component ($\overline{k'v}$) is generally predicted well in the region far from the wall by all three models, although the EVM provides excessive overprediction close to the wall ($y^+ \sim 10$).

In contrast to this, the predictive performance of $\overline{k'u}$ (in the x direction) shows significant differences between the models. Because the turbulent diffusion vector predicted by the EVM completely aligns with the spatial gradient of k, the EVM cannot predict $k'u$ as shown in Fig. 2a [see Eq. (3)]. The GGDH model has some possibility of predicting the streamwise component by introducing the Reynolds-stress anisotropy. As seen in Fig. 2b, however, the GGDH model gives extreme underprediction of $k'u$ and, thus, its predictive performance still leaves quite a wide margin for improvement.

On the other hand, the quadratic model generally gives a reasonable prediction for both $k'u$ and $k'v$, although some underprediction is seen in the limited region around $y^+ \sim 30$. Note that no previously proposed explicit algebraic model for Eq. (1) could capture this basic feature of $k'u$ as precisely as the present quadratic model does. Hence, it is expected that the quadratic model will lead to more advanced algebraic turbulent diffusion models with further modification for the model coefficient C_k.

IV. Conclusions

To improve the prediction accuracy for the turbulent diffusion term appearing in two-equation models, a priori estimations have been performed by processing the LES data of a fundamental channel flow. In this study, some representative algebraic expressions were reexamined carefully, and we proposed a new way of modeling the turbulent diffusion. It introduces the quadratic products of the Reynolds-stress tensor, appropriately reflecting the knowledge obtained from the present investigation. A priori tests show that the model has the basic capability of predicting the turbulent diffusion term much more precisely, not only in the wall-normal direction, but also in the streamwise direction.

References

[1]Daly, B. J., and Harlow, F. H., "Transport Equations in Turbulence," *Physics of Fluids*, Vol. 13, 1970, pp. 2634–2649.

[2]Hanjalic, K., and Launder, B. E., "A Reynolds Stress Model of Turbulence and Its Application to Thin Shear Flows," *Journal of Fluid Mechanics*, Vol. 52, 1972, pp. 609–638.

[3]Magnaudet, J., "Modelling of Inhomogeneous Turbulence in the Absence of Mean Velocity Gradient," *Applied Scientific Research*, Vol. 51, 1993, pp. 525–531.

[4]Hanjalic, K., "Advanced Turbulence Closure Models: A View of Current Status and Future Prospects," *International Journal of Heat and Fluid Flow*, Vol. 15, 1994, pp. 178–202.

[5]Amano, R. S., Goel, P., and Chai, J. C., "Turbulence Energy and Diffusion Transport of Third-Moments in a Separating and Reattaching Flow," *AIAA Journal*, Vol. 26, No. 3, 1988, pp. 273–282.

[6]Nagano, Y., and Tagawa, M., "A Structural Turbulence Model for Triple Products of Velocity and Scalar," *Journal of Fluid Mechanics*, Vol. 215, 1990, pp. 639–657.

[7]Kawamura, H., Sasaki, J., and Kobayashi, K., "Budget and Modelling of Triple-Moment Velocity Correlations in a Turbulent Channel Flow Based on DNS," *Proceedings of 10th Symposium on Turbulent Shear Flows*, Pennsylvania State Univ., University Park, PA, 1995, pp. 26.13–26.18.

[8]Craft, T. J., Launder, B. E., and Suga, K., "Prediction of Turbulent Transitional Phenomena with a Nonlinear Eddy-Viscosity Model," *International Journal of Heat and Fluid Flow*, Vol. 18, 1997, pp. 15–28.

[9]Abe, K., Kondoh, T., and Nagano, Y., "On Reynolds-Stress Expressions and Near-Wall Scaling Parameters for Predicting Wall and Homogeneous Turbulent Shear Flows," *International Journal of Heat and Fluid Flow*, Vol. 18, No. 3, 1997, pp. 266–282.

[10]Abe, K., and Suga, K., "Towards the Development of a Reynolds-Averaged Algebraic Turbulent Scalar-Flux Model," *International Journal of Heat and Fluid Flow*, Vol. 22, No. 1, 2001, pp. 19–29.

[11]Suga, K., and Abe, K., "Nonlinear Eddy Viscosity Modelling for Turbulence and Heat Transfer Near Wall and Shear-Free Boundaries," *International Journal of Heat and Fluid Flow*, Vol. 21, No. 1, 2000, pp. 37–48.

