# 2

# PROGRESS IN CRAY-BASED ALGORITHMS FOR COMPUTATIONAL ELECTROMAGNETICS

*J. P. Brooks, K. K. Ghosh, E. Harrigan,*
*D. S. Katz, and A. Taflove*

## 2.1 Background and Summary

In the recent past, many new computer architectures have been proposed as being cost-effective for simulating key physical phenomena and engineering problems on indefinitely expanding scales. Usually, these architectures concurrently apply large numbers (greater than 100) of either off-the-shelf or custom microprocessors and associated memory units, providing the necessary processor-to-processor communication. Special programming languages or compilers may be needed to efficiently utilize the moderate to massive processor concurrency implied.

At Cray Research, the most recent series of supercomputers is based upon the Cray Y-MP/8 architecture running the UNICOS operating system. The general design philosophy behind the Y-MP is to deliver balanced supercomputer performance by combining a set of fast vector functional units, a very high-speed central memory bandwidth, an extremely efficient I/O subsystem, and software that allows use of these resources from high-level languages such as standard FORTRAN. Specifically, the Y-MP/8 has:

- Up to eight processors clocked at 6 nsec, operating concurrently if needed under the autotasking compiler;
- A common central memory of up to 128-million words (128 Mwords), with a memory bandwidth of 4 billion bytes per second (4 Gbytes/sec) per CPU;
- An optional solid-state device (SSD) providing up to 512 Mwords of secondary memory storage, achieving peak transfer rates to main memory of 2.5 Gbytes/sec.

Standard FORTRAN code can be ported to the Y-MP usually within hours or a few days, thereby minimizing program development time.

Given the availability of alternative computer architectures, it is reasonable for workers engaged in supercomputer simulations at the cutting edge of science and technology to have the following questions concerning Cray-based algorithms regarding possibilities for expanding simulation sizes well beyond current levels:

> "Can we efficiently exploit the multiprocessing potential of the Y-MP/8 architecture....that is, will the concurrent use of eight Y-MP processors provide simulation speedups of almost 8:1?"

"Can we go out of core on the Y-MP/8 for the largest problems, especially onto magnetic disk(s), without suffering from being I/O bound (running time dominated by slow reads or writes to disk)?"

Cray Research has been involved in addressing these problems in a number of contexts. This paper reports recent progress in this area in an important engineering application, computational electromagnetics (CEM). Since 1986, Cray Research has been involved in porting and optimizing CEM codes for its Cray-2, X-MP, and Y-MP computer series at the request of major U.S. aerospace firms. Of particular interest here is the potential for using CEM codes to model the largest possible three-dimensional (3-D) targets for electromagnetic wave scattering and radar cross-section (RCS). As described in the next three sections, Cray has been working with three principal algorithms of current interest in CEM:

*a. Method of Moments*

The frequency-domain method of moments (MM) has been the main tool in formulating detailed models of EM wave radiation, scattering and RCS. In this area, two important groups of predictive codes are widely used for RCS analysis of arbitrary shaped 3-D conducting structures: (1) Rao-Wilton-Glisson triangular surface patching [1]; and (2) Newman quadrilateral surface patching [2]. For either group of codes, Cray Research analysts determined that the primary computational problem involves the solution of ultra-large, dense, complex-valued linear systems arising from the MM procedure. Target electrical sizes are such that MM linear systems can greatly exceed 10,000 (10K) equations, resulting in matrices having storage requirements substantially exceeding the available central memory or solid-state device (SSD). Further, because it is desired to obtain the monostatic RCS at a large number of illumination angles (right-hand sides), the number of right-hand sides can be in the thousands, approximating the order, $N$, of the MM matrix.

An initial strategy evolved to develop a complex-valued lower-upper matrix decomposition program that utilizes an efficient out-of-core scheme and is adaptable to multiple CPU usage. The result was the program, CLUD—Complex Lower-Upper Decomposition, with versions developed for the Cray-2, X-MP, and Y-MP. Subsequently, Cray analysts developed an out-of-core solver for the N right-hand sides problem, and then a parallel-processing version of the N right-hand-sides code. This work rapidly gained popularity among MM users, and Cray scientists have provided assistance to members of the CEM community in adapting these new tools to many different MM codes.

In 1989, J. Brooks of Cray Research took this work to its present state by developing a highly efficient out-of-core LU decomposition code for dense complex-valued matrices. His work, based on BLAS-3 algorithms (Basic Linear Algebra Subroutines), is summarized in Section 2.2 of this chapter. MM matrix sizes up to 40K×40K have been benchmarked. Concurrently using the eight processors available on the Cray Y-MP/8, average computation rates exceeding 2 billion floating point operations per second (2 Gflops) are achieved using 60 million words (60 Mwords) of central memory. In one benchmark example, a total program running time (wall clock time) of only 1.99 hours is required to process a 20K × 20K matrix. Brooks has also developed BLAS-3 based out-of-core solvers for symmetric matrices. These should have interesting applications to MM modeling of body-of-revolution types of objects. However, they are not covered in this chapter.

*b. Iterative Spectral Domain Methods*

As an alternative to the full-matrix MM, a variety of iterative spectral domain approaches for CEM has been developed for modeling EM wave scattering and RCS. Here, the fast Fourier transform (FFT) is widely used. In 1989, K. Ghosh of Cray Research developed parallel, out-of-core formulations of the Temperton FFT algorithm for 1-D, 2-D, and 3-D. The 1-D work is the focus of Section 2.3 of this chapter, but selected 2-D and 3-D benchmark results are also provided. On the Cray Y-MP/8, these codes achieved sustained computation rates as high as 2.3 Gflops. In one benchmark example, a wall clock time of only 18.4 sec is required to perform an FFT of size $2^{27}$ points (134 M). Ghosh's codes are currently being applied to optimize spectral domain formulations for modeling the RCS of large 3-D targets.

*c. Space-Grid Time-Domain Solvers*

In the past 4 years, non-matrix grid-based time-domain CEM algorithms have been the subject of increasing interest. These algorithms promise to avoid many of the problems of high-dimensional computational burdens, error accumulation, and slow convergence inherent in frequency-domain MM and iterative CEM techniques. They are becoming increasingly useful for modeling the RCS of structures of complex shape and material composition, and large electrical size (spanning more than 10 wavelengths in three dimensions).

Since 1990, Cray Research has concentrated on the optimization of two in-core, grid-based, 3-D time-domain EM codes: finite-difference time-domain (FD-TD) by Taflove [3]; and finite-volume time-domain (FV-TD) by Shankar [4]. The FD-TD work is summarized in Section 2.4 of this chapter. On the Y-MP/8, both FD-TD and FV-TD codes achieved sustained computation rates exceeding 1.6 Gflops. In one benchmark example for FD-TD, a wall clock time of only 3 minutes 40 seconds is required to solve for 23-million vector EM field components. This represents a speed-up factor of 7.97:1 relative to one Y-MP processor (maximum possible speed-up = 8:1).

We now proceed with the discussion of the three Cray-based CEM algorithms. The reader should note that all of the multi-tasked code performances and out-of-core performance rates are given implicitly for unloaded or lightly loaded machines. This, of course, is the maximum performance. Workers not having access to unloaded or lightly loaded machines may experience reduced performance, even if using the same software.

## 2.2 Large Complex LU Decomposition for the Method of Moments

This section discusses a new algorithm and code for performing out-of-core LU decompositions for large, dense, complex-valued matrices of the type resulting from MM modeling of radar cross-section. Because the method is based on BLAS-3 kernels, it runs at near peak performance on all Cray Research computers. In addition, the method employs asynchronous input/output (I/O) and requires a reduced number of total operations. The operation count can be further reduced by making use of larger memory sizes.

### *a. The Method*

*The Problem with CLUD.* An out-of-core LU decomposition solver for complex matrices (CLUD) has been available in the Cray Research BENCHLIB for some time. Although CLUD works well, it has two drawbacks that needed to be addressed for very large problems.

I. The input/output (I/O) for CLUD is either synchronous to disks, or synchronously staged from disk to the Cray solid-state device (SSD). If the matrix is scaled to fit entirely in SSD, this is not troublesome. In fact, CLUD runs at near-peak performance on Cray X-MP and Y-MP machines. However, a 20K $\times$ 20K MM matrix would require an 800-Mword SSD, which is not currently available. In CLUD, problems of this size require synchronous I/O between disks and SSD. That is, large "superslabs" of the matrix are moved synchronously between disk and SSD, and smaller "slabs" are transferred between the SSD and memory. (A slab is a matrix block consisting of a large number of adjacent columns of the matrix.) This method reduces I/O wait time considerably compared to small slabs to disk only, but the synchronous I/O from disk to SSD reduces overall performance for very large problems. The total amount of synchronous I/O to disk in CLUD is approximately

$$nslabs^2 * slabsize/2$$

where *nslabs* is the total number of slabs (or superslabs if using an SSD) and *slabsize* is the size of a slab (or superslab).

