Progress In Electromagnetics Research, PIER 42, 247–259, 2003

MPI-BASED PARALLELIZED PRECORRECTED FFT ALGORITHM FOR ANALYZING SCATTERING BY ARBITRARILY SHAPED THREE-DIMENSIONAL OBJECTS

L.-W. Li [†] **and Y.-J. Wang**[‡]

Department of Electrical and Computer Engineering
National University of Singapore
Kent Ridge, Singapore 119260

E.-P. Li

Institute of High Performance Computing
Science Park, Singapore 117528

Abstract—In this paper, how to parallelize the Pre-corrected FFT algorithm for solving the scattering problem of large scale is presented and discussed. The P-FFT technique developed by our group earlier was extended in the current analysis. To show the efficiency of the MPI-based parallelization algorithm, the experiment results are given in the latter part of the paper and various comparisons are made for such a demonstration.

[†] Also with High Performance Computation for Engineered Systems (HPCES) Programme, Singapore-MIT Alliance (SMA), Singapore/USA 119260/02139

[‡] Also with Institute of High Performance Computing, Science Park, Singapore 117528

1. INTRODUCTION

With the growth of the new technologies, the operating frequencies of various electrical and/or electronic systems have been moving toward the higher end of the frequency spectrum. As a result, the scale of electromagnetic problems is becoming much larger, with respect to the resulted smaller wave length. For solving a practical problem of very large scale, the computation of various desired physical quantities is usually very expensive, requiring a large amount of CPU time and computational complexity [1, 2]. Although the capability of a Personal Computer (PC) has been greatly improved, it still cannot meet the actual computational requirement for large scale electromagnetic systems. This paper presents the parallelization of our existing Pre-corrected FFT (P-FFT) algorithm and its implementation for computing three-dimensional electrically large scaled scattering problems on high performance multiprocessor platforms and clusters.

The Pre-corrected FFT algorithm is a kind of fast code originally developed for analyzing a wide variety of electrostatic fields of printed circuits [3, 4] and its best cost is to $O(N \log N)$. Nie-Li-Yuan-Yeo have successfully used the pre-corrected FFT (P-FFT) method to analyze electromagnetic scattering of arbitrarily shaped three-dimensional objects [5, 6]. The code for computing this kind of scattering problems usually runs on a PC. As compared with fast multipole algorithm [1], less execution time and memory requirement are needed in P-FFT approach. It still takes a very long time to get the results, however, when the number of unknowns becomes very large although the pre-corrected FFT is an efficient fast algorithm. We may need a long time to obtain comprehensive results from a normal PC when the current distributions on the surfaces of an airplane are calculated. For example, it takes 9 hours to obtain the results of physical quantities on a Pentium 1G PC if the object has 21240 unknowns [5, 6]. To increase the computational speed, we need to parallelize [7, 8] the existing pre-corrected-FFT algorithm for electrically large systems. In this algorithm, FFT occupies most of CPU time (up to about 75%). So it is important to parallelize the FFT part of the code in order to speed up the pre-corrected FFT algorithm.

At a single incident angle, solving the equation of $ZI = V$ with FFT occupies most of the CPU time when the scattering of an incident plane wave by an arbitrarily shaped scatterer is characterized. In reality, 360 scanning angles are needed so as to obtain a complete distribution of scattering by a common object. Of course, step sizes can be reduced by half when the object is a single-symmetric object such as an airplane, and sometimes even by a quarter when the object double-symmetric (for instance, a dielectric or conducting sphere). In the pre-corrected FFT algorithm, the computational steps are reduced as computing the scattering results at the next incident angle refer to the result of the front incident angle.

Obviously, we can shorten the running time if we can fully utilize the potential capability of a supercomputer or a cluster of computers. In this paper, the way of parallelizing the serial pre-corrected FFT code is given, and the computational time of both serial and parallel algorithms is compared and discussed.

2. FUNDAMENTAL KNOWLEDGE

2.1. The Pre-Corrected FFT Algorithm

Similarly to those of other algorithms such as the fast multipole algorithm (FMM), the main difficulty in P-FFT is also how to approximate the potentials over a large range and how to compute the near-zone and far-field interactions. The basic idea of P-FFT is that uniform grid potentials are used to represent the large distance potentials and directly calculate the nearby interactions. This procedure includes four steps, namely,

- to project the element singularity distributions to point singularities on the uniform grid,
- to compute the fields at the grid points due to the singularities at the grid points using the FFT,
- to interpolate the grid point fields onto the elements, and
- to directly compute nearby interactions.

This procedure is summarized in Figure 1 in terms of steps (1) to (4).