R. M. C. So
Associate Editor

Parallel Performance Analysis of FVTD Computational Electromagnetics Code

José A. Camberos* and Michael D. White†
U.S. Air Force Research Laboratory,
Wright–Patterson Air Force Base, Ohio 45433

Introduction

COMPUTATIONAL processing capability continues to limit the complexity and size of numerical simulations. Advancements in computer hardware technology, notably the development of powerful reduced instruction set computing microprocessors, high-density dynamic random access memory, and ultrahigh-speed switching networks now allow the construction of machines with the power to run applications across hundreds to thousands of processors that outstrip the performance of traditional vector supercomputers. This advancement in hardware still requires added effort in algorithm design because intelligent compilers for these machines do not yet fully exploit the parallelism in a computer code or map that parallelism to a distributed memory environment. It remains, therefore, a crucial task to design and test software for the best possible parallel performance across a variety of machines. At present, the popular practice of domain decomposition achieves a high degree of parallelism in grid-based engineering applications of computational electromagnetics (CEM) and computational fluid dynamics (CFD).

Because the Maxwell equations form a set of hyperbolic partial differential equations, like the equations of fluid dynamics, methods developed in CFD apply also to CEM. The finite difference time-domain approach on staggered grids was pioneered by Yee[1] in 1966, and Shankar et al.[2] developed a finite volume approach with the method originally developed by Lax and Wendroff.[3] Of interest also is the technique introduced by Shang[4] incorporating the flux-vector splitting methodology of Steger and Warming,[5] well known in CFD. This technique has been implemented and tested against theoretical solutions, experimental data, and frequency-domain methods.[6] To continue the development of this technique with parallel machines, the finite volume time-domain electromagnetic computer code CHARGE was developed by Blake[7] at Wright–Patterson Air Force Base. A domain decomposition strategy was also implemented and analyzed by Blake and Buter,[8] who concluded that although theory favored higher-dimensional decompositions for superior performance (in terms of scalability), this did not always result in practice. That work also indicated that parallel performance improved when the dimensionality of the domain decomposition closely matched the physical topology of the underlying machine architecture. One question that remained was whether an automatic domain decomposition strategy could evenly divide the work among a given number of processors (load balancing) regardless of machine architecture (portability). Rapid advances in computer hardware also required that the electromagnetic computer code again be compiled and tested on a variety of platforms. This study, therefore, focused on the portability and scalability of the CEM code on the massively parallel machines currently available.

Theoretical Background

Maxwell equations are well known to describe adequately electromagnetic phenomena of engineering interest. Constitutive relations for simple media (linear, isotropic, and homogeneous) include $\boldsymbol{D} = \epsilon \boldsymbol{E}$, $\boldsymbol{B} = \mu \boldsymbol{H}$, and $\boldsymbol{J} = \sigma \boldsymbol{E}$. These allow some flexibility in

Received 14 July 2000; revision received 22 June 2001; accepted for publication 4 July 2001. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

*Aerospace Engineer, Computational Sciences Branch, Aeronautical Sciences Division, Air Vehicles Directorate. Senior Member AIAA.

†Computational Sciences Branch; also Visiting Scientist, OAI/ICOMP, Cleveland, OH 44142. Member AIAA.

writing the equations; with $\boldsymbol{D}$ and $\boldsymbol{B}$ representing the electric and magnetic field variables, respectively, the Maxwell equations in vacuo are

$$\nabla \times \frac{\boldsymbol{D}}{\epsilon_0} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{1}$$

$$\nabla \times \frac{\boldsymbol{B}}{\mu_0} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J} \tag{2}$$

$$\nabla \cdot \boldsymbol{D} = \rho \tag{3}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{4}$$

where ϵ_0 and μ_0 represent free-space conditions and relate to the speed of light squared as $c^2 = 1/\epsilon_0\mu_0$.

Limited only by one's preference, constitutive relations can be used to write equally valid expressions using the $\boldsymbol{B}$ and $\boldsymbol{E}$ variables, or the $\boldsymbol{H}$ and $\boldsymbol{D}$ variables, or the $\boldsymbol{H}$ and $\boldsymbol{E}$ variables. For practical purposes typical of engineering applications, only the curl equations (1) and (2) are solved; the divergence conditions (3) and (4) are degeneracies of Eqs. (1) and (2) in free space (where $\boldsymbol{J}$ and ρ are zero). Equations (1) and (2) may then be written symbolically as a single equation:

$$\frac{\partial \boldsymbol{\mathcal{Q}}}{\partial t} + \frac{\partial \boldsymbol{\mathcal{F}}_1}{\partial x_1} + \frac{\partial \boldsymbol{\mathcal{F}}_2}{\partial x_2} + \frac{\partial \boldsymbol{\mathcal{F}}_3}{\partial x_3} = \mathcal{R} \tag{5}$$

where the state and source vectors are

$$\boldsymbol{\mathcal{Q}} = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{D} \end{bmatrix}, \qquad \mathcal{R} = -\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{J} \end{bmatrix} \tag{6}$$

respectively. The flux vectors $\boldsymbol{\mathcal{F}}_1$, $\boldsymbol{\mathcal{F}}_2$, and $\boldsymbol{\mathcal{F}}_3$ represent the remaining curl operator terms (see Camberos and White[9] for the complete formulas).