II. The CLUD algorithm is based on a SAXPY type kernel which works on individual columns. This kernel runs at peak performance on the Cray X-MP and Y-MP, but not on the Cray-2 system because of a high ratio of memory operations to computation.

*New Approach.* Because the Cray math software group had optimized the BLAS-3 (Basic Linear Algebra Subroutines) to run at near-peak performance on all Cray machines, an algorithm was used that was based on these kernels. A block-oriented method was adapted from LAPACK* to run out-of core. The original routine, CGETRF (LAPACK's standard in-core complex LU solver), uses two BLAS-3 kernels,

* LAPACK is a public domain, transportable linear algebra library in FORTRAN 77 designed to replace EISPACK and LINPACK libraries. It was jointly developed by Argonne National Laboratory, the Courant Institute of Mathematical Sciences, and the Numerical Algorithms Group, Ltd. It is based on BLAS-2 and BLAS-3 algorithms and solves dense linear algebra problems efficiently on high-performance computers.

CGEMM (complex matrix multiply) and CTRSM (complex triangular backsolve). Both of these kernels run at near-peak performance on all Cray machines.

To adapt CGETRF to run out of core, the matrix is divided into slabs. The matrix is decomposed from left to right, one slab at a time. Computation works on pairs of slabs. In order to compute a new leading slab, all preceding slabs need to be brought into memory, one at a time, for computation. This is an I/O pattern similar to that used in the existing CLUD code. However, three slab-sized memory buffers are used in the new code to allow for *asynchronous I/O*. The partial-pivoting scheme used in CGETRF is preserved in the new out-of-core version, which is designated CGETRFO. Slabs are updated according to an index vector. CGETRFO also employs an enhanced complex matrix multiply scheme, described below.

*Optimized Complex Matrix Multiplication.* Some additional features are used in CGETRFO to enhance the performance of complex matrix multiplication. Multiplication of the complex numbers $(a + ib) \times (c + id)$ normally requires the computation of $ac - bd$ and $ad + bc$. That is:

$$(a + ib) * (c + id) = (ac - bd) + i(ad + bc)$$

This requires four multiplications and two additions to compute. The computation can be completed with fewer multiplications (see Fam, Ref. [5]) using an identity attributed to Golub [6]:

$$(a + ib) * (c + id) = (a(c - d) + (a - b)d) + i(b(c + d) + (a - b)d)$$

Since the quantity $(a - b)d$ is used twice, this method takes only three multiplications and six additions. Since additions and multiplications are comparable on Cray machines, this identity does little for optimizing complex multiplication. However, if $a$, $b$, $c$ and $d$ are themselves the imaginary and real parts of two matrices, the identity still holds. That is, complex matrix multiplication can be written in a similar manner. Under this assumption, a multiplication is $O(n^3)$ operations while an addition is only $O(n^2)$. Thus, applying the Fam/Golub approach results in about 25% savings in the overall operation count for complex matrix multiplication. The price to pay for this is the workspace required to hold several temporary matrices.

A routine called CMXMA was written to take advantage of the efficient Fam/Golub complex matrix multiplication. CMXMA converts complex matrix multiplication to three real matrix multiplies and several matrix additions. This routine will be available in SCILIB 6.0 as CGEMMS.

In addition, the Strassen's real matrix multiply (SGEMMS) was used to further save on operations in CGETRFO. SGEMMS is a Strassen's algorithm [7] extension to the standard BLAS-3 matrix multiply routine, SGEMM. In this algorithm, the matrix is recursively cut in half. At each stage, the full-size matrix multiplication is converted to seven half-size matrix multiplications and several half-size matrix additions. Under a normal algorithm, a full-size matrix multiplication would be converted to eight half-size matrix multiplications and the total operations required would be conserved. Hence, SGEMMS reduces the operations necessary to about 7/8 at each stage in the recursion, if we ignore the $O(n^2)$ overhead of matrix additions. In general, doubling the size of a given matrix requires only 7/8 of the operations necessary using the traditional method. The recursive method is halted when sub-matrices reach sizes less than or equal to 64 (128 on the Cray-2 machine). Thus, large matrices have an increased relative speedup. For this reason, it is advantageous to run the decomposition with wide slabs. SGEMMS was written by the math software group and is scheduled for a future release of UNICOS.

### *b. The Results*

*Computation Rate for Matrix Multiplication.* Although the resulting routine requires a large workspace, a significant performance increase was achieved over CGEMM. The effective gigaflop* (Eflop) rates

* Complex matrix multiplication with the usual method takes $8n^3$ operations. However, as stated, the method used in CMXMA is based on a scheme that requires fewer operations. Thus, quoting the performance of CMXMA in actual gigaflops may be misleading when comparing algorithm and machine performance. To normalize the performance across different machines and algorithms, we define an "effective gigaflop" rate simply by assuming $8n^3$ operations and dividing by the computation time. Because the the actual operation count is less, of course, we see Eflop rates that seem to outperform the capabilities of the machines.

This parallels LINPACK guidelines that, for example, require one to divide $2/3n^3$ by the solution time when counting Gflops for the

| Cray-2/128 SRAM (sn2025) | | | Y-MP 8/32 6.41ns (sn1001) | |
|---|---|---|---|---|
| Size | Time (sec) | Eflops | Time (sec) | Eflops |
| 128 | 0.0085 | 1.96 | 0.0065 | 2.56 |
| 256 | 0.058 | 2.29 | 0.048 | 2.79 |
| 500 | 0.41 | 2.43 | 0.34 | 2.92 |
| 512 | 0.41 | 2.61 | 0.36 | 2.95 |
| 1000 | 2.75 | 2.91 | 2.27 | 3.58 |
| 1024 | 2.94 | 2.92 | 2.25 | 3.80 |
| 2048 | 20.6 | 3.33 | | |

**Table 2.1 Effective Gigaflop (Eflop) Rates for CMXMA.**

for routine CMXMA are shown in Table 2.1 for a Cray-2/128 having four processors and 128 Mwords of central memory, and a Cray Y-MP 8/32 having eight processors and 32 Mwords of central memory.

It should be noted that the column width used for slabs in CGETRFO corresponds to the size of the matrices multiplied by CMXMA. Therefore, wider slabs give increased computational performance for the LU problem.

*Overall LU Decomposition.* Table 2.2 shows the results achieved on several large LU decomposition problems on the 4-processor Cray-2/256 and the eight-processor Cray Y-MP 8/64.

To illustrate the significance of the benchmark results of Table 2.2, consider the 20K MM matrix case for the Y-MP 8/64. We see that 138 Gbytes of I/O are discharged to and from 7 DD-40 disk drives during this run. Yet, the wall clock time is only 1.99 hours, of which only 228 sec (3.8 min) represents I/O wait time. In fact, 90% of the actual I/O operations are performed concurrently with the floating point arithmetic by virtue of the asynchronous I/O scheme, and therefore do

$1000 \times 1000$ real-matrix LU decomposition benchmark, regardless of how many operations one's solution actually requires. One drawback of counting Eflops for CMXMA is that it is slightly less accurate because the total number of additions is increased. However, there seems to be general agreement that the Eflop rate is an easier metric for explaining performance than alternatives such as just using the solution time.

| M | N | C | I/O | IOT | T | GFA | GFE | D |
|---|---|---|---|---|---|---|---|---|
| C2 | 10K | 2K | 6.4 | 37 | 0.33 | .968 | 2.20 | 8 |
| C2 | 20K | 1K | 73.6 | 304 | 2.69 | 1.12 | 2.20 | 8 |
| C2 | 30K | 968 | 245 | 837 | 9.11 | 1.12 | 2.19 | 8 |
| C2 | 40K | 1K | 550 | 1331 | 20.8 | 1.16 | 2.28 | 8 |
| Y | 10K | 910 | 11.2 | 57 | 0.24 | 2.20 | 2.94 | 7 |
| Y | 20K | 500 | 138 | 228 | 1.99 | 2.24 | 2.98 | 7 |
| Y | 30K | 334 | 671 | 697 | 7.15 | 2.10 | 2.80 | 7 |
| Y | 40K | 250 | 2086 | 1824 | 16.8 | 2.12 | 2.82 | 7 |

**Table 2.2 Benchmarks for Out-of-Core LU Decomposition Code. Abbreviations: M= Machine (C2 = Cray-2/256 or Y = Y-MP 8/64); N= Matrix Order N; C = Columns per slab; I/O = Input/Output to disks (Gbytes); IOT = I/O wait time (sec); T = Wall clock time (hours); GFA = Actual Gflops; GFE = Effective Gflops; D = Number of DD-40 disks.**

not contribute to the observed wait time As matrix size increases, the relative efficiency of the asynchronous scheme improves, with the I/O concurrency factor rising to 94% for the 30K matrix, and to 95% for the 40K matrix. Effectively, what has been shown is that the massive I/O associated with solving huge MM matrices can be almost completely buried. This permits Crays with small central memories (as little as 16 Mwords) to efficiently process these problems using relatively inexpensive disk drives, and to achieve average computation rates approaching the maximum peak values permitted by the hardware.

