Experience with ADI-FDTD Techniques on the Cray MTA Supercomputer

Harry Jordan[1], Shahid Bokhari[2], Shawn Staker[3],
Jon Sauer[1], Mona ElHelbawy[1], and Melinda Piket-May[1]

1. University of Colorado at Boulder, Dept. of Electrical and Computer Engineering
2. University of Engineering and Technology, Lahore, Pakistan
3. MIT Lincoln Laboratory, Lexington, Massachusetts

ABSTRACT

Finite difference, time domain (FDTD) simulations are important to the design cycle for optical communications devices. High spatial resolution is essential, and the Courant condition limits the time step, making this problem require the level of high-performance system usually only available at a remote center. Model definition and result visualization can be done locally.

Recent application of the alternating direction implicit (ADI) method to FDTD removes the Courant condition, promising larger time steps for meaningful turnaround in simulations. At each time step, tridiagonal equations are solved over single dimensions of a 3D problem, but all three dimensions are involved in each time step. Thus, for a distributed memory multiprocessor, no partition of the data prevents tridiagonals from crossing processors without remapping every time step. Likewise, for cache based or vector computers, there is a stride of $N \times N$ for some tridiagonals on a $N \times N \times N$ grid. There is plenty of parallelism because $N \times N$ tridiagonals can be solved simultaneously. This makes the problem well suited to a machine like the Cray multithreaded architecture (MTA) that has a large, flat memory and uses parallelism to hide memory latency.

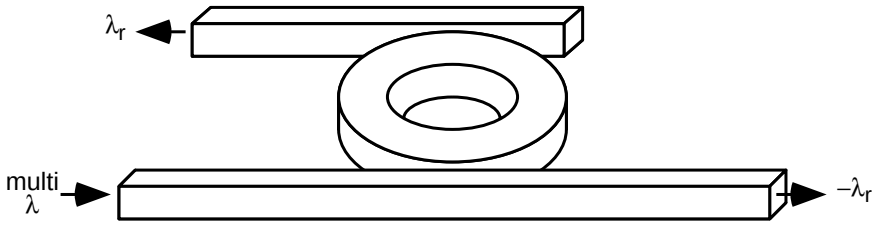
A Cray MTA implementation of the ADI-FDTD code executes tridiagonal solvers in parallel on multiple threads and sucessfully hides memory latency, achieving just over one floating point operation per clock cycle per processor for a grid of size 200×200×200 on an 8 processor system at the San Diego Supercomputer Center. The 8 processor speed is 2.06 Gflop and the efficiency is 98%. Comparing one MTA processor, with a 250MHz clock to a 500 MHz Alpha system, the MTA is three times as fast for a 50×50×50 grid size. A vectorized version of the code run on one Cray T90 processor is three times faster than one MTA processor for a 100×100×100 grid size.

Keywords: optics, simulation, ADI-FDTD, Cray MTA, performance

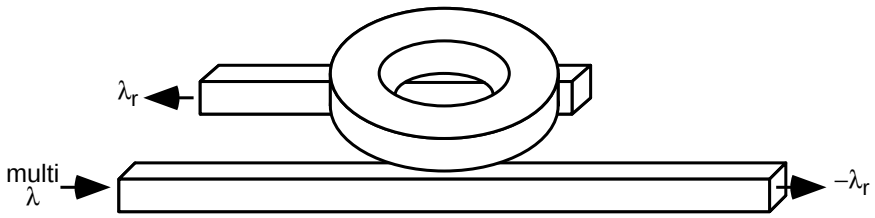
1. THE OPTICAL RESONATOR MODELING PROBLEM

With the tremendous growth in optical networking, the ability to accurately model optical devices in order to do cost/performance trade-offs is becoming increasingly important to competitive success. The finite difference, time domain (FDTD) method for electromagnetics analysis by direct solution of Maxwell's equations offers detailed insight into the behavior of optical devices with complex geometries, but at a high computational cost. Coupled waveguides, resonators, grating structures, and photonic crystals can all potentially be analyzed in depth by FDTD methods [1]. The most success so far has been with appropriate 2D analysis of primarily planar structures. The use of 3D analysis, required by all but the simplest devices, moves the computational requirements into the realm of the highest performance computers, available only remotely to all but a few sites. An ideal design cycle for a device requiring 3D analysis would consist of geometric layout and parameter specification on a local workstation, followed by FDTD simulation on a remote, top performance system, with result visualization and analysis back on the workstation.

