Solving Maxwell's Equations with a Finite Difference Time Domain Method on a Cluster of Workstations in an Industrial Environment*

Anders Ålund† and Ulf Thibblin‡

Abstract

A parallel, 3-D Finite Difference Time Domain (FDTD) code has been developed and numerical simulations of the electromagnetic interaction of a plane wave with an aircraft model were performed on a cluster of workstations in an industrial environment.

For a large size problem run on a network of four HP 9000-715/50 interconnected with Ethernet a speedup factor of 2.8, as compared to the one processor case, was observed.

1 Introduction

Workstations in industry are often idle at night and over weekends. One way of utilizing this source of computing power and reduce the load on the supercomputer at the computer center is to run the computer programs in parallel on a cluster of workstations connected by a network. There is no extra cost for new hardware. Only the parallelization of the software is necessary.

In a joint project between the Swedish Institute of Applied Mathematics, ABB Corporate Research, Saab Military Aircraft, and Volvo Aero Corporation the possibility of running industrial production codes in parallel on workstation clusters is studied.

This paper focus on Computational Electromagnetics and describes one of the techniques used to solve Maxwell's partial differential equations, the well-known Finite Difference Time Domain (FDTD) method, which explicitly follows the evolution of an incident electromagnetic wave in time as it impinges upon the scatterers. This method is particularly well suited to transient analysis for pulsed incident radiation.

The serial version of the FDTD code used at Saab Military Aircraft (Saab MA) has been rewritten using a message passing programming model with PVM as message passing system, and has been run on workstation clusters at Saab MA.

2 The FDTD Code

The Finite Difference Time Domain (FDTD) method for electromagnetic computations, introduced by Yee in 1965 [4], has been extensively discussed in the literature [5], and is presented briefly below.

FDTD is based on a discretization of Maxwell's curl equations for linear media,

$$\begin{aligned}\mu\frac{\partial \mathbf{H}}{\partial t} + \nabla \times \mathbf{E} &= \mathbf{0}\\ \epsilon\frac{\partial \mathbf{E}}{\partial t} - \nabla \times \mathbf{H} &= -\mathbf{J}\end{aligned}$$

where $\mathbf{E}$ is the electric field, $\mathbf{H}$ the magnetic field, $\mathbf{J}$ the source current density, μ the magnetic permeability and ϵ the electric permittivity. The spatial and time derivatives are approximated with central differences. The form of the resulting difference equations depend on the actual coordinate system where the Cartesian one is the most commonly used. The Cartesian grid is based on a unit cell. Different components of the electric field are given in a staggered fashion, and so are the components of the magnetic field. Also, the magnetic field points are staggered between the electric field points that are located at the center of the rectangular cell faces (or vice versa). Time is discretized with equidistant steps and the $\mathbf{E}$ and $\mathbf{H}$ fields are separated in time with one half of the time step. The algorithm uses a leapfrog scheme in time and space to follow the evolution of the electric and magnetic fields in the computational space. The resulting finite difference equations are explicit and result in a fast and, in its basic form, robust algorithm for electromagnetic field computations. To calculate a given component of the $\mathbf{H}$-field, only four neighboring $\mathbf{E}$-field values are needed (and vice versa for the $\mathbf{E}$-fields). This means that in a parallel version of the code, where the computational space is divided into blocks and the electromagnetic fields in each block are computed on different processors, only the fields at the block boundaries have to be communicated to other blocks.

*Research supported by NUTEK, Swedish National Board for Industrial and Technical Development, under grant 5321-93-04384, project no 9304384.

†Swedish Institute of Applied Mathematics, Chalmers Teknikpark, S-412 88 Göteborg, Sweden

‡Saab Military Aircraft, S-582 88 Linköping, Sweden

3 Parallel Implementation

Many codes used in the aerospace industry today to solve Maxwell's equations are based on the FDTD method on regular Cartesian grids. The EMA3D [3] code used at Saab MA is of this type and has formed the basis for the implementation of the parallel FDTD code.

The regularity of the computational grid makes the decomposition onto a distributed-memory machine relatively straight forward. Each processor handles the calculations in one or more blocks of the grid. When cells on a block boundary need data from their neighbors in adjacent blocks in order to update the electric and magnetic fields in each iteration, these data must be transferred between the processors over a network. In each iteration many floating point operations are needed to update the solution in the cells inside the block. Only data on the boundaries of the block are sent via the network. The time needed for the computations normally dominate over the time for communication in the total elapsed time to solve a problem.