For such large matrices, roundoff error can become a serious problem. The code discussed above uses partial pivoting which improves stability; however the matrix multiply kernel is less accurate than CGEMM. In our test cases, the code achieves good results to 13 decimal places solving one right-hand side in diagonally dominant random matrices of size 20K, but only 4 or 5 decimal places for matrices having randomly distributed entries between $-0.5$ and $+0.5$. In these cases, solutions can be improved through single-precision iterative refinement. More work is needed in the detailed error analysis of this method.

## 2.3 A Parallel Out-of-Core 1-D FFT for Cray Y-MP Systems

Applications demanding use of efficient Fourier transform algorithms in computational modeling and data processing exist in many areas of science and engineering. A considerable amount of work has been directed towards designing efficient fast Fourier transform (FFT) algorithms for vector parallel supercomputers (for example, see [8]). Users of Cray X-MP and Y-MP systems running either COS or UNICOS have access to several 1-D FFT routines resident in SCILIB and BENCHLIB. These routines have been developed and packaged during the past decade by various analysts. A summary of the routines available to users of the Cray X-MP and Y-MP is provided in [9]. Some Cray users have a need for out-of-core solutions because of a mismatch between the size of the problems being solved and the central memory resources available on their systems. This prompted Cray Research to develop and package very efficient out-of-core FFTs centered around CFFTMLT, an existing kernel from SCILIB.

This section describes parallelized out-of-core 1-D FFTs developed for use on Cray Y-MP computer systems.* We discuss the performance delivered by the I/O subsystem of the Y-MP, including techniques for optimizing the I/O. Benchmarks demonstrate that the I/O subsystem allows transfer of data to a variety of physical and logical devices at average speeds close to the peaks expected from the hardware, that is, close to 2 Gbytes/sec. It is significant that this rate is competitive with a transfer of 2 words per cycle from memory to an individual processor. The transfer of data to striped disks occurs at rates of the order of 100 Mbytes/sec. The performance of cached disks lies somewhere in between.

This section also discusses a parallelized 1-D FFT built around an existing single-processor kernel. It is significant that when the SSD is used as an auxiliary device, the I/O transfer rate of 2.0 Gbytes/sec encourages the use of concurrent processing for out-of-core FFTs. As an example, the CPU usage for an FFT of size 32-million points is

* The FFTs discussed here are based on powers of two. We find no memory advantage from using FFTs which are not based on powers of two. Any degradation in memory access for power-of-two FFTs has been minimized both by the design of the memory subsystem and by the algorithmic methods applied.

23.9 sec on a single Cray Y-MP processor, and the associated wall clock time is 26.3 sec. The I/O time of 2.4 seconds implies a transfer rate of 1.8 Gbytes/sec.

All of the measurements were performed on a Cray Y-MP 8/128 (sn1033) at Mendota Heights executing UNICOS 5.1. Each test was repeated multiple times, with the best performance data observed during these repetitions reported here. The machines were usually lightly loaded.

*a. Basis*

The need for out-of-core FFTs arises from applications that require transforms where the size of the sequence exceeds the size of memory available. In some instances, the total memory required exceeds the size of the physical memory a process can access. The philosophy underlying an out-of-core solution is that pieces of the sequences fit in memory and therefore the transform could be computed in parts.

The theoretical foundation for an out-of-core formulation (see the Appendix of this chapter for the details) is summarized as follows. We view a 1-D data sequence as an $N_1$ by $N_2$ matrix where $N = N_1 N_2$ and $N_1 = N^{1/2}$. First, the rows of the matrix need to be transformed. We compute $N_1$ transforms, each of length $N_2$. Since the data resides on disk, the points of the sequence have to be read with a stride. The I/O would be very inefficient if we were to read with a stride and fetch the rows. A better method is to transpose the matrix and rewrite the matrix to the external device. When the transpose is performed, we use the fact that if a matrix $A$ is viewed as a collection of submatrices $A_{ij}$, then its transpose $B$ is obtained by transposing the submatrices $A_{ij}$ and forming the new matrix $B = [B_{ij}]$, where $B_{ij} = A^T_{ji}$. These submatrices are records now, written one at a time to the external devices. Because they are relatively large in size, the I/O transfers proceed at near-peak speeds.

A phase shift is performed as indicated in (7) of the Appendix. Next, one needs to compute $N_2$ transforms of length $N_1$. A transpose is necessary again to access the elements of the sequence as a column instead of as a row. So the transpose is performed before the transform is computed. Finally, a transpose is necessary to put the data in the right order. The two multiple transforms are excellent candidates for concurrent execution. The phase shift is performed on elements of an $N_1$ by $N_2$ matrix and lends itself to parallel execution, as does the transpose.

### *b. Performance, Single CPU, Using the Solid-State Device (SSD)*

The out-of-core software for 1-D FFTs has been designed to exploit the device-independent I/O software available on Cray Y-MP computer systems. What this implies for the user is that the same software may be used regardless of the source and destination of the data sets representing the linear sequence to be transformed. From an operational standpoint, this is accomplished by specifying the characteristics of the input file and a scratch file in a shell script, and is essentially a user-definable parameter that is set at execution time. The friendliness of this approach is a convenience that allows using the out-of-core software in widely different site-dependent situations.

In this section, we consider Cray's primary physical means of external mass storage, the solid-state device (SSD). We shall look at three different ways of configuring the SSD as a fast logical device, and discuss the associated rates and elapsed times of data transfer:

1. SSD configured as a secondary data segment (SDS);
2. SSD configured as a filesystem (e.g. `/ssd`); and
3. SSD used as a logical device cache for a disk-configured filesystem, i.e., a filesystem such as `/tmp` could be cached on the SSD.

The reader is referred to [10] for the Unix environment variables needed for assigning disks and SSDs.

To be specific, consider as the physical device a 512 Mword SSD. The three different logical devices are chosen via shell scripts that would accomplish the equivalent of the commands below.

```
# CASE 1 - SDS
env FILENV=filezz assign -a SDS -s u -n 65536 fort.31
env FILENV=filezz assign -a SDS -s u -n 65536 fort.32
env FILENV=filezz a.out < infile > outfile
# CASE 2 - /ssd
env FILENV=filezz assign -a /ssd/abc/xx1 -s u -n 65536 fort.31
env FILENV=filezz assign -a /ssd/abc/xx2 -s u -n 65536 fort.32
env FILENV=filezz a.out < infile > outfile
# CASE 3 - ldcache  (/tmp as cached filesystem)
env FILENV=filezz assign -a /tmp/abc/xx1 -s u -n 65536 fort.31
env FILENV=filezz assign -a /tmp/abc/xx2 -s u -n 65536 fort.32
env FILENV=filezz a.out < infile > outfile
```

| | | SDS (s) | | /ssd (s) | | ldcache (s) | |
|---|---|---|---|---|---|---|---|
| S | Size | CPU | Wall | CPU | Wall | CPU | Wall |
| 20 | 1. M | 0.51 | 0.57 | 0.52 | 0.58 | 0.52 | 0.67 |
| 21 | 2.1M | 1.07 | 1.18 | 1.08 | 1.21 | 1.08 | 1.39 |
| 22 | 4.2M | 2.22 | 2.45 | 2.24 | 2.87 | 2.24 | 2.52 |
| 23 | 8.4M | 4.57 | 5.03 | 4.61 | 5.16 | 4.60 | 5.86 |
| 24 | 16.8M | 9.40 | 10.3 | 9.47 | 10.6 | 9.47 | 12.4 |
| 25 | 33.6M | 19.5 | 21.4 | 19.6 | 21.8 | 19.6 | 40.4 |
| 26 | 67.1M | 40.3 | 44.6 | 40.7 | 45.1 | 40.7 | 70.5 |
| 27 | 134.2M | 82.8 | 91.3 | 82.8 | 92.0 | 83.4 | 340.0 |

**Table 2.3 Comparison of SDS, /ssd, and ldcache as external devices for out-of-core 1-D FFT (single Cray Y-MP processor). S = $\log_2$(size).**

The application software implementing out-of-core FFTs utilizes "BUFFER IN" and "BUFFER OUT" calls to request transfer of data. The option named "-s u" is used in the scripts above to request bypassing of library and system buffers. The condition imposed by this request is that all transfers have to be a multiple of the sector size (512 words). Positioning within a file is accomplished with the aid of calls to "SETPOS". It is significant that all of the high performance I/O can be accomplished from within the FORTRAN source. Two other issues are relevant. First, FORTRAN's sequential "read" and "write" statements could be used as well, in conjunction with calls to "SETPOS". Second, the choice of an external device is made when the code is executed, not during compilation or linking. The Cray I/O libraries allow this vital "device independence" for all devices, from the fastest SDS down to single disks.