To establish the parameter ranges in a concrete case that can be addressed with near-term algorithms and computer resources, we look at the design of a ring resonator coupled to waveguides and used for stripping off one wavelength in a wavelength division multiplexed (WDM) signal. The planar version of this device, sketched in Fig. 1(a), has been simulated by Hagness [2] using 2D FDTD methods. The efficiency of the device is a very sensitive function of the gap between a waveguide and the ring. This gap is hard to control with planar fabrication. If the waveguides and ring are fabricated on different layers, however, as sketched in Fig. 1(b), the gap becomes a function of vertical layer spacing, which can be controlled more precisely. This



(a) Planar configuration of ring resonator and waveguides



(b) Coupling of waveguides to resonator in the 3rd dimension

FIGURE 1. Two and three dimensional ring resonator configurations

makes the device a 3D structure, and while it promises lower cost per performance through simpler fabrication and enhanced coupling, it also increases the computational cost of simulation significantly. At the usual optical fiber center frequency of 200 THz, or 1.5 μm wavelength, the standard Yee algorithm [3] for FDTD analysis requires about 20 samples per wavelength. This demands a 75 nm grid cell in free space, or a 22 nm grid cell in a material of index 3.4, used in the device analyzed by Hagness. This is more than sufficient to resolve the geometric structure of a 5-10 μm diameter ring resonator coupled to 0.5 μm waveguides by gaps of about 0.2 μm. With the fineness of the grid determined by the optical frequency, the device geometry determines the extent of the grid. If the length and width of a plan view were 22 μm by 11 μm and the thickness

modeled were 2.2 μm, the FDTD grid would be $1000\times500\times100 = 5\times10^7$ points. The time dimension, however, is the most demanding. The Yee algorithm becomes unstable for a time step larger than that given by the Courant-Levy condition, $\Delta t = \Delta x/c$, for this problem 0.25×10^{-15}s. Simulation of a 10 ps signal passing through the device would require 40,000 time steps. Thus, the time dimension significantly dominates any space dimension in its computational extent.

A recent development in FDTD algorithms, the alternating direction implicit (ADI) method, promises to lessen the computational requirements of the time domain in FDTD simulation by removing the Courant condition on the time step [4]. The ADI form of FDTD is unconditionally stable for any time step, with solution accuracy decreasing for large time steps. Its use is indicated in so-called over resolved problems, where the frequencies of interest demand a finer space grid than necessary to resolve the problem geometry. The method propagates information over more than one grid cell per time step by using an implicit tridiagonal equation set in one of the three space dimensions. The implicit coupling of grid points occurs along each space dimension twice in the six substeps making up a single time step in the ADI-FDTD method. Both amplitude and phase accuracy must be considered in suiting the time step to the solution requirements [5].

2. COMPUTATIONAL AND PARALLEL ASPECTS OF ADI-FDTD

The standard Yee FDTD algorithm is completely parallel over all three space dimensions within a time step. First, the three E field components, E_x, E_y, and E_z at time step n+1/2 are computed from the H field components at time step n and old E components at n–1/2. Then, H_x, H_y, and H_z at time n+1 are computed from E at time n+1/2 and H at time n. Independent updates at each point of the space grid yield a high degree of parallelism that can be exploited easily by vector processing or shared or distributed memory multiprocessors. Multiple vector processors can use one space dimension for vectorization and a second for distributing work over multiple processors. In distributed memory, domain decomposition can be applied, and communication overhead is proportional to the surface to volume ratio of the subdomains.