To simplify the formulas, all variables are nondimensionalized so that the speed of light in free space equals unity. Transformed to a general curvilinear coordinate system (ξ, η, ζ), these equations become

$$\frac{\partial \hat{\boldsymbol{\mathcal{Q}}}}{\partial t} + \frac{\partial \hat{\boldsymbol{\mathcal{F}}}_1}{\partial \xi} + \frac{\partial \hat{\boldsymbol{\mathcal{F}}}_2}{\partial \eta} + \frac{\partial \hat{\boldsymbol{\mathcal{F}}}_3}{\partial \zeta} = \hat{\mathcal{R}} \tag{7}$$

where the carat represents the transformed quantities such that $\mathcal{J}\hat{\boldsymbol{X}} = \boldsymbol{X}$, where $\mathcal{J}$ equals the coordinate transformation Jacobian. For time-invariant hexahedral computational cells, the semidiscrete formula for solving Eq. (7) is

$$\mathcal{V}\frac{\mathrm{d}\hat{\boldsymbol{\mathcal{Q}}}}{\mathrm{d}t} + \sum_{j=1}^{6} \hat{n} \cdot \boldsymbol{F}_j \, \mathrm{d}A_j - \mathcal{V}\hat{\mathcal{R}} = 0 \tag{8}$$

where $\boldsymbol{F} = \hat{\boldsymbol{\mathcal{F}}}_1\hat{e}_\xi + \hat{\boldsymbol{\mathcal{F}}}_2\hat{e}_\eta + \hat{\boldsymbol{\mathcal{F}}}_3\hat{e}_\zeta$ and $\mathcal{V}$ equals the cell volume. For a structured grid, the transformed coordinates usually are orthogonal, and, therefore, only one of the flux components is nonzero in a given direction. The unit vector $\hat{\boldsymbol{n}}$ is normal to the cell face $\mathrm{d}A_j$ and the transformed coordinate system has unit direction vectors $(\hat{e}_\xi, \hat{e}_\eta, \hat{e}_\zeta)$. The surface fluxes required in Eq. (8) are obtained using a technique originally developed by Steger and Warming[5] for CFD. In this flux-vector splitting scheme, the fluxes are decomposed according to the sign of the eigenvalues associated with the flux Jacobian matrices. For cell faces oriented in a given direction the flux vectors are calculated with the formula

$$\hat{\boldsymbol{\mathcal{F}}} = \hat{\boldsymbol{\mathcal{F}}}^+(\hat{\boldsymbol{\mathcal{Q}}}^L) + \hat{\boldsymbol{\mathcal{F}}}^-(\hat{\boldsymbol{\mathcal{Q}}}^R) \tag{9}$$

where L and R denote values of the state variables as reconstructed at the left (inner) and right (outer) sides of a cell surface. Reconstruction of the state variables determines the spatial order of accuracy. When van Leer's MUSCL[10] code is used, the state variables at the cell interface are

$$\hat{\boldsymbol{\mathcal{Q}}}^L = \hat{\boldsymbol{\mathcal{Q}}}_j + \tfrac{1}{4}\phi\left(\Delta\hat{\boldsymbol{\mathcal{Q}}}_j + \kappa\Delta^2\hat{\boldsymbol{\mathcal{Q}}}_j\right) \tag{10}$$

$$\hat{\boldsymbol{\mathcal{Q}}}^R = \hat{\boldsymbol{\mathcal{Q}}}_{j+1} - \tfrac{1}{4}\phi\left(\Delta\hat{\boldsymbol{\mathcal{Q}}}_{j+1} - \kappa\Delta^2\hat{\boldsymbol{\mathcal{Q}}}_{j+1}\right) \tag{11}$$

where

$$\Delta\hat{\boldsymbol{\mathcal{Q}}}_j = \hat{\boldsymbol{\mathcal{Q}}}_{j+1} - \hat{\boldsymbol{\mathcal{Q}}}_{j-1} \tag{12}$$

$$\Delta^2\hat{\boldsymbol{\mathcal{Q}}}_j = \hat{\boldsymbol{\mathcal{Q}}}_{j-1} - 2\hat{\boldsymbol{\mathcal{Q}}}_j + \hat{\boldsymbol{\mathcal{Q}}}_{j+1} \tag{13}$$

A value of $\phi = 0$ gives a first-order scheme (the original Steger–Warming[5] technique). As implemented, the parameters ϕ and κ are set equal to 1 and $\frac{1}{3}$, respectively, for third-order accuracy on a regular mesh. With formulas for computing the required surface fluxes in Eq. (8), the equation is then integrated in time using a two-stage Runge–Kutta scheme that is second-order accurate, as detailed by Blake and Buter.[8] For stability, the time-step restriction uses the requirement obtained by Weber[11] for this kind of solution technique.