Table 2.3 shows the CPU time and elapsed (wall clock) time for FFTs of sizes $2^{20}$ through $2^{27}$ when using these devices as secondary storage for input/output data and for intermediate storage. A graphical description of this table is shown in Fig. 2.1.

The following comments pertain to these results:

1. The I/O to the SSD is optimum when the SSD is used as a secondary data segment (SDS) primarily because the user is able to bypass the overhead of executing code necessary for file manage-

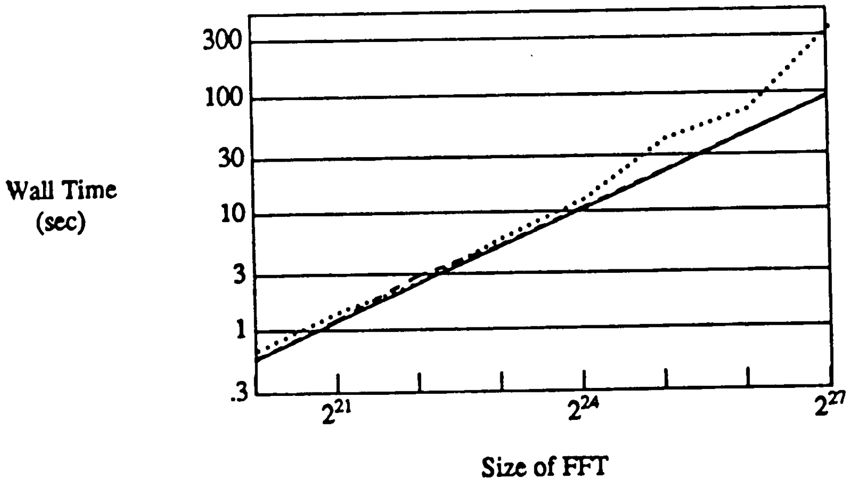


**Figure 2.1** **Wall time for 1-D out-of-core FFT — comparison of SDS, /ssd and LDCACHE. Solid: SDS, dashed: /ssd, dotted: LDCACHE.**

ment. Each I/O operation involves a system call to initiate the transfer (overhead $\cong 10\mu s$) followed by a high-speed data transfer from the SSD to central memory. Since this takes place at a rate of 2 words/clock, the Y-MP cycle time of 6 nsec leads to a peak transfer rate of 2.5 Gbytes/sec.

2. When reading to and writing from a file on **/ssd**, one incurs the cost of performing operations relevant to UNICOS file management. It is believed that the physical data transfer from a file resident on **/ssd** to central memory occurs at the fast SDS-to-memory transfer rate. In fact, it is the added software overhead that leads to a smaller effective transfer rate to **/ssd** than to SDS.
3. The performance is consistently better when the file is resident on **/ssd** than when the file system is cached (on the SSD) for the simple reason that there is overhead involved in managing a cache. Here, the net data transfer rate to central memory is usually much lower than the peak rate of 2 Gbytes/sec because of cache misses, falling as low as 100 Mbytes/sec, as shown in Fig. 2.2.

*c. Performance, Single CPU, Using Striped Disks*

Disk striping is a technique for enhancing the effective transfer

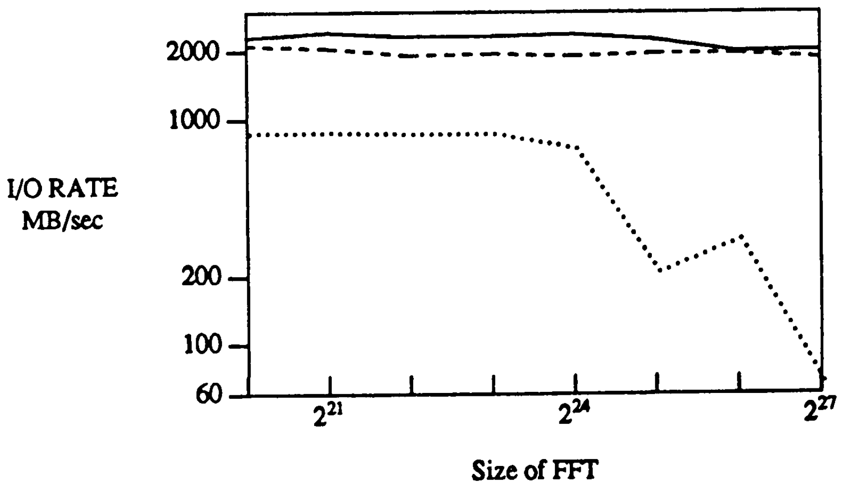


**Figure 2.2** **I/O Rates on SDS /ssd ldcache for one-dimensional out-of-core FFTs (Rates 1.07 faster on a 6 ns CRAY Y-MP.) Solid: SDS, dashed: /ssd, dotted: LDCACHE.**

rates to and from disk-based file systems by employing a form of concurrent I/O to multiple disk devices. In this section, we will use the out-of-core FFT as an example to show how a system of striped disks can provide an alternative to an SSD in Cray Y-MP systems without an SSD. The data transfer rate to a group of 12 striped disks is of the order of 100 Mbytes/sec, provided that the transfer size is large enough to exploit the bandwidth of each of the component disks. It should be emphasized that under UNICOS, individual users have the resources available to stripe a file, provided that the file system has been configured to use multiple partitions.

We first discuss the setup that allows users to choose striped disks as the external device. Consider the following:

```
# Out-of-core FFT with striped disks as the external device
setf -p1-12 -n65536b:42b /tmp/abc/xx1
setf -p1-12 -n65536b:42b /tmp/abc/xx2
assign FILENV=filezz assign -a /tmp/abc/xx1 -s u fort.31
assign FILENV=filezz assign -a /tmp/abc/xx2 -s u fort.32
assign FILENV=filezz a.out < infile > outfile
```

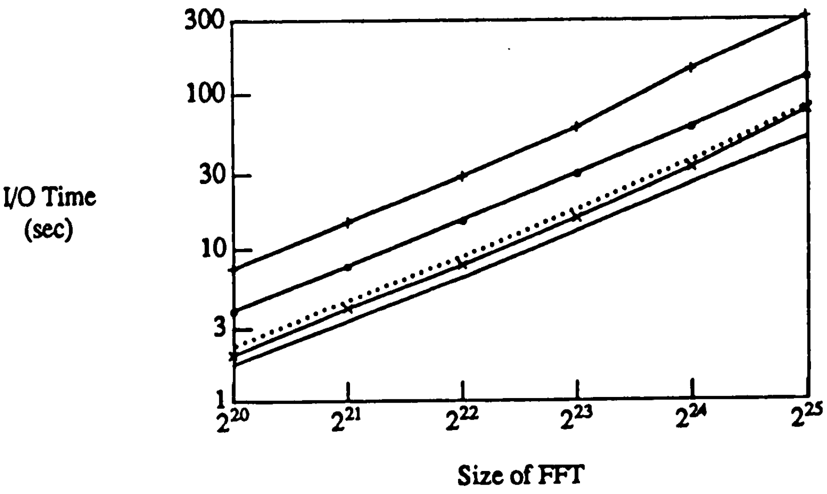


**Figure 2.3 I/O Wait time for one-dimensional out-of-core FFT with striped DD49 (solid - 12disks, x - 10disks, dotted - 8disks, bullet - 4disks, + 2disks).**

In the script above, we have striped files **xx1** and **xx2** over 12 partitions, pre-allocating a total of 65,536 blocks of size 512 words. In this example, the allocation size per partition is 42 blocks of size 512 words, a number that corresponds to the track size of a DD-49. This indicates a request that for large transfers, entire tracks be written out to the partitions concurrently.

The UNICOS command "**df -p**" displays the layout of a file system and may be used to plan the striping of a preallocated file. The character string in the last column of a display from "**df**" indicates the I/O processor (IOP) and the I/O system (IOS) used. Concurrent I/O is achieved by using multiple IOSs, multiple IOPs within each IOS (such as a BIOP and a DIOP), and multiple devices to each IOP.

Figure 2.3 shows the I/O wait time of a system configured to provide striping for as many as twelve DD49s. The wait time for an FFT of size 16.8M ($2^{24}$) is seen to be less than 30 sec. From Table 2.3, we see that the corresponding wait time using an SDS is 0.9 sec (10.3 sec – 9.4 sec). Thus, the system of 12 striped disks is a factor of about 30 times slower than an SDS, achieving a data transfer rate of about 80 Mbytes/sec.