A message passing system is needed for the data transfer between the processors. A widely used such system is PVM (Parallel Virtual Machine) [1, 2]. It is available on many workstations and parallel computers. PVM 3.2.6 is used for the implementation of the parallel code. The code is written in Fortran 77 using single precision arithmetic.

The parallel code is organized in a host – slave fashion. One host process is responsible for file I/O and the supervision of the slave processes. The host reads the number of slave processes to be started and a partition of the problem, i.e. a list of processors to start slave processes on and the blocks to be handled by each slave process. The host process starts each slave process and sends configuration and common data to each slave. The host is responsible for the control flow of the slaves by issuing commands to be interpreted by each slave.

Each slave process receives the configuration and common data for its block, the corresponding subset of the initial solution and is then responsible for updating the solution in its block. Where two blocks share an interface boundary their boundary data must be transferred between the corresponding slave processes by explicit message passing. This is done twice in each iteration in the time stepping, once for the electric field and once for the magnetic field.

The handling of the boundary conditions is implemented using *ghost*-variables. A *ghost*-variable contains a copy of the field variables from a neighboring block or values calculated from the absorbing boundary condition on an outer boundary. By the use of *ghost*-variables, the loops updating the inner field variables do not have any conditional statements at the boundaries and the communication is localized. After the simulation, each slave process sends its part of the solution back to the host processor and terminates.

4 Numerical Experiments

In this section we report on the experiences with the parallel FDTD code, computing a solution to Maxwell's equations for an aircraft model with a plane wave exitation.

The generic aircraft model is used to evaluate the accuracy of different computer codes. Although simple, it represents many important features of an aircraft in terms of its electromagnetic behavior. The aircraft model is shown in Figure 1 below.

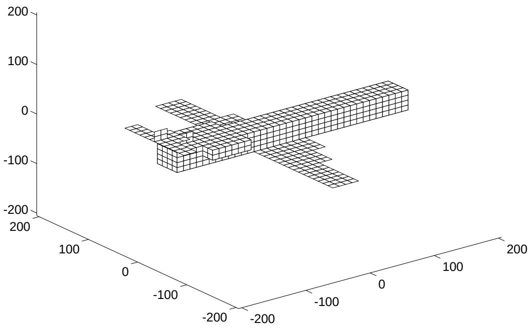

Figure 1: The generic aircraft model

The experiments were carried out on two workstation clusters at Saab MA. One cluster has Silicon Graphics Indigo 2 with 150 MHz MIPS R4400 processors and the other consists of HP9000-715/50 workstations with 50 MHz PA-RISC 7100 processors. The workstations are interconnected with a 10Mbit/s Ethernet. For comparison, we also provide results from runs on one Cray Y-MP processor. Table 1 on the next page shows the times in seconds/iteration for problem sizes ranging from 64K cells to 8M cells. Due to memory limitations the largest problem sizes were only tested on the SGI cluster.

There are several ways to measure performance of parallel applications. One common performance number is the speedup, defined as $S_p = T(p)/T(1)$, where $T(p)$ is the time needed to solve the problem on p processors. The speedup is shown in Figures 2 and 3 for the SGI and HP test cases, respectively.

To check scalability, it is also interesting to study the communication costs when increasing the total problem size, by increasing the number of processors with fixed problem size per processor. This is shown in Figures 4 and 5.

The SGI and HP workstations used in the experiments are approximately 15 times slower than one Cray Y-MP processor. It is also interesting to note that the time on one processor does not increase linearly with the problem size. This is due to the limited cache size on the workstations.