Boundary conditions (BC) for the current results are limited to the perfect electrically conducting (PEC) surface and the far-field radiation boundary. The far-field BC truncates the flux-vector splitting calculation so that only interior cells are used. The PEC surface requires that the tangential electric and the normal magnetic fields vanish, whereas the normal derivatives of the contravariant normal electric and tangential magnetic fields are zero. Details of the implementation may be found elsewhere.[9]

Computer Code Implementation

Solving the electromagnetic equations as given by Eq. (8) is only part of the simulation process. In general, the total solution procedure requires grid generation about a given geometry, domain decomposition, solution of the electromagnetic equations, and post-processing of results. Other than grid generation, the CHARGE distribution package[12] contains computer code(s) designed to automate mostly the solution process. The code automatically determines grid connectivity for arbitrarily complex multi block grids and interpolation stencils (for over set grids) with a search strategy that eliminates the problem of producing orphan points like other techniques.[13] A separate preprocessing program automatically completes the domain decomposition by combining all grids into a single, unstructuredlike grid that is then partioned in a balanced manner. This nearly ideal load balancing technique results in good parallel performance across different computer architectures. The decomposition technique is known as the recursive coordinate bisection method,[14] and it provides good load balancing when using 2^n computational nodes, where $n \in [1, 2, \ldots)$. In serial mode, no decomposition is needed. Blake and Shang[15] give a more detailed description of all algorithms and procedures implemented in the CHARGE computer code package. Another unique feature of the code is the abstraction of library-specific calls in the C programming language. This allows a single code implementation for use with different parallel message passing libraries including Parallel Virtual Machine (PVM) and Message Passing Interface (MPI).[15]

It is well known that balancing work load and minimizing communication between processors are essential for the effective use of parallel machines. In programming numerical procedures for solving the electromagnetic equations, another requirement for effective use of parallel computers has been highlighted: the cache memory and memory hierarchy utilization.[15] Also, for practical applications, a computer program must be capable of solving a wide range of problems with a minimum of modifications. CFD has met this goal with either the overset grid and/or the unstructured grid techniques. Either way, the key to accommodate complex geometric shapes hinges on the mapping of all grid topologies onto a common framework, which allows for balanced work loads with good domain decomposition. Unstructured grids require uniform and redundant references in cell connectivity, an ideal approach for the widest range of applications. In parallel computing using domain decomposition, the same common data structure automatically satisfies connectivity across the partitioned domain. Once cell connectivity is established, calculations are made using a cell-based approach. This feature is unique to the computer program CHARGE, where data is processed one cell at a time until the whole database is updated in time, reducing

communication time between processors significantly in comparison with standard array-based operations.[15]

Terminology and Definitions

Parallel programs can potentially solve bigger problems in less time. To evaluate how well a computer program performs in parallel mode, several measures of performance are available. The most common are speed up and efficiency.[16] Speed up is typically defined as the ratio of program execution time on a serial machine to the execution time on a parallel machine. Because large-scale problems may exceed the capacity of the hardware to be run serial, the present work considers a more general definition of a relative speed up and efficiency based on a reference run on some small number of processors. In the ideal, the speed up will scale linearly with the number of processors. In terms of efficiency, a program exhibits linear speed up when the efficiency equals one. Typically, efficiency will be less than or equal to one. If it drops below the inverse of the processors used, the program has experienced slow down and very poor performance due to excessive communication between processors.

Relative Speed Up

Parallel performance was quantified by defining relative speed up as the ratio of the run time based on a reference number of processors (typically one or two, but for this case we used eight because of memory limitations for very large-scale problems) and the run time for n_p processors:

$$S(n_p) \equiv [t(n_0)/t(n_p)] \times n_0 \tag{14}$$

where n_0 equals the reference number of processors. The run time $t(\#)$ can be set equal to the program execution time for a fixed number of calculations. In this work, the run time equals the average time required to compute one period of radar cross section (RCS) data calculations. Other definitions of parallel speed up exist, and so the reader may need to discern the correct meaning when comparing different results.

Efficiency

Measuring parallel performance can also be based on the efficiency of a parallel code, defined as the ratio of the relative speed up to the number of processors used:

$$\varepsilon(n_p) \equiv S(n_p)/n_p = t(n_0)/t(n_p) \times (n_0/n_p) \tag{15}$$

where Eq. (14) was used to expand the expression.