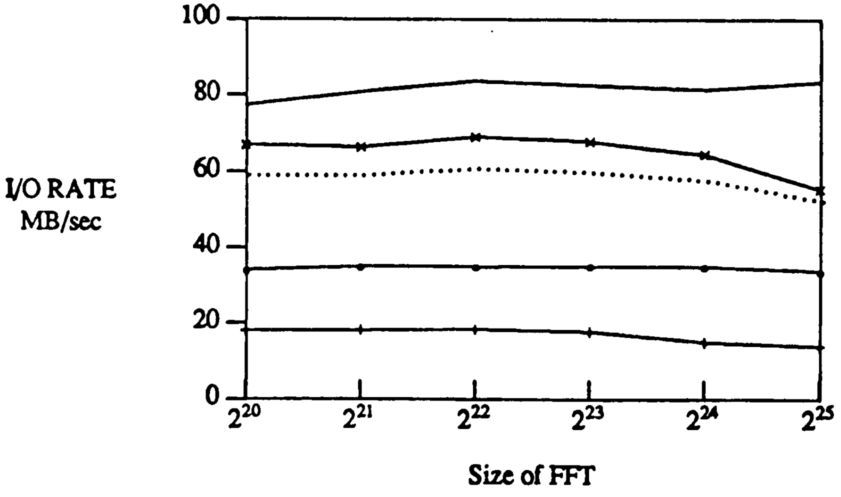


**Figure 2.4 I/O rate for 1-D out-of-core FFT with striped DD49 (solid - 12disks, x - 10disks, dotted - 8disks, bullet - 4disks, + 2disks).**

Figure 2.4 graphs measured I/O data transfer rates achieved by striping DD49s. The usual single-disk sustained rate of about 9 Mbytes/sec is seen to nearly double with two striped disks, and quadruple with four disks. The trend continues, and data transfer rates of 80 Mbytes/sec are realized for 12 disks, as stated above. The peak transfer rate realized in this application depends on the transfer size. In order to make each disk of a striped disk system deliver at its peak, the transfer sizes need to be large enough to make the latency and seek time relatively small.

The size of the sectors allocated per partition can also affect the I/O time. Intuitively, we expect that for large transfers, a large allocation size such as two tracks (84 sectors) is more efficient than allocating one track (42 sectors) at a time. In fact, measurements reveal that 2-track allocation is preferred even for moderately sized transfers. Half-track allocations should be avoided because the effective data transfer rate is low due the overhead of latency and seek time.

DD40s and DD49s have been reported as having similar sustained data transfer rates of 10 Mbytes/sec. To check this, we implemented out-of-core FFTs first with a group of eight striped DD40s and then with a group of eight striped DD49s, and compared the I/O wait times.

| | | C | Wall-Clock Time (s) | | | |
|---|---|---|---|---|---|---|
| S | Size | 1 CPU | 1 CPU | 2 CPUs | 4 CPUs | 8 CPUs |
| 20 | 1M | 0.51 | 0.57 | 0.32 | 0.22 | 0.13 |
| 21 | 2.1M | 1.07 | 1.18 | 0.66 | 0.40 | 0.27 |
| 22 | 4.2M | 2.22 | 2.45 | 1.36 | 0.80 | 0.54 |
| 23 | 8.4M | 4.57 | 5.03 | 2.80 | 1.64 | 1.09 |
| 24 | 16.8M | 9.40 | 10.3 | 5.72 | 3.34 | 2.19 |
| 25 | 33.6M | 19.5 | 21.4 | 11.8 | 6.85 | 4.44 |
| 26 | 67.1M | 40.3 | 44.6 | 24.3 | 14.0 | 9.18 |
| 27 | 134.2M | 82.8 | 91.3 | 49.7 | 28.7 | 18.4 |

**Table 2.4 Parallel Out-Of-Core 1-D FFT. S = $\log_2$(size); C = CPU Time(s).**

After statistical variations were taken into account, we found little difference between the I/O performance of the striped DD40s and DD49s.

### *d. Parallel Out-of-Core 1-D FFT*

In this section, we discuss preliminary results for programming the out-of-core 1-D FFT for concurrent operation on up to eight processors of the Cray Y-MP. Cray autotasking is used to exploit the parallelism that is built into the basic algorithm (Section 2.3.b and the Appendix). The code is parallelized simply by inserting two autotasking directives for concurrent execution of the work performed by CFFTMLT, and by allowing the preprocessor to identify the parallelism in the phase shift and the transpose. Wall-clock times for the parallel version of the code are shown in Table 2.4.

The 5:1 limit in speed-up factor seen in Table 2.4 is due to the fact the total number of FFTs computed by each processor is small. From the discussion of Section 2.3b, the size of the fraction of the given sequence of size $N$ that is memory-resident at any time is determined by the product of $N^{1/2}$ and a width parameter $w$, where $w$ is also the number of 1-D FFTs to be solved concurrently by $N_p$ processors (a burden of $w/N_p$ FFTs per processor, executed by invoking CFFTMLT). We choose $w = M/N^{1/2}$, where $M$ is in the order of the available central memory. Referring to the analysis of CFFTMLT's per-

| | | Wall-Clock Time(s) | | | | |
|---|---|---|---|---|---|---|
| X | Y | 1 CPU | 2 CPUs | 4 CPUs | 8 CPUs | M |
| 256 | 256 | 0.0175 | 0.00887 | 0.00457 | 0.00277 | 6.32 |
| 512 | 256 | 0.0377 | 0.0190 | 0.00964 | 0.00544 | 6.93 |
| 256 | 512 | 0.0377 | 0.0190 | 0.00965 | 0.00543 | 6.94 |
| 512 | 512 | 0.0805 | 0.0404 | 0.0204 | 0.01050 | 7.67 |
| 1024 | 1024 | 0.355 | 0.179 | 0.0900 | 0.0452 | 7.85 |
| 2048 | 2048 | 1.61 | 0.804 | 0.399 | 0.203 | 7.93 |
| 4096 | 4096 | 6.91 | 3.450 | 1.750 | 0.877 | 7.88 |

**Table 2.5 Parallel In-Core 2-D FFT. X = $x$-size; Y = $y$-size; M = Max. Speed-Up.**

| Size | Wall-Clock Time(s) | | | | |
|---|---|---|---|---|---|
| $(x = y = z)$ | 1 CPU | 2 CPUs | 4 CPUs | 8 CPUs | M |
| 16 | 0.00286 | 0.00157 | 0.00100 | 0.00087 | 3.28 |
| 32 | 0.0134 | 0.00700 | 0.00374 | 0.00218 | 6.16 |
| 64 | 0.0846 | 0.0428 | 0.0217 | 0.0112 | 7.55 |
| 128 | 0.762 | 0.380 | 0.192 | 0.0966 | 7.89 |
| 256 | 6.85 | 3.369 | 1.706 | 0.858 | 7.98 |

**Table 2.6 Parallel In-Core 3-D FFT. M = Max. Speed-Up.**

formance presented in [9], we note that the cost per FFT decreases as the total number increases. Higher speed-up factors here clearly require increasing the total number of FFTs computed by each processor.

### *e. Maximum Speed-Up Using Autotasking: 2-D and 3-D FFT Benchmark Data*

We now provide benchmark data that show that speed-up factors substantially greater than 5:1 can be achieved for FFTs using Cray autotasking on the Y-MP/8. Tables 2.5 and 2.6 provide benchmark results for the wall-clock times of parallel in-core 2-D and 3-D FFTs, respectively, based on CFFTMLT.

It is clear that maximum speed-up factors can approach 8:1 (the number of processors of the Cray Y-MP/8) for the multidimensional FFTs, indicating an excellent distribution of workload among the processors. Using the following formulas for the operations count for the 1-D, 2-D, and 3-D complex FFT algorithms employed by Cray Research:

| | |
|---|---|
| $5 * m * \log_2(m)$ | 1-D FFT of dimension $m$ |
| $5 * m * n * \log_2(m * n)$ | 2-D FFT of dimensions $m$ and $n$ |
| $10 * l * m * n * \log_2(l * m * n)$ | 3-D FFT of dimensions $l$, $m$, and $n$ |

FFT performance on the Y-MP/8 has been measured in the range of less than 100 Mflops to over 2.35 Gflops, depending upon the dimensionality and respective sizes of $l$, $m$, and $n$.