In the ADI version of FDTD, coupled equations give up independence in one of the three space dimensions for each of the six substeps making up a time step. Equations form tridiagonal systems over x (y or z) in two out of the six substeps. Hence, no one space dimension maintains independence, and thus parallelism, over a full time step. At any substep, however, there is ample parallelism for current system sizes over the two space dimensions maintaining independence. Although there is sufficient parallelism, no fixed mapping of space points to processors will keep coupled points in the same processor over a full time step.

Across the common types of parallel systems, less effort is required to implement the ADI-FDTD method on a shared memory multiprocessor than on a vector processor or a distributed memory multiprocessor. Since no fixed mapping will maintain coupled points in one processor over a time step, communication proportional to the full space grid volume would be required at each time step on a distrbuted memory multiprocessor. On a vector processor, the simultaneous solution of multiple tridiagonal systems must be vectorized. This is quite tractable, as will be seen in the description of our experiments on the Cray T90, but it is slightly beyond the automatic capabilities of current vectorizing compilers.

The character of one of the six substeps in a time step is shown by the code in Fig. 2. The substep updates the E_x^h field component at the halfway point of the time step with tridiagonal systems coupled in the y direction. A subsequent substep will compute E_x at the full time step with coupling in the z direction. Global declarations of the field and material constant arrays are hidden in the include file, but it can be seen that three local 1D arrays are introduced to solve the tridiagonal system. To parallelize this code for a shared memory multiprocessor, the compiler needs only deduce, or be told, that the subroutine invocations are independent for different (`i`, `k`) pairs. It can then parallelize either or both of the loops over `i` and `k` across multiple processes.

3. PERFORMANCE OF THE ADI-FDTD METHOD ON THE MTA

The Cray multithreaded architecture (MTA) supports easy porting of software for serial machines. Problems of data partitioning, mapping, and re-ordering that are important in supercomputers made up of collections of commodity microprocessors do not arise in the MTA. On the eight processor MTA system at the San Diego Supercomputer Center, a serial ADI-FDTD code with minimal parallel directives was able to achieve more than one floating point operation per clock cycle on a large 200×200×200 space grid. Little change in this per processor speed is expected when the machine is expanded, as planned, to 16 or even 32 processors. The parallel code runs nearly three times faster on one 250 MHz processor of the MTA than the orig-

```
c       ..... exh iteration .....

      do 70 i = 1, ie
       do 60 k = 2, ke
        call trsexh(i, k)
60     continue
70    continue
```

```
      subroutine trsexh(i, k)
      include 'trincl'
      real tmp,receps
      real a(je),b(je),d(je)

      do j = 2, je
       receps=1./eps(k,i,j)
       a(j)=-tt4myy*receps
       b(j)=1. + tt2myy*receps
       d(j)=ex(k,i,j)+receps*(t2y*(hz(k,i,j)-hz(k,i,j-1)) -
     1  t2z*(hy(k,i,j)-hy(k-1,i,j)) - tt4mxy*
     2  (ey(k,i+1,j)-ey(k,i,j)-ey(k,i+1,j-1)+ey(k,i,j-1)))
      enddo

      do j = 3, je
       tmp=a(j)/b(j-1)
       b(j)=b(j) - tmp*a(j-1)
       d(j)=d(j) - tmp*d(j-1)
      enddo

      exh(k,i,je) = d(je)/b(je)
      do j = je-1, 2, -1
       exh(k,i,j)=(d(j) - a(j)*exh(k,i,j+1))/b(j)
      enddo
      return
      end
```

FIGURE 2. Serial code for one of six substeps in an ADI-FDTD time step

inal serial code runs on a 500MHz Alpha system for a 50×50×50 space grid. At the cost of user level code restructuring for vectorization, a single processor of the Cray T90 vector supercomputer is about three times faster than one MTA processor on a 100×100×100 grid problem.