Data Processing Time

The time required to process a computational instruction can also be used to assess the parallel performance of a code. For this work, this is defined as the average run time divided by the number of cells and iterations required per period of RCS calculations. This gives an estimate of the data processing time (DPT) defined by the formula

$$\text{DPT} \equiv t(n_p)/N_c \cdot N_I \tag{16}$$

where N_c equals the number of grid cells and N_I equals the number of iterations required per period of RCS calculations. The product $N_c \cdot N_I$ is fixed for a given grid. A steady DPT allows the run time to increase linearly with a fixed number of processors as problem size increases. This is a favorable characteristic for parallel programs.

Floating Point Operation Rate

Another measure of parallel performance, although not widely used, is the floating point operation rate (FPR), defined as the ratio of the average number of floating point operations (FLOPs) per processor divided by the average run time for the given calculation:

$$\text{FPR}(n_p) \equiv \text{FLOPs}/t(n_p) \tag{17}$$

To obtain the average number of floating point operations, the ssrun utility on the Silicon Graphics, Inc. (SGI), Origin 2000 was used and switched off of the multiply–add instruction. The utility tracks the number of floating point operations per processor. From this, an average number of floating point operations per processor is obtained per period of RCS calculations. The floating point operation rate for each machine can then be obtained by dividing the result by the average run time.

Memory Usage

Although not a measure of parallel performance, memory usage indicates how much memory a given problem is using. One can then estimate the minimum number of processors required to solve very large-scale problems. The only machine that had an easy to use utility for this was the Cray T3E, which had available job accounting (JA). An estimate for typical memory usage was obtained by dividing the CPU time–memory integral by the average number of seconds of user CPU time (both given by JA).

Parallel Implementation and Compiling Options

CHARGE was compiled and executed on several different available machines. These included the IBM SP2/SP3 systems and the SGI Origin 2000 located at the Aeronautical Systems Center Major Shared Resource Center (MSRC) at Wright–Patterson Air Force Base, Ohio, and a Cray T3E located at the Naval Oceanographic Office MSRC at the Stennis Space Center, Mississipi. Basic hardware features of each machine are summarized at the respective MSRC home pages on the World Wide Web.

Compilations on all four platforms used the $-O3$ option for highly optimized code. In addition, the most aggressive in-lining option was used as available. This kind of aggressive optimization has the potential to alter a code's basic algorithm and can affect computational results. Standard mathematic libraries and MPI, which has become the industry standard, was used on all machines.

Problem Definition and Geometry

The geometry consisted of a generic finned-missile model 21.5 in. long. The surrounding computational grid was constructed and modified using commercial software.[17] The first structured block grid contained 2.31×10^6 cells. A larger grid was also used containing 4.77×10^6 cells. The largest grid had 10.72×10^6 cells and was used to test parallel scalability with problems of increasing size. To calculate DPT for a given grid requires the number of cells and number of iterations per period of RCS calculations. These are constants for a fixed grid and are automatically calculated in setting up grid connectivity and estimating time-step size. For grid 1, $N_c \cdot N_I = 1.4 \times 10^9$; for grid 2, $N_c \cdot N_I = 2.0 \times 10^{10}$; and for grid 3, $N_c \cdot N_I = 9.2 \times 10^{15}$. An electromagnetic sinusoidal wave varying at 4 GHz illuminated the target.

Data Collection and Reduction Methods

CHARGE solves the electromagnetic equations in the scattered field formulation and calculates the RCS for a given target. Output includes the values of dependent variables (electric and magnetic fields) and the RCS after each period. Convergence is reached when the RCS varies by 1% or less. For this problem, this stage was reached at about 20 periods of RCS calculations. The execution time for a given number of periods then represents the measured run time to be used in data reduction. The number of processors is set at the beginning of program execution. From these two numbers, one can obtain the average run time per period, relative speed up, and relative efficiency. On the SGI Origin 2000 systems, program execution time was obtained for 5, 10, 15, 20, and 25 cycles combined with using 8, 16, 32, 64, and 128 processors. Program execution time on all of the other machines was obtained for 5 and 10 cycles, combined with using 8, 16, 32, 64, and 128 processors.

Calculating DPT requires the average run time and the problem size parameter equal to the product of grid cell number and iterations required per period of RCS calculations as defined earlier. Floating point operation rate and memory usage required special utility functions available only on some of the parallel systems used. These included ssrun on the SGI O2K and JA on the Cray T3E. On the SGI O2K, ssrun tracks the number of floating point operations

per processor from which an average can be obtained. Because the same problem was solved on all four parallel systems tested, the FLOP rate is determined for each system once the average run time is obtained. The number of FLOPs per processor were obtained on the SGI O2K using 8, 16, and 32 processors. Memory usage on the Cray T3E was obtained with the JA utility using 8, 16, and 32 processors.