## 2.4 Multiprocessing Space-Grid Time-Domain Codes

This section reviews computational aspects of an emerging class of approaches for numerical modeling of EM wave interactions with large, complex structures: direct space-grid, time-domain solvers for Maxwell's time-dependent curl equations. The primary current approaches in this class, the finite-difference time-domain (FD-TD) and finite-volume time-domain (FV-TD) techniques [3, 4], are remarkably robust, providing highly accurate modeling predictions for a wide variety of EM wave interaction problems. They are analogous to existing mesh-based solutions of fluid-flow problems in that the numerical model is based upon a direct, time-domain solution of the governing partial differential equation. Yet, they are nontraditional approaches to numerical electromagnetics for engineering applications, where frequency-domain integral equation approaches (MM) have dominated.

### *a. Computational Aspects*

*Algorithms.* FD-TD and FV-TD methods for Maxwell's equations are based upon fine-grained, volumetric sampling of the unknown near-field distribution within and surrounding the structure of interest. The sampling is at sub-wavelength (sub-$\lambda_o$) resolution to avoid aliasing of the field magnitude and phase information. Overall, the goal is to provide a self-consistent model of the mutual coupling of all of the electrically-small volume cells comprising the structure and its near field, even if the structure spans tens of $\lambda_o$ in 3-D and there are tens of millions of volume cells.

The primary FD-TD and FV-TD algorithms used today are fully explicit, second-order accurate grid-based solvers employing highly vectorizable schemes for time-marching the six vector components of the EM near field at each of the volume cells. The explicit nature of the solvers is maintained by either leapfrog or predictor-corrector time integration schemes. Present methods differ primarily in how the space grid is set up (either regular, unstructured, or body-fitted), and how EM field continuity is maintained at the interfaces of adjacent volume cells. As a result, the number of floating point computations needed to update the six field vector components at a single cell over one time step can vary by about one order of magnitude from one algorithm to another.

However, the choice of algorithm is not at all straightforward, despite this large range of computational burden. There is an important tradeoff decision to be made. Namely, the simpler solvers have more involved mesh generation requirements, and the meshes they utilize may not be compatible with those used in other engineering studies (primarily computational fluid dynamics and structural dynamics for aerospace structures). The more complex solvers can utilize existing mesh generators, but require substantially more computer arithmetic per space cell per time step.

*Vectorizability and Concurrency.* Both FD-TD and FV-TD algorithms (in-core) are highly vectorized, having been benchmarked at about 200 Mflops on a single Cray Y-MP processor for real models. However, the attainment of even higher Mflop rates may be hampered by the fact that the the space grids have an unavoidable number of non-standard cells requiring either scalar operations, mixed scalar-vector operations, or odd-lot vector operations working on small sets of data. These non-standard cells result from a number of factors, including the need to program a near-field radiation condition at the outermost grid boundary (simulating the grid continuing to infinity), and the need to stitch varying types of meshes together to accommodate complex structure shapes and cavities.

On the Cray Y-MP/8, it has been found possible to achieve nearly 100% concurrent utilization of all eight processors using autotasking for both 3-D FD-TD and FV-TD in-core codes (see discussion in Section 2.4.b for the FD-TD case). This is despite the codes' residual scalar, mixed scalar-vector, and odd lot vector operations mentioned above, which serve to limit vectorization. Only minor modifications to the original single-processor FORTRAN were required for the FD-TD case. These were accomplished over a period of several days.

*Storage.* Years of modeling experience with FD-TD and FV-TD have shown that $0.1\lambda_o$ is the coarsest space resolution suitable for engineering accuracy ($\pm$1dB) in predictions of the near and far fields. That is, the space grid must resolve the shortest EM wavelength in its domain (highest frequency component of the illuminating pulse) by better than 10 parts. It is easily seen that at least 1,000 volume cells per $\lambda_o^3$, equivalently 6,000 vector-field components per $\lambda_o^3$, must be maintained. For the simplest grid-based time-domain Maxwell's solver, FD-TD, applied to model non-dispersive media, the required computer storage is 18,000 real words per $\lambda_o^3$. (The factor of three arises from the use of one real word to store the most recently computed value of each vector field component, and two real words to store the two material-dependent updating coefficients associated with each vector field component.)

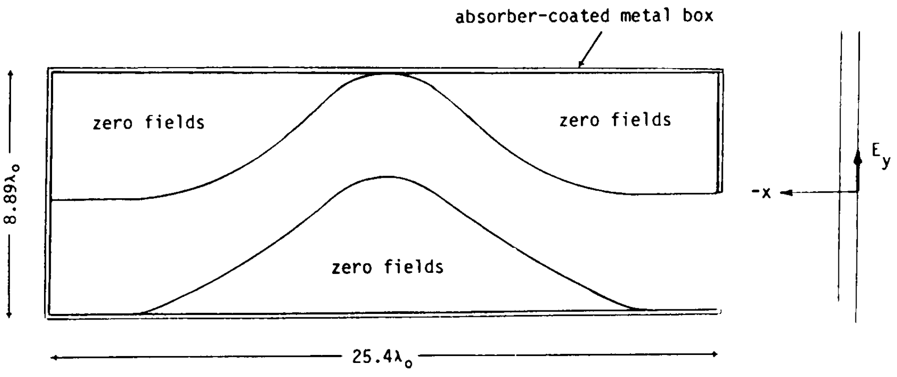
With current interest in computational modeling of 3-D structures spanning more than 20 $\lambda_o$ (i.e., having volumes exceeding about 10,000 $\lambda_o^3$), we see that FD-TD memory requirements may exceed 200 Mwords. The cubic dependence of the required computer memory upon structure dimension points the way toward billion-word storage needs for FD-TD in the near future, and even larger demands by other grid-based solvers using more elaborate algorithms. These requirements exceed existing Cray-based possibilities for central memory and SSD, and can be easily scaled to exhaust even the most ambitious Cray Research plans for such memory resources in the future. Clearly, what is needed is the application of out-of-core techniques to the grid-based time-domain Maxwell's solvers.

In fact, the out-of-core techniques discussed in detail in Section 2.3 in the context of performing very large FFTs, and already applied to very large MM matrices as discussed in Section 2.2, have already been applied successfully to the 3-D FD-TD algorithm. Developed for a single-processor of the Cray X-MP, the out-of-core FD-TD code uses asynchronous I/O, variable buffering, and disk-striping to achieve I/O concurrency of better than 95% relative to the computer arithmetic operations when 12 drives are used. In effect, the observed I/O wait time (impact on wall clock time) is less than 1/20 of the actual time for transferring data to and from the drives. This code is being adapted to UNICOS for the Cray Y-MP/8, eventually permitting eight-processor autotasking. The result of these efforts should be a 3-D FD-TD code that can solve for more than one-billion vector field unknowns in 2 to 4 hours, if CPU concurrency can be maintained near the high level measured for a multiprocessing in-core FD-TD code (see below).

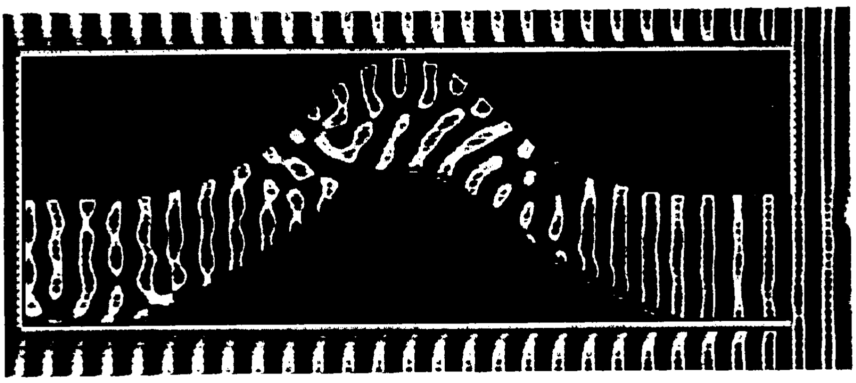
### *b. Example: Jet Engine Inlet, Conformally Modeled*

*Problem Description.* To illustrate the use of a multiprocessing in-core FD-TD code to model the RCS properties of an electrically large 3-D structure of engineering significance, we consider the serpentine jet engine inlet of Fig. 2.5a. Here, the overall system design problem involves sizing and shaping the engine inlet to meet both aerodynamic specifications (thrust) and EM specifications (monostatic RCS response for continuous sinusoidal illumination at 10 GHz). For the purposes of this publication, we shall assume that the inlet is embedded within a simple rectangular metal box coated with commercially available Emerson & Cuming Type AN-73 radar absorbing material (affording approximately 30 dB suppression of EM wave reflections at 10 GHz). Thus, the FD-TD computed near-field and far-field EM response is primarily a function of the inside wall shaping of the inlet, and not any exterior embedding.