3.1 Specifics of the system and program measured

The Cray MTA multithreaded architecture [6] is a shared memory multiprocessor system, in which each processing unit is a pipelined MIMD computer in the same sense that a Cray T90 processor is a pipelined SIMD computer. That is, instructions from multiple independent instruction streams, or threads, occupy the execution pipeline simultaneously. The architecture uses some of the parallelism among threads to keep the execution pipeline retiring instructions at full bandwidth in spite of memory latency delaying some fraction of the threads. Further parallelism can be used for threads running on additional processing units. With up to 128 threads per processing unit and the additional use of single stream lookahead techniques within each thread, several hundred cycles of memory latency can be tolerated while still retiring one three-operation instruction per clock cycle from the execution pipeline. The memory is flat and uniformly accessible with no data caching. Although the clock rate of the Cray MTA is currently 250 MHz, as compared to 500 MHz to 1 GHz for contemporary commodity microprocessors, the MTA is capable of sustaining more than one floating point operation per clock tick per processor for large data sets. The Cray

MTA system at the San Diego Supercomputer Center, on which we made our measurements, had 8 processors and 8 Gigabytes of memory.

The performance measurements were made on a simplified ADI-FDTD code developed by Shawn Staker under the supervision of Prof. Melinda Piket-May at the University of Colorado, Boulder. A practical code for the ring resonator problem described previously would include complexity in the treatment of the grid boundaries that would influence performance only proportional to the ratio of grid surface to volume. The specific electromagnetics problem solved by the measured program is that of a rectangular metal cavity, half filled with dielectric of constant $\varepsilon_r = 64$. The size of the simplified test code is about 500 lines of Fortran. The Fortran arrays used in the computation are statically allocated, so dynamic memory allocation and pointers do not hinder the compiler in recognizing parallelism, as is true of programs in some common benchmark sets [7]. Thirteen real arrays of the size of the space grid are needed for field components and the dielectric constant. Auxiliary one dimensional arrays for the tridiagonal solvers are only allocated for active threads.

The serial code of Fig. 2 was run relatively unchanged on the MTA. The only compiler directives that could convey useful information are shown in Fig. 3. The two `c$tera assert parallel` directives tell the compiler that executions of the

```
c$tera assert parallel
      do 70 i = 1, ie
c$tera assert parallel
       do 60 k = 2, ke
        call trsexh(i, k)
        .  .  .
```

```
c$tera parallel off
c$tera inline
      subroutine trsexh(i,k)
        .  .  .
```

FIGURE 3. Directives enabling parallel compilation on the MTA

tridiagonal subroutine `trsexh(i, k)` are independent for distinct (`i`, `k`) pairs, the `c$tera parallel off` directive tells the compiler to optimize the subroutine for sequential execution, and the `c$tera inline` directive tells the compiler to in-line the subroutine. Actually, with the subroutine in-lined, the compiler deduces the independence of the iterations of the `i` and `k` loops without the directives. After parallelization, multiple threads execute the tridiagonal solver simultaneously, and each has three temporary arrays, `a(je)`, `b(je)`, and `d(je)`, that constitute space overhead of the parallelization. This space overhead is proportional to the grid extent, `je`, in the *y* direction times the number of threads used.

3.2 Results of the performance measurements

Both the number of MTA processors and the problem size were varied in the performance measurements. Cuboids of sizes 50^3, 100^3, 150^3, and 200^3 were measured on one through eight processors. No program modifications whatever are required to go from one to multiple processors. The same compiled binary is run with different processor limits. The time per single FDTD time step (time per cycle) is plotted versus the number of processors in Fig. 4. The continuous curves represent perfect speedup, or $T(1)/P$, where $T(1)$ is the time with one processor, and P is the number of processors. Fig. 5 shows the same data on a semi-log plot to make visible the small deviations from perfect speedup for small problem sizes and large numbers of processors. The speedup is nearly perfect for problem sizes larger than 100^3.