Problem-size (cells)	Number of procs	Machines		
		SGI	HP715	Cray Y-MP
64K	1	0.25	0.28	
128K	1	0.48	0.53	
	2	0.34	0.38	
256K	1	0.99	1.05	
	2	0.62	0.67	
	4	0.56	0.48	
512K	1	1.66	1.92	0.13
	2	1.15	1.21	
	4	1.01	0.82	
1M	1	3.26	3.82	0.24
	2	1.84	2.19	
	4	1.71	1.33	
2M	1	6.78	7.64	0.45
	2	3.74	4.18	
	4	2.65	2.36	
4M	2	7.12		0.88
	4	5.03	4.47	
8M	4	9.63		

Table 1: Elapsed time in seconds/iteration

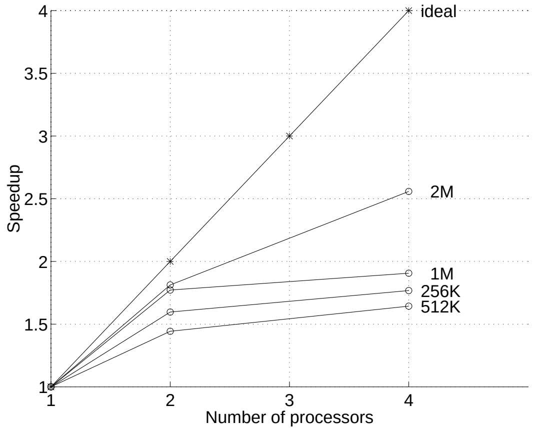

Figure 2: Speedup with fix total problem size on SGI

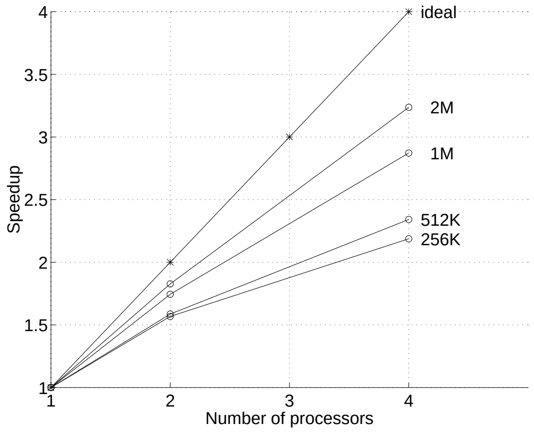

Figure 3: Speedup with fix total problem size on HP

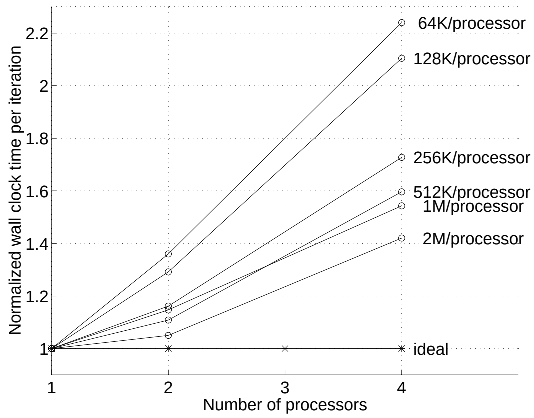

Figure 4: Normalized time with fix problem size per processor on SGI

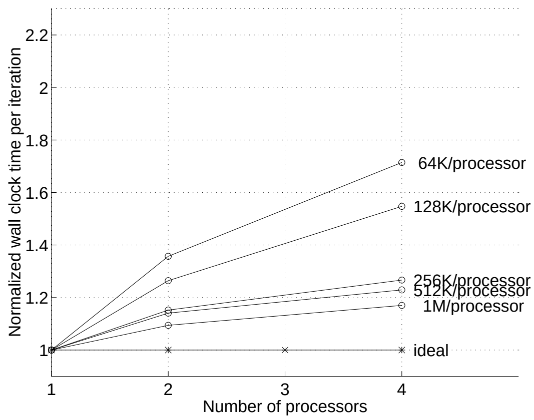

Figure 5: Normalized time with fix problem size per processor on HP

5 Analysis

In this section we study the computational work in each step of the main loop, analyze the communication overhead in the parallel FDTD code and derive a simple model for the balance between work and communication.

Let n_x, n_y and n_z be the dimensions of the computational grid in the x-, y- and z-directions, b_x, b_y and b_z the number of blocks in each direction and $p = b_x b_y b_z$ the number of processors used. The time needed to update the solution one timestep is estimated as the sum of the time used in each step in the main computational loop and consists of the following components:

t_{RBC}	Radiating boundary conditions
t_{OBC}	Scattering object boundary conditions
t_{SRC}	Source terms
t_{EH}	Update E- and H-fields
t_{COM}	Communicate interface variables

Electric and magnetic fields are updated in each cell of the computational volume so the work is proportional to $n_x n_y n_z$.