Average run time for the SGI O2K was calculated by dividing total program execution time by the number of periods specified, then averaging the numbers obtained. Alternatively, one may print the time required per period during a calcuation. This then gives an average run time per period of RCS calculations by combining data from the 5- and 10-cycle computations. Another way to obtain run time per period of RCS calculations would be to add all of the execution times for a given number of processors and divide by the total number of periods represented. This is not a true average but is still representative of the time required to execute the program one cycle on each machine.

Results and Discussion

Results of the average run times for each machine were tabulated by Camberos and White.[9] The average run time per period decreased as the number of processors increased. A program exhibits good scalability if the execution time is approximately cut in half as the number of processors doubles. Ideally, this would be perfect linear speed up. Good program scalability can be observed in the general trend of the data for all four machines. Figure 1 depicts parallel performance as measured by relative speed up. The results show a maximum speed up from 84.59 to 119.88 times the extrapolated single processor performance. The best performance was observed on the Cray T3E with a speed up of 119.88. Following this were the relative speed-up values of 99.40 and 96.69 on the SGI-O2K and the IBM-SP3, respectively. As one might predict from the hardware specifications, the speed up on the IBM SP2 trailed the performance on the other machines: The CEM code's relative speed up was only 84.59 at the maximum number of processors tested (128). Parallel computation maintains an efficiency of 93.7% on the T3E with 128 processors. This drops to 77.7 and 75.5% for the O2K and SP3, respectively. Parallel performance on the IBM-SP2 trails at 66.1%.

The performance level of the code on the Cray T3E is noteworthy. In part, the success results from the T3E architecture that configures the system nodes in a three-dimensional torus. The T3E also has the highest bisection bandwidth for memory accessing among the four platforms tested. This translates into more effective parallel computing as the number of processors increases, with a corresponding increase in communication time. On any machine, maximum speed up and efficiency are realized by balancing the workload per processor, minimizing interprocessor communication, and managing cache memory effectively. In Figs. 1–4, speed up based on single processor performance is around two orders of magnitude on 128 processors with parallel efficiency still acceptable on the four platforms tested. For very large-scale numerical simulations, the maximum memory size of each processor (or node) imposes a minimum on the number of processors required. This makes the maximum speed up that a distributed memory parallel computer can achieve, relative to a single processor, unrealizable when the problem size exceeds the processor memory available. The shared-distributed memory configuration, such as the SGI-O2K, is free from this restriction.

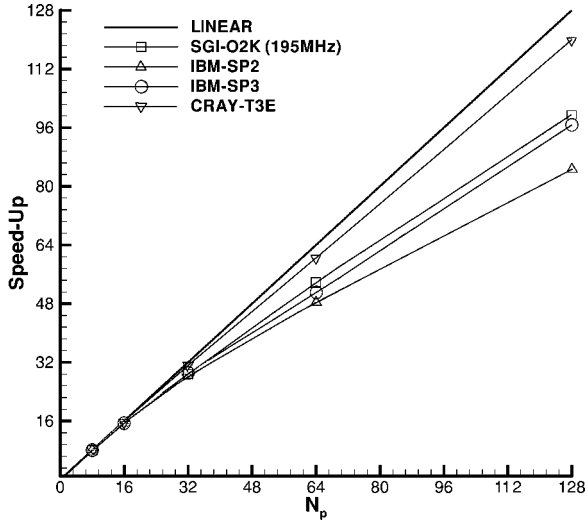

Fig. 1 Relative speed up for selected machines.

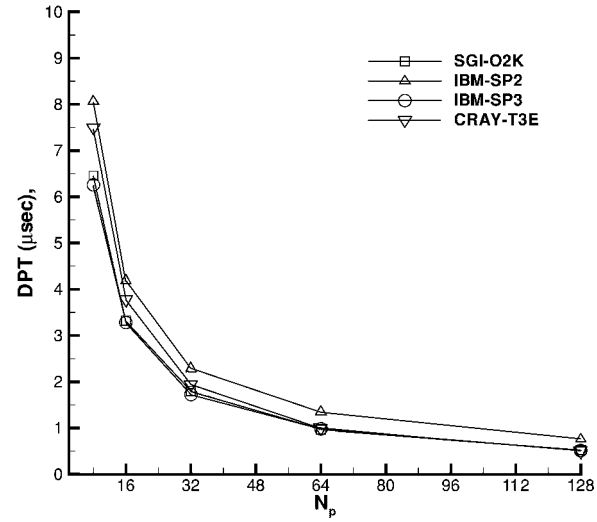

Fig. 2 Data processing rate results with 2.3×10^6 cell grid.