The floating point operation rate was obtained by reading the internal floating point operation counters of the MTA and is plotted for the same ranges of parameters as the time step data in Fig. 6. One floating point operation per clock tick would give a rate of 250 Mflops per processor. It is seen that this rate is somewhat exceeded on the 200^3 grid size. The efficiency computed for 8 processors on the 200^3 grid is 98%. The time and rate results are quite smooth. There is no evidence of the “magic numbers” that appear in performance measurements on machines with cache size or vector stride limits.

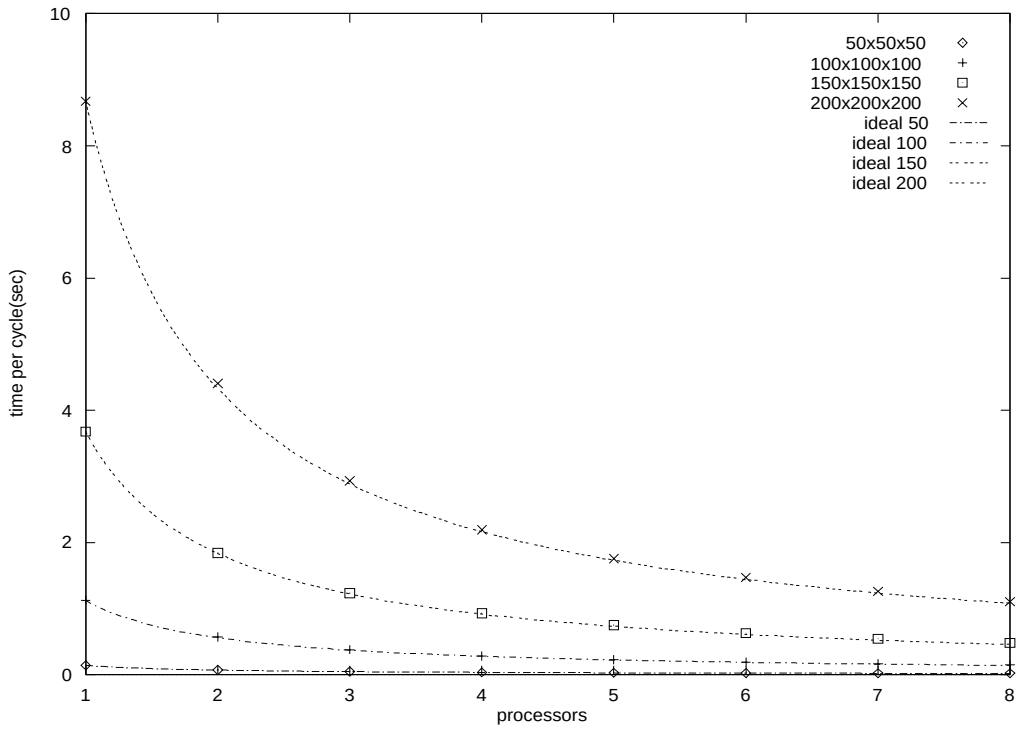


FIGURE 4. CPU seconds per FDTD time step on the MTA system

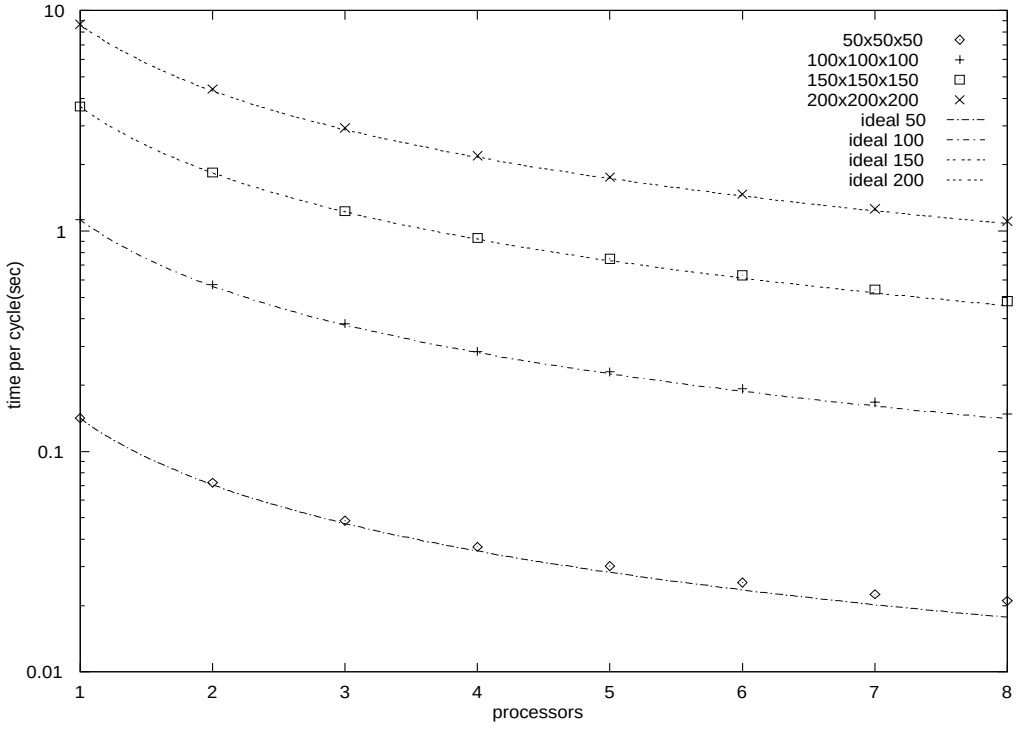


FIGURE 5. Semilog plot of Fig. 4 data showing deviations from ideal

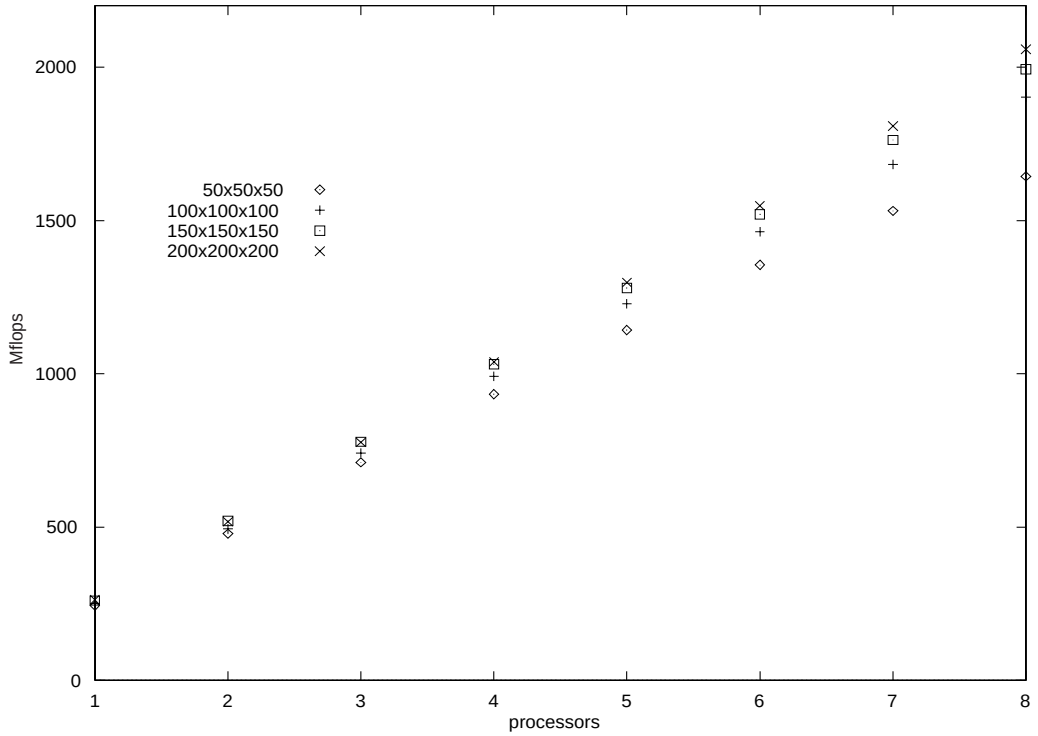


FIGURE 6. Near linear speedup of the floating point rate on the MTA

4. PERFORMANCE OF VECTORIZED ADI-FDTD ON THE CRAY T90

Some preliminary experiments have been done to compare the MTA performance with that of the Cray T90 vector computer [8]. To date, only one processor comparisons have been done, so the primary issue is vectorization of the ADI-FDTD algorithm. When the code of Fig. 2 was run unchanged on the T90, the compiler could not vectorize the tridiagonal solvers and no in-lining or compiler directives were of use. Thus, a single T90 CPU was able to achieve only about 100 Mflops on a 50×50×50 grid size. The key to vectorization is to move one of the two parallel