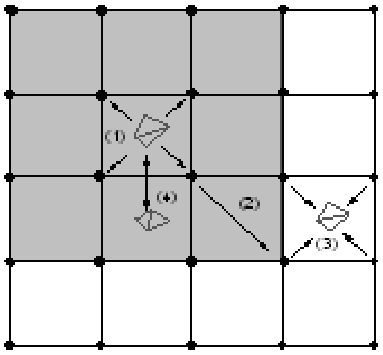


Figure 1. The 3-D representation of the procedures of the pre-corrected FFT algorithm ($p = 2$).

2.2. Parallel 3-D FFT Algorithm

The 3-D Discrete Fourier Transform (DFT) is defined by the following equation [7]:

$$\boldsymbol{X}_{\ell_1,\ell_2,\ell_3} = \sum_{\ell_1}^{N_1-1} \sum_{\ell_2}^{N_2-1} \sum_{\ell_3}^{N_3-1} x_{\ell_1,\ell_2,\ell_3} \omega_{N_1}^{r_1\ell_1} \omega_{N_2}^{r_2\ell_2} \omega_{N_3}^{r_3\ell_3} \tag{1}$$

where $\omega_{N_j}^{r_j\ell_j}$ denotes the twiddle factor and is defined as follows:

$$\omega_N^{r\,\ell} = \exp\left(\frac{2\pi r\ell}{N}\right), \tag{2}$$

while $r_j = 0, 1, 2, \cdots, N_j - 1$ and $\ell_j = 0, 1, 2, \cdots, N_j - 1$ with j being either 1, 2, or 3; and N_1, N_2, and N_3 stand for the dimensions along x-, y-, and z-directions, respectively.

The sequential computation of 3-D DFT can be carried out in the following 3 steps:

- Firstly, form a series of (ordered) 1D-FFTs on the $N_2 \times N_3$ rows (of length N_1 each).
- Secondly, construct a series of (ordered) 1D-FFTs on the $N_1 \times N_3$ rows (of length N_2 each).
- Thirdly, make a series of (ordered) 1D-FFTs on the $N_1 \times N_2$ rows (of length N_3 each).

The computing cost is thus in the $O[N_1N_2N_3\log(N_1N_2N_3)]$.

With the parallel 3-D FFT algorithm, each processor can execute some 1D-FFTs in each step simultaneously. For example, if we assume that there are n processors available, then the $N_3 \times N_2$ rows (of length N_1 each) in first step can be divided by n into n groups of 1D-FFTs. Therefore, each processor does one group of 1D-FFTs independently. In addition, the data of 3D-FFTs should be divided, distributed to each processor before each step, and gathered after each step.

2.3. MPI (Message Passing Interface)

Message Passing Interface (MPI) was proposed as a standard by a broadly based committee of vendors, implementors, and users [9, 10]. Now, it becomes a definition of interfaces among a cluster of computers or the processors of a multiprocessor parallel computer. It provides a platform on which users can reasonably distribute a task to a cluster of computers or the processors of a multiprocessor parallel computer.

The key problem that MPI-based programming relates is how to distribute the tasks of users to processors according to the capability of each processor and reduce the communications among processors as little as possible where the communication time is expressed as follows:

$$\text{Communication time} = \text{Latency} + \frac{\text{Message size}}{\text{Bandwidth}}.$$

There are two main types of MPI-based supercomputers: shared memory and distributed memory (*i.e.*, local memory) machines. For the later kind of computers, reducing the communications is especially crucial as the speed of communication is far slower than that of computation.

One of MPI-based platforms' virtues is that it is easy to write programs on the MPI-based platforms. Only eight functions in MPI library are indispensable. Below shown is the list of these functions [9, 10]:

- `MPI_Init( )` — initializes MPI
- `MPI_Comm_size( )` — finds out how many processors there are;
- `MPI_Comm_rank( )` — finds out which processor it is;
- `MPI_Send( )` — Sends a message;
- `MPI_Recv( )` — Receives a message;
- `MPI_Scatter( )` — distributes distinct messages in the group;
- `MPI_Gather( )` — gathers distinct messages in the group;
- `MPI_Finalize( )` — Terminates MPI.

2.4. The Platform

The parallel code of this work is written in Fortran 90 format and runs on an IBM supercomputer which belongs to IHPC (Institute of High Performance Computing, a National Laboratory in Singapore). The parameters of the IBM supercomputer are listed below:

- 7-node IBM p690 model 681,
- PowerPC_POWER4 CPU 1.3 GHz,
- 32 processors per node,
- 64 GBytes memory per node, and
- AIX 5L version 5.1 operating system

The MPI library is linked into the executable code.

There are mainly four programming modes on parallel computers: `Single Instruction, Single Data Stream (SISD)`, `Single Instruction, Multiple Data Stream (SIMD)`, `Multiple Instruction, Single Data Stream (MISD)`, and `Multiple Instruction, Multiple Data Stream (MIMD)`. In this paper, we only consider the architecture SIMD. That is, we only write one code, then use the same data files, and finally distribute one copy of code and data to each available processor.