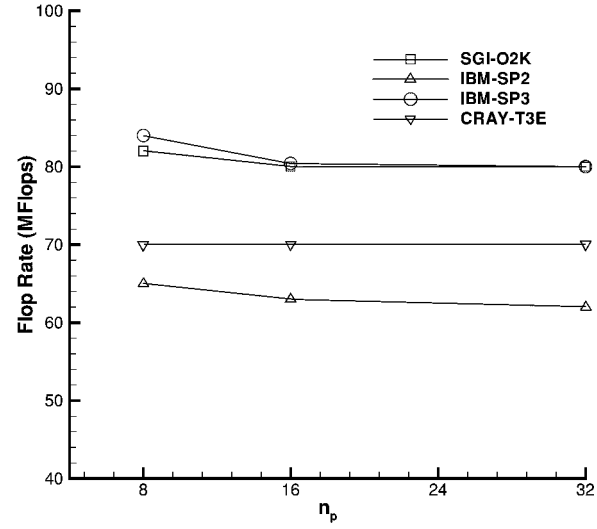

Fig. 3 FLOP rate results with 2.3×10^6 cell grid.

A different order of performance is observed in noting the data processing time shown in Fig. 2. Here a higher processing time means slower operation, an undesirable attribute. When only eight processors are used the IBM-SP2 has the highest data processing time (8.06 μs per cell per iteration). This is followed by the Cray T3E at 7.49 μs per cell per iteration. Performance of the CEM code based on data processing time on the SGI-O2K and IBM-SP3 is comparable at 6.46 and 6.26 μs per cell per iteration, respectively. When using 128 processors, the parallel performance of the code is nearly identical on the O2K, SP3, and T3E, demonstrating the more effective interprocessor communication of the Cray machine. The IBM-SP2 remains slower by about 0.763 μs per cell per iteration with 128 processors.

Figure 3 shows the FLOP rate for 8, 16, and 32 processors on each machine. A desirable trait here for any computer program is

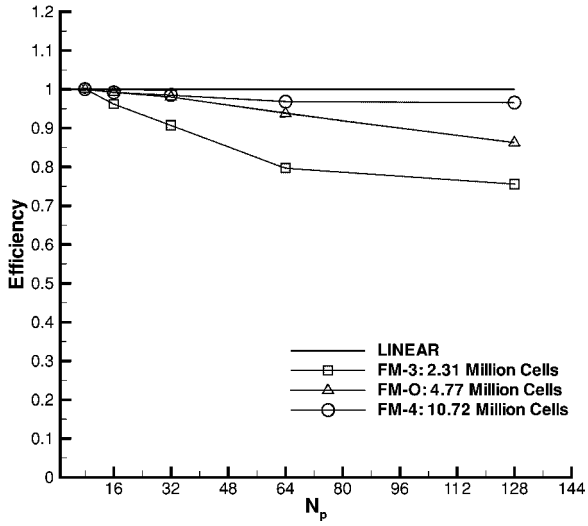

Fig. 4 Parallel efficiency on IBM-SP3.

to exhibit a steady, if not strictly constant, FLOP rate, preferably as high as possible. A steady rate means that as the number of FLOPs per processor decrease with an increasing number of processors, the execution time decreases in direct proportion. The performance of the CEM code CHARGE based on this criterion is quite good across all parallel platforms tested. From Fig. 3, the FLOP rate varies from 65 mega-FLOPs on the IBM-SP2 to 84 mega-FLOPs on the SP3. Performance on the SGI-O2K is comparable at 82 mega-FLOPs using eight processors. Performance on the Cray T3E is about 70 mega-FLOP, which is about 17% lower than the IBM-SP3. These results demonstrate that speed up and efficiency do not completely describe the parallel performance of a computer code, and a wide variety of tests should be made to ascertain a code's effective use of parallel architectures.

In addition to demonstrating the portability of the cell-based CEM code CHARGE, the tests performed in support of this work also show its scalability with problems of increasing size. On a given machine, a good parallel code should exhibit better scalability as measured by speed up or efficiency as the problem size increases. This code demonstrates exactly this behavior in Fig. 4, which depicts the relative efficiency on the IBM-SP3 with grids increasing in size. Figure 4 clearly shows that computing efficiency steadily improves as the size of the problem increases. With 128 processors used, the relative efficiency increases from 75.5 to 86.2%, and finally to 96.6% for grids comprising 2.31, 4.77, and 10.72×10^6 cells, respectively. The trend is easily explained by the reduced communication overhead in comparison to the number of FLOP operations required for each problem. The authors also obtained results on the latest SGI Origin 2000 system with R10000 processors rated at 300 MHz. C. J. Suchyta of SGI performed the calculations using 8, 16, 32, 64, 128, and 256 processors for five periods of RCS computations. Run time, speed up, efficiency, and DPT support the conclusions reached earlier. Up to 128 processors, the results look quite good and compare favorably with those obtained on the other machines.[9]

Resource utilization in terms of memory usage was obtained on the Cray T3E system using the JA utility available. Maximum memory requirements were about 1.78 GB for the 32 processor case. This indicates that for this very large-scale problem (grid comprised of 2.31×10^6 cells), a minimum number of processors is required to obtain a solution, making comparison with a serial or single processor machine impossible at present.