loops into the tridiagonal solver subroutine, introduce two dimensional temporary arrays for the diagonals to be processed in parallel and vectorize over multiple tridiagonal solvers. The substep code shown in Fig. 2 is thus replaced by that shown in Fig. 7. No further restructuring of the code is necessary for the T90 to be able to successfully vectorize it. The code is vectorized over whichever of the nested loops has independent instances. When there is a choice, as there in in the first two loop nest of the subroutine, the compiler vectorizes the loop that yields the best access stride to vectors from the two and three dimensional arrays. In the substep shown in Fig. 7, the vectorization is over the k index in all cases, since k is the first index for both 2D and 3D arrays and thus yields stride one access to vectors with the Fortran array storage layout. This version of the code achieves 650 Mflops on a 50×50×50 problem and 827 Mflops on a 100×100×100 problem.

A detailed analysis of the performance of the six substeps in the 100^3 sized problem shows that the achieved rates vary from 613 to 979 Mflops. This is a result of the access stride to vectors, and can be improved by further tuning of the code for the T90. Avoiding powers of two in matrix dimensions by using slightly more space to make dimensions odd is one way to solve the stride problem. Also, although the 3D field array dimensions cannot be reordered to ensure vector access over the first dimension in all six steps, the 2D auxiliary arrays in the