The radiating boundary conditions are updated on the outer surfaces of the computational volume. The work is therefore proportional to $n_x n_y + n_y n_z + n_z n_x$.

Source terms are updated on surfaces, which in our case are placed close to the outer boundary and we approximate the work to be proportional to $n_x n_y + n_y n_z + n_z n_x$.

The update of the scattering object boundary conditions involve no floating point operations and the work is neglected in our model.

Floating point performance unfortunately depends on n_x, n_y and n_z. For RISC architecture processors, a combination of n_x, n_y and n_z as well as the processor's cache size and organization determine the speed.

The total time for one timestep on one processor, $T(1)$, is estimated as

$$T(1) = K_3 n_x n_y n_z + 2K_2(n_x n_y + n_y n_z + n_z n_x)$$

where K_3 is the time to update an interior cell and K_2 the time to update a boundary cell.

The generalization to the multi-processor case reads

$$T(p) = T(1)/p + t_{COM}$$

where the communication time for the four 32-bit floating point numbers communicated between each cell on an interface boundary is estimated as $t_{COM} = 16\beta((b_x-1)n_y n_z + (b_y-1)n_x n_z + (b_z-1)n_x n_y)$ where β is the average time to communicate one byte of data between two processors. To further simplify the model, the work on surfaces is neglected, $n_x = n_y = n_z = n$ and $b_x = b_y = b_z = b$. Then, $p = b^3$ and the equation reads

$$T(p) = \frac{K_3 n^3}{p} + 48\beta(p^{1/3} - 1)n^2.$$

The speedup is

$$S(p) = T(1)/T(p) = \frac{p}{1 + \frac{48\beta p(p^{1/3}-1)}{k_3 n}}.$$

In Figure 6, speedup estimates according to this model are shown for two types of communication networks. For Ethernet, $\beta = 10^{-6}$ is used, and for the fiber optics FDDI network, $\beta = 10^{-7}$ is used. $K_3 = 3 \times 10^{-6}$ in both cases. With larger size problems it would be possible to achieve good speedup figures with up to 20 workstations using FDDI.

6 Conclusions

An industrial production code solving Maxwell's equations with the Finite Difference Time Domain (FDTD) method has been parallelized using PVM to run on a number of workstations in a cluster. The total work for the conversion including tests was a couple of man-months.

On a cluster of four HP 9000-715/50 workstations on Ethernet we reach close to 20% of the performance of one Cray Y-MP processor and compared to one workstation we get a speedup of 2.8 for a problem with 1 million grid points.

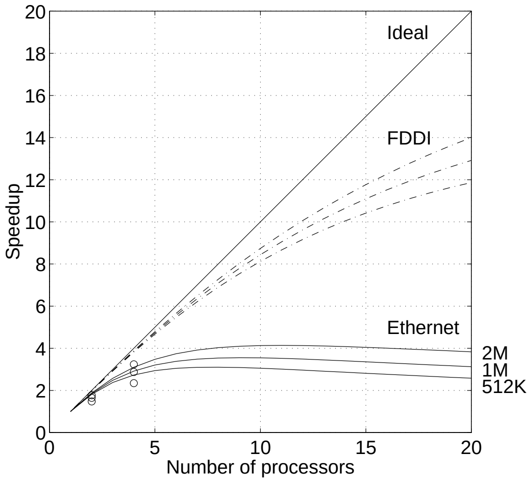

Figure 6. Estimated speedup for the FDTD code

References

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, "PVM 3 User's Guide and Reference Manual", ORNL/TM-11826 (1993), Oak Ridge National Laboratory, Oak Ridge, TN 37831.

[2] V. S. Sunderam, G. A. Geist, J. Dongarra,R. Manchek, "The PVM concurrent computing system: Evolution, experiences, and trends", *Journal of Parallel Computing*, 20 (1994), 531-546.

[3] EMA Electromagnetic Applications, Inc., Denver.

[4] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media.", IEEE Trans. on Ant. Prop., AP-17, pp 302-7, May 1966.

[5] See for example:
A. Taflove and K. R. Umashankar, "The Finite-Difference Time-Domain Method for Numerical Modeling of Electromagnetic Wave Interactions With Arbitrary Structures.", in PIER2, Finite Element and Finite Difference Methods in Electromagnetic Scattering, ed. M. A. Morgan, Elsevier Science Publishing Co., New York, 1990, pp 287-373.