In Fig. 2.5a, the incident wave is assumed to propagate from right to left (in the $-x$ direction), and is polarized so that its electric field points across the narrow dimension of the inlet (the $+y$ direction). The aperture of the inlet is located at the right, and the inlet is shorted by a conducting wall representing the turbine assembly at the far left. With the box dimensions set at 30" × 10.5" × 10", the overall inlet/box target configuration spans $25.4\lambda_o \times 8.89\lambda_o \times 8.47\lambda_o$ at 10 GHz. For this target, the FD-TD space cell size is 1/8" ($\lambda_o/9.43$), and the overall lattice has $270 \times 122 \times 118$ cells containing 23,321,520 unknown field components ($4,608\lambda_o^3$). Starting with zero-field initial conditions, 1,800 time steps are used (95.25 cycles of the incident wave) to march the field components to the sinusoidal steady state. The computer running time is 3 min, 40 sec per monostatic angle on the Cray Y-MP/8 using eight-processor autotasking. An excellent processor-concurrency factor of 7.97/8 is achieved. This yields an average computation rate of 1.6 Gflops.



(a) Geometry of engine inlet embedded in a rectangular metal box coated with radar absorbing material. 10-GHz incident wave propagates from right to left.



(b) Instantaneous distribution of the total $E_y$ field component at time step no. 1701 in a 2-D observation plane cutting through the z-center of the geometry.

**Figure 2.5 Conformal-mesh FD-TD computational modeling of a 3-D jet engine inlet (eight-processor Cray Y-MP, 1.6 Gflops average rate, 3 min 40 sec solution time for 23-million field components).**

Figure 2.5b depicts the instantaneous distribution (positive and negative values) of the total $+y$-directed electric field component in a 2-D observation plane ($x$ - $y$ plane) cutting through the $z$-center of the 3-D engine inlet. This photograph is derived from a color videotape display of the propagating electric field penetrating the inlet, generated directly by the FD-TD time-stepping. The display is taken late in the time-stepping when the internal field is settled into a standing wave.

Clearly, in addition to simple data for the RCS pattern, FD-TD modeling provides details of the complex standing wave pattern within the engine inlet, especially when visualized in the time domain using color video technology. The latter visualization shows that a general pulsing of the field pattern within the inlet occurs in the sinusoidal steady state, emitting backscattered energy in a regular series of bursts. It may be possible to use this highly detailed time-domain near-field information to better design such articles in the future.

### *c. Present Work and Future Directions*

At present, grid-based time-domain CEM models of 3-D structures spanning more than $30\lambda_o$ are being developed for the Cray Y-MP/8. These are apparently the largest detailed CEM models ever attempted. Work at this time is in the following areas:

(1) Automated mesh generation;
(2) Multi-processing, out-of-core software;
(3) Sub-cell models for fine-grained structural features such as coatings;
(4) Higher-order algorithms; and
(5) Application to non-traditional CEM areas, including ultra-high speed phenomena.

By 1993, the next-generation Cray Y-MP/16 should be routinely available to the academic and engineering communities. The Y-MP/16 will have up to six times the performance of the Y- MP/8 system. The performance gains come from the following:

1. $1.5\times$ from the clock period;
2. $2\times$ from the double pipes in each CPU; and
3. $2\times$ from double the number of processors.

Extrapolating from present benchmarks with the Y-MP/8, this machine should provide a steady 10-Gflop computation rate for grid-based time-domain CEM codes when using 16-processor autotasking. The proverbial "billion-unknown" CEM problem (a 3-D computational

| **Size** | CPU time | Elapsed Time (seconds) | | | |
|---|---|---|---|---|---|
| | (seconds) | 1 CPU | 2 CPUs | 4 CPUs | 8CUPs |
| 1048576 | 0.17 | 0.21 | 0.12 | 0.08 | 0.06 |
| 2097152 | 0.36 | 0.44 | 0.25 | 0.16 | 0.12 |
| 4194304 | 0.76 | 0.91 | 0.53 | 0.34 | 0.25 |
| 8388608 | 1.58 | 1.87 | 1.09 | 0.70 | 0.52 |
| 16777216 | 3.24 | 3.82 | 2.21 | 1.41 | 1.03 |
| 33554432 | 6.73 | 7.88 | 4.55 | 2.88 | 2.10 |
| 671088864 | 14.0 | 16.3 | 9.34 | 5.88 | 4.23 |

**Table 2.7 Parallel out-of-core 1-D FFTs for CRAY C90 systems. Notes: Memory used: 67 MW I/O device: 256 MW SSD**

volume of about 150,000 $\lambda_o^3$) could be completed in as little as 30 minutes per monostatic RCS observation. Multiprocessing out-of-core software should enable even larger volumes to be modeled in their entirety. Using such software, the era of the "entire airplane in the grid" would be opened for a number of important defense systems for radar frequencies up to about 10 GHz. Automated geometry generation would permit the CEM modeling to utilize structure databases developed by non-EM engineers, leading to design cost reduction and the possibility of innovative design optimizations.

## 2.5 Addendum

Since the completion of this chapter the CRAY C90 system has become operational. Tables 2.7 through 2.14 update timings for FFTs and LU decomposition on one to sixteen processors of this system.

| Size | | Elapsed Time (seconds) | | | | |
|---|---|---|---|---|---|---|
| Nx | Ny | 16 CPUs | 8 CPUs | 4 CPUs | 2 CPUs | 1 CPU |
| 64 | 64 | 0.000185 | 0.000313 | 0.000579 | 0.00113 | 0.00222 |
| 128 | 128 | 0.000350 | 0.000637 | 0.00123 | 0.00243 | 0.00483 |
| 256 | 256 | 0.000770 | 0.00147 | 0.00288 | 0.00572 | 0.0114 |
| 512 | 512 | 0.00382 | 0.00464 | 0.0092 | 0.0183 | 0.0366 |
| 1024 | 1024 | 0.00938 | 0.166 | 0.0331 | 0.0662 | 0.132 |
| 2048 | 2048 | 0.0360 | 0.0702 | 0.140 | 0.279 | 0.559 |
| 4096 | 4096 | 0.154 | 0.302 | 0.603 | 1.21 | 2.41 |
| 8192 | 8192 | 0.715 | 1.375 | 2.660 | 5.31 | 10.6 |

**Table 2.8 Parallel in-core 2-D FFT for CRAY C90 system.**

| Size | | 16 CPUs | | 8 CPUs | | 4 CPUs | | 2 CPUs | | 1 CPU | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Nx | Ny | CPU | Wall | CPU | Wall | CPU | Wall | CPU | Wall | CPU | Wall |
| 16384 | 16384 | 54.4 | 8.86 | 53.3 | 11.90 | 52.8 | 18.30 | | | | |
| 16384 | 8192 | 26.6 | 4.38 | 26.0 | 5.86 | 25.7 | 9.00 | 25.6 | 15.3 | 25.5 | 28.0 |
| 8192 | 8192 | 13.0 | 2.17 | 12.6 | 2.88 | 12.5 | 4.41 | 12.4 | 7.49 | 12.4 | 13.7 |

**Table 2.9 Parallel out-of-core 2-D FFT for CRAY C90 system. (Note: Memory Used: 200 MWords, I/O Device: 1 GW SSD.)**

## LU Results

With the release of UNICOS 6.0, a routine called CGEMMS (Strassen's CGEMM) was implemented in the Cray Scientific Library (Scilib). This routine uses the same algorithm described for CMXMA in LU factorization.

The CGETRFO algorithm is very deterministic. Performance is a function of the speed of the kernel (CGEMMS) and of the speed of the I/O device. A simulation program was constructed that runs through the same slab loops used by CGETRFO. At each stage of the algorithm, the computation time can be determined by computing the number of floating point operations and dividing by the effective computation rate. Also, I/O time can easily be computed since the size

| Size | | | Elapsed Time (seconds) | | | | |
|---|---|---|---|---|---|---|---|
| Nx | Ny | Nz | 16 CPUs | 8 CPUs | 4 CPUs | 2 CPUs | 1 CPU |
| 16 | 16 | 16 | 0.000280 | 0.000240 | 0.000347 | 0.000600 | 0.001110 |
| 32 | 32 | 32 | 0.000587 | 0.000794 | 0.00143 | 0.00275 | 0.005510 |
| 64 | 64 | 64 | 0.00260 | 0.00412 | 0.00805 | 0.0160 | 0.0320 |
| 126 | 128 | 128 | 0.0174 | 0.0336 | 0.0667 | 0.133 | 0.266 |
| 256 | 256 | 256 | 0.160 | 0.293 | 0.585 | 1.17 | 2.34 |
| 512 | 512 | 512 | | 2.76 | 5.58 | 11.1 | 22.1 |