tridiagonal solvers can be transposed, or not, to give stride one access in each of the six substep cases. This has not yet been done, but the result would clearly be an overall rate between 827 and 979 Mflops. We can thus conclude that one T90 processor is 3-4 times as fast as one MTA processor provided the ADI-FDTD code is restructured for vectorization. The space cost of restructuring is three N×N temporary arrays for an N×N×N grid size.

Remaining to be done is the extension of the T90 code to multiple processors. To do this, multitasking must be employed in

```
c       ..... exh iteration .....
      do 70 i = 1, ie
        call texhkj(i)
70    continue
```

```
      subroutine texhkj(i)
       include 'trincl.inc'
       real tmp, receps
       real a(ke,je), b(ke,je), d(ke,je)

       do k = 2, ke
        do j = 2, je
          receps = 1./eps(k,i,j)
          a(k,j) = -tt4myy * receps
          b(k,j) = 1. + tt2myy * receps
       d(k,j)=ex(k,i,j)+receps*(t2y*(hz(k,i,j)-hz(k,i,j-1)) -
     1  t2z*(hy(k,i,j)-hy(k-1,i,j)) - tt4mxy*
     2  (ey(k,i+1,j)-ey(k,i,j)-ey(k,i+1,j-1)+ey(k,i,j-1)))
        enddo
       enddo

       do k=2, ke
        do j=3, je
          tmp = a(k,j)/b(k,j-1)
          b(k,j)= b(k,j) - tmp*a(k,j-1)
          d(k,j)= d(k,j) - tmp*d(k,j-1)
        enddo
       enddo

       do k=2, ke
        exh(k,i,je) = d(k,je)/b(k,je)
       enddo

       do k= 2,ke
        do j=je-1, 2, -1
         exh(k,i,j) = (d(k,j)-a(k,j)*exh(k,i,j+1))/b(k,j)
        enddo
       enddo
       return
       end
```

