, Order Number 312487-003, June 1994.

[11] Snir, M., Otto, S., Huss-Lederman, S., Walker, D. W., and Dongarra, J., *MPI: The Complete Reference*, MIT Press, 1996.

[12] Shang, J. S., "A Fractional-Step Method for Solving 3-D Time-Domain Maxwell Equations," *Journal of Computational Physics* 118, pp. 109-119, 1995.

[13] Southworth, G. C., *Principles and Applications of Waveguide Transmission*, D. Van Nostrand Co., Princeton, New Jersey, 1956.

[14] Balanis, C. A., *Advanced Engineering Electromagnetics*, John Wiley & Sons, 1989.

[15] Mohammadian, A. H., Shankar, V. and Hall, W. F., "Application of Time-Domain Finite-Volume Method to Some Radiation Problems in Two and Three Dimensions", *IEEE Transactions on Magnetics*, vol. 27, No. 5, September 1991.

[16] Steger, J. L. and Warming, R. F., "Flux Vector Splitting of the Inviscid Gasdynamics Equations with Application to Finite Difference Methods," *Journal of Computational Physics*, Vol. 40, pp. 263-93, April 1981.

[17] van Leer, B., "Flux-Vector Splitting for the Euler Equations," Technical Report 82-30, ICASE, September 1983.

[18] Shang, J. S. and Gaitonde, D., "Characteristic-Based, Time-Dependent Maxwell Equations Solvers on a General Curvilinear Frame," AIAA Paper 93-3178, July 1993.

[19] Kumar, V., Grama, A., Gupta, A. and Karypis, G., *Introduction to Parallel Computing*, Benjamin/Cummings Publishing Company, 1994.