Conclusions

All computer simulations that solve differential equations have a physical domain of dependence. Data transfer within a computational domain is necessary. With the domain decomposition approach adopted for parallel calculations, data exchange is required at the edges of the domains. For the most effective use of parallel computer resources, the interprocessor communication overhead should be minimized by fully utilizing processor memory. Therefore, problem-specific memory requirements should be matched appropriately with available processor partitions. In short, parallel performance analyses reflect how well hardware and software are matched. Nothing is gained with vast computer resources if the software does not utilize those resources effectively. Thus, programming efforts in algorithm design remain a crucial task to exploit fully and harness the considerable power now available with parallel computing machines. The results of a performance analysis for a finite volume, time-domain, cell-based CEM code have been presented. This work supports the conclusion that the code is portable, scalable, and performs well on various massively parallel computer architectures.

Acknowledgments

A grant of High Performance Computing time from the Department of Defense Shared Resource Centers supported this work. The authors also acknowledge the sponsorship of U.S. Air Force Office of Scientific Research (Program Manager, Arje Nachman) and the leadership of Joseph Shang at the Computational Sciences Branch, Wright–Patterson Air Force Base. Special thanks to C. J. Suchyta of Silicon Graphics, Inc., for testing the code on the Origin-2000 300-MHz machine and for the many helpful tips during the course of this work.

References

[1]Yee, K. S., "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Transactions on Antennas and Propagation*, Vol. AP-14, May 1966, pp. 302–307.

[2]Shankar, V., Hall, W. F., and Mohammadian, A. H., "CFD-Based Finite Volume Procedure for Computational Electromagnetics," AIAA Paper 89-1987, June 1989.

[3]Lax, P., and Wendroff, B., "Systems of Conservation Laws," *Communications on Pure and Applied Mathematics*, Vol. 13, 1960, pp. 217–237.

[4]Shang, J. S., "Characteristic-Based Methods for the Time-Domain Maxwell Equations," AIAA Paper 91-0606, Jan. 1991.

[5]Steger, J., and Warming, R., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite Difference Methods," *Journal of Computational Physics*, Vol. 40, April 1981, pp. 263–293.

[6]Sherer, S. E., and Blake, D. C., "Validation of a FVTD Solver Using Standard Benchmark Geometries," AIAA Paper 99-0338, Jan. 1999.

[7]Blake, D. C., "Advances in Time-Domain Electromagnetic Simulation Capabilities Through the use of Overset Grids and Massively Parallel Computing," Ph.D. Dissertation, Dept. of Aeronautical and Astronautical Engineering, Air Force Inst. of Technology, Rept. AFIT/DS/ENY/97-2, Wright–Patterson AFB, OH, March 1997.

[8]Blake, D. C., and Buter, T. A., "Domain Decomposition Strategies for Solving the Maxwell Equations on Distributed Parallel Architectures," *Applied Computational Electromagnetics Society Journal*, Vol. 12, No. 3, 1997, pp. 4–15.

[9]Camberos, J. A., and White, M. D., "Performance Analysis of Finite Volume, Time-Domain, Cell-Based Computational Electromagnetic Code on Massively Parallel Platforms," AIAA Paper 2000-0958, Jan. 2001.

[10]van Leer, B., "Flux Vector Splitting for the Euler Equations," Inst. for Computer Applications in Science and Engineering, TR 82-80, Hampton, VA, Sept. 1983.

[11]Weber, Y. S., "Investigations on the Properties of a FVTD Method for Computational Electromagnetics," AIAA Paper 95-1964, Jan. 1995.

[12]Blake, D. C., White, M. D., and Camberos, J. A., *CHARGE User's Manual*, U.S. Air Force Research Lab., Wright–Patterson AFB, OH, 1999.

[13]Blake, D. C., "Application of Unstructured Grid Domain Decomposition Techniques to Overset Grids," *Eighth SIAM Conference on Parallel Processing for Scientific Computing* [CD-ROM], Society for Industrial and Applied Mathematics, 1997, pp. 1–8.

[14]Kumar, V., Grama, A., Gupta, A., and Karypis, G., *Introduction to Parallel Computing—Design and Analysis of Algorithms*, Benjamin/Cummings, New York, 1994, pp. 452, 453.

[15]Blake, D. C., and Shang, J. S., "Procedure for Rapid Prediction of Electromagnetic Scattering from Complex Objects," AIAA Paper 98-2925, Jan. 1998.

[16]Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, 1997, pp. 245–278.

[17]*Gridgen Version 12*, Pointwise, Inc., Bedford, TX, 1997.

J. Kallinderis
Associate Editor