FIGURE 7. Vectorizable version of a typical ADI-FDTD substep

addition to vectorization. Since there is still one loop with independent instances left outside the vectorized tridiagonal solvers, allocating independent work to multiple processors should not be a problem. Overhead in using multiple vector processors is typically higher than what we observed in the MTA as the number of processors increased, so the experiment is interesting.

5. CONCLUSIONS

The ADI-FDTD method promises to be an important tool in modeling optical devices for cost/performance tradeoffs. The method runs well and speeds up nearly linearly with the number of processors on the Cray MTA multithreaded architecture. Serial code can be parallelized by the MTA compiler with no restructuring by the user. The MTA averages more than one floating point operation per cycle of the 250 MHz clock over the computation, while an Alpha processor with twice the clock rate

is three times slower. Code restructuring and the introduction of temporary arrays is required before the T90 compiler can vectorize the code effectively, but when this is done, a single T90 processor is 3-4 times faster than one MTA processor. This code is not one that makes maximal use of the unique features of the multithreaded architecture, as would a code that had a fairly random access pattern in a large shared memory. It does favor a shared memory system because there is no static partitioning of the data for distributed memory that applies over a full time step. The problem is vectorizable, but at the cost of some space overhead and user restructuring of the code. The measurements demonstrate that the MTA delivers a very large fraction of its peak performance on this problem with minimal programming effort.

ACKNOWLEDGEMENTS

This research was supported in part by the Air Force Office of Scientific Research under Grant No. F49620-00-1-0386 and the Colorado Photonics and Optoelectronics program.

REFERENCES

[1] A. Taflove, and S. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method, 2nd ed.*, Artech House, Boston, MA, Chap. 16 (2000).

[2] S. Hagness, D. Rafizadeh, S. T. Ho, and A. Taflove, "FDTD microcavity simulations: design and experimental realizations of waveguide-coupled single-mode ring and whispering-gallery-mode disk resonators," *J. Lightwave Technology*, Vol. 15, pp. 2154-65 (1997).

[3] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. on Antennas and Propagation*, Vol. 14, No. 3, pp. 302-307 (May 1966).

[4] T. Namiki, "A new FDTD algorithm based on alternating-direction implicit method." *IEEE Trans. on Microwave Theory and Techniques*, Vol. 47, pp. 2003-07 (Oct. 1999).

[5] F. Zheng, and Z. Chen, "Numerical dispersion analysis of the unconditionally stable 3D ADI-FDTD method," *IEEE Trans. on Microwave Theory and Techniques*, Vol. 49, No. 5, pp. 1006-10 (May 2001).

[6] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The Tera computer system," Proc. 1990 International Conf. on Supercomputing, (June 1990).

[7] S. Brunett, J. Thornley, and M. Ellenbecker, "An initial evaluation of the Tera multithreaded architecture and programming system using the C31 parallel benchmark suite," *Proc. Supercomputing 98*, Orlando, FL (1998).

[8] Cray Inc., *Guide to Parallel Vector Applications*, 004-2182-003. Available online at http//dynaweb.sdsc.edu:80/dynaweb/t90_c90_y90_cray_fp