**Table 2.10 Parallel in-core 2-D FFT for CRAY C90 system.**

| Size | | | Number of CPUs | | | |
|---|---|---|---|---|---|---|
| Nx | Ny | Nz | 16 | 8 | 4 | 2 |
| 32 | 32 | 32 | 9.38 | 6.9 | 3.84 | 2.0 |
| 64 | 64 | 64 | 12.3 | 7.76 | 3.97 | 2.0 |
| 128 | 128 | 128 | 15,28 | 7.91 | 3.75 | 2.0 |
| 256 | 256 | 256 | 14.62 | 7.98 | 4.0 | 2.0 |
| 512 | 512 | 512 | | 7.98 | 3.96 | 1.99 |

**Table 2.11 Speedup of in-core 3-D FFT.**

| Size | 8 CPUs | | 16 CPUs | |
|---|---|---|---|---|
| | Time | Eflops | Time | Eflops |
| 128 | 0.001968 | 8524.33 | 0.001060 | 15823 |
| 256 | 0.014467 | 9277.20 | 0.007317 | 18343 |
| 500 | 0.102708 | 9736.30 | 0.049712 | 20116 |
| 512 | 0.101597 | 10568.66 | 0.052114 | 20604 |
| 1000 | 0.719674 | 11116.14 | 0.348659 | 22945 |
| 1024 | 0.712005 | 12064.42 | 0.365284 | 23516 |
| 2047 | 4.987002 | 13779.72 | 2.577519 | 26661 |

**Table 2.12 Performance of CGEMMS (scilib) on CRAY C916. Eflops = Effective Megaflops based on $8N^3$ operations. Time is in seconds.**

| Machine | Size | Mcol | I/O | I/O Time (sec) | Time (hrs) | MFE | D |
|---|---|---|---|---|---|---|---|
| C916/8 | 10000 | 1000 | 10.40 | 0.00 | 0.07 | 10660.17 | SSD |
| C916/8 | 20000 | 1000 | 73.60 | 47.50 | 0.53 | 11220.81 | 16 |
| C916/8 | 30000 | 512 | 449.50 | 121.37 | 1.84 | 10875.14 | 16 |

**Table 2.13** Benchmarks for out-of-core LU decomposition codes. Notes: I/O = Gigabytes Transferred Gigabytes. I/O time is in seconds. Time for solution is in hours. MFE = Effective Megaflops. Mcol = width of column used for slabs D = Disks. 16 DD60s were used with a peak transfer rate of 320 Mbytes/sec. An SSD was used for the 10K result.

| Machine | Size | Mcol | I/O | I/O Time (sec) | Time (hrs) | MFE | D |
|---|---|---|---|---|---|---|---|
| C916/8 | 10000 | 1000 | 10.40 | 12.60 | 0.07 | 10897.75 | 16 |
| C916/8 | 20000 | 1000 | 73.60 | 49.00 | 0.54 | 10992.25 | 16 |
| C916/8 | 30000 | 512 | 449.50 | 134.80 | 1.94 | 10335.51 | 16 |
| C916/8 | 40000 | 512 | 1061.30 | 277.80 | 4.61 | 10286.09 | 16 |
| C916/16 | 10000 | 1000 | 10.40 | 14.80 | 0.04 | 20997.38 | 16 |
| C916/16 | 20000 | 1000 | 73.60 | 59.00 | 0.27 | 21920.75 | 16 |
| C916/16 | 30000 | 512 | 449.50 | 182.00 | 1.01 | 19830.71 | 16 |
| C916/16 | 40000 | 512 | 1061.30 | 386.00 | 2.39 | 19796.41 | 16 |

**Table 2.14 Simulated results for C916 with 16 DD60 Disk Drives.**

of the slab and the I/O rates are known. At each stage of the algorithm, computation time is compared against I/O time and the maximum is used in the time calculations.

The parallelization scheme in CGETRFO was changed slightly. In the original code, parallelization was done inside the matrix multiply kernel. The new version of CGETRFO does square matrix multiplies in parallel as it works down a slab of data. For this reason, actual results tend to be slightly faster than the simulated results.

## Appendix: Mathematical Details for Out-Of-Core FFT Scheme

The defining equation for a one-dimensional DFT is:

$$Y_k = \sum_{j=0}^{N-1} X_j e^{2\pi i(jk/N)} \qquad k = 0, 1, \ldots, N-1 \tag{1}$$

The arrays X and Y are redefined as:

$$X(j) \to X(j_1, j_2) \qquad \begin{array}{l} j_1 = 0, 1, \ldots, N_1 - 1; \\ j_2 = 0, 1, \ldots, N_2 - 1 \end{array} \tag{2}$$

$$j \leftarrow j_1 + j_2 * N_1 \tag{3}$$

$$k \leftarrow k_1 + k_2 * N_1 \tag{4}$$

$$\mathbf{X} = \begin{bmatrix} X_0 & X_{N_1} & \cdots \\ X_1 & X_{N_1+1} & \cdots \\ X_2 & X_{N_1+2} & \cdots \\ \vdots & \vdots & \ddots \\ X_{N_1-1} & X_{2N_1+1} & \cdots \end{bmatrix} \tag{5}$$

$$Y(k) \to Y(k_1, k_2) \qquad \begin{array}{l} k_1 = 0, 1, \ldots, N_1 - 1; \\ k_2 = 0, 1, \ldots, N_2 - 1 \end{array} \tag{6}$$

The new defining equation is:

$$Y(k_2, k_1) = \sum_{j_1=0}^{N_1-1} e^{2\pi i(j_1 k_1/N_1)} \left[ e^{2\pi i(j_1 k_2/N)} \sum_{j_2=0}^{N_2-1} X(j_1, j_2) e^{2\pi i(j_2 k_2/N_2)} \right] \tag{7}$$

In (7), the sum within the square brackets represents a 1-D discrete Fourier transform (DFT) of length $N_2$ that can be computed with a fast Fourier transform (FFT) algorithm. $N_1$ such FFTs are computed, one for each $j_1$.

After the phase factor shown in the square brackets is applied, the first sum of (7) represents another 1-D DFT of length $N_1$, again calculable using an FFT algorithm. These 1-D FFTs of length $N_1$ are computed for each $k_2$, so there are $N_1$ of them.

So, in essence, the problem of computing a single long DFT has been transformed into computing several smaller (approximately $N^{1/2}$) DFTs. Even if the original sequence of length $N$ cannot reside in memory, the smaller FFTs (of length $N_2$ during Pass 1 and length $N_1$ during Pass 2) might, and the larger problem is thus solved. Since multiple FFTs need to be computed simultaneously, "`cfftmlt`" may be invoked. Further, multiple CPUs may be employed to compute these groups of 1-D FFTs.

The input data resides on disk. The sequences generated after the first pass through the data also are put on disk. Finally, the output sequence is also placed on disk. Where a solid-state device (SSD) is available and the size of the data sequence permits, all files can reside on the SSD.

All input/output (I/O) is performed in a raw mode that bypasses user, system and library buffers. Consequently, all I/O is done in multiples of 512-word blocks to utilize the high bandwidth of the devices. In addition, the structure of the data in disk is such that a series of submatrices that make up the full 2-D matrix $X(j_1, j_2)$ are stored, with random access to the subblocks. This "block-oriented" storage and retrieval of the data is crucial to the high performance I/O scheme that has been used.

## Acknowledgments

D. S. Katz and A. Taflove were supported in part for this work by Cray Research, General Dynamics, P.O. 4059045, National Science Foundation Grant ASC-8811273, and Office of Naval Research Contract N00014-88-K-0475.

## References

[1] Rao, S. M., D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propagat.,* **30,** 409–418, 1982.

[2] Newman, E. H., "Polygonal plate modeling," *Electromagnetics,* **10,** 65–83, 1990.

[3] Taflove, A., "Review of the formulation and applications of the finite-difference time-domain method for numerical modeling of electromagnetic wave interactions with arbitrary structures," *Wave Motion,* **10,** 547–582, 1988.

[4] Shankar, V., A. H. Mohammadian, and W. F. Hall, "A time-domain, finite-volume treatment for the Maxwell equations," *Electromagnetics,* **10,** 127–145, 1990.

[5] Fam, A. T., "Efficient complex matrix multiplication," *IEEE Trans. Computers,* bf 37, 877–879, 1988.

[6] Golub, G., and C. F. van Loan, *Matrix Computations,* Johns Hopkins University Press, 1983.

[7] Strassen, V., "Gaussian elimination is not optimal," *Numerische Mathematik,* **13,** 354–356, 1969.

[8] Bailey, D. H., "A high-performance FFT algorithm for vector supercomputers," *International Journal of Supercomputer Applications,* 82–87, 1988.

[9] Ghosh, K. K., "One-dimensional in-core FFTs on Cray Y-MP computer systems," *Technical Memo TM 1,* Benchmarking Department, Cray Research Inc.

[10] *UNICOS User Commands Reference Manual, SR-2011(5.0),* Cray Research Inc., 1989.