3. PARALLELIZED PRE-CORRECTED FFT ALGORITHM

In view of the problem of electromagnetic scattering, parallelization can be carried out in two ways. One is done according to incident angles and the other is parallelization of P-FFT algorithm.

3.1. The First Way of Parallelization

Generally, $180° \times 360°$ scanning need be done to get a complete distribution of scattering for asymmetric objects. Of course, computational expenses can be reduced by half when objects are symmetric, *e.g.*, a plane, and even to one angle when object is a uniform ball. So for a real object, the scattering on an object due to an incident plane wave needs to be characterized from a range of continuous incident angles. The whole incident angles can be divided into n groups by the sum of available processors as equally as possible. Each processor is responsible for the computation of one group.

For example, assuming that there are 10 available same processors which are numbered as $p_0, p_1, \cdots, p_9$ and the angle of incidence in the electromagnetic scattering varies from 0° to 90° at an angle step of 1°, we can get the angle cycles that each processor is allocated:

$$\begin{array}{rl} p_0: & \text{angles from } 0^\circ \text{ to } 8^\circ; \\ p_1: & \text{angles from } 9^\circ \text{ to } 17^\circ; \\ \vdots & \vdots \\ p_9: & \text{angles from } 81^\circ \text{ to } 90^\circ. \end{array}$$

For $p_0, p_1, \cdots, p_9$, the scattering cross sections at angles of $0^\circ, 9^\circ, \cdots, 81^\circ$ at a step of 9°, respectively, need be computed independently and simultaneously. Then the scattering at the other angles in each group is achieved dependently on the above result of the same group.

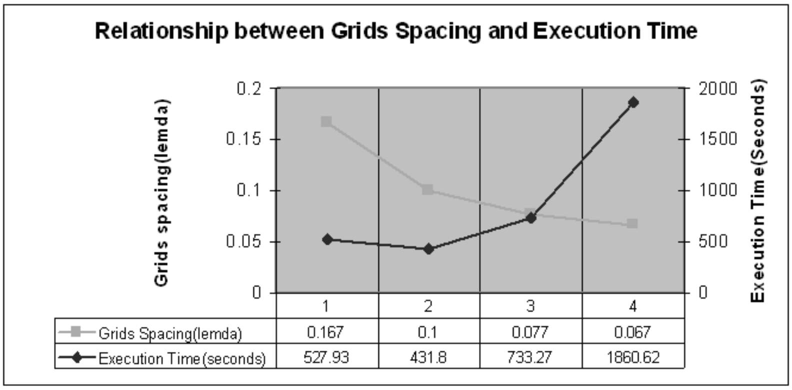
Pay attention to that p_9 needs to operate an angle scanning for 10 times while the other processors only do 9 times. This is because there are totally 91 angles and they cannot be equally divided by 10. So the final running time relies on the operation of p_9. In this layer of parallelism, the computational results from all processors are collected with the function `MPI_GATHER(•)` provided in MPI library, and written into a file in hard disk for future use.

When the parallelization of the first layer is implemented, there exists one problem that must be seriously treated. Generally for arbitrarily shaped three-dimensional objects, the convergence speed of convolution of scattering at an angle of incidence is different from that at another angle of incidence. The convergence speed is therefore related to the shapes of objects. If the speeds have much difference from each other, the algorithm should be improved according to the following method. We allocate first n incident angles to n processors one by one, then next n angles to n processors again, and etc., until the end of line where incident angles are reached.

Assume that in terms of running time, a fraction p of a code can be parallelized and that the remaining $1-p$ cannot be parallelized. According to Amdahl's law, the parallel running time required will be $1-p+p/n$ of the serial running time in the ideal situation if there are n processors available.

3.2. The Second Way of Parallelization

Theoretically, each of four steps in the pre-corrected FFT algorithm can be parallelly executed. However, the statistics of the execution time of each step shows that the third step (interpolating grid potentials) and the fourth step (correction) occupy most of CPU



	1	2	3	4
Grids Spacing(lemda)	0.167	0.1	0.077	0.067
Execution Time(seconds)	527.93	431.8	733.27	1860.62

Figure 2. Relationship between grids spacing and execution time.

time, about 10–30% and 40–60%, respectively. Although the P-FFT algorithm reduces the convolution computation by using coarse grids, Fast Fourier Transforms (FFTs) still cost much time. On the other hand, more corrections need to be done in order to obtain high enough accuracy when the grids spacing becomes larger. Although the P-FFT approach can use coarser grids in order to reduce FFT execution time which used to be long, it does not mean that the bigger the grids spacing is, the less the execution time. When the grids spacing increases, the computational time of precorrection also increases because the threshold of nearby area becomes bigger. There is a balance between nearby correction area and the grids spacing. The following example shows in Fig. 2 the relationship between these two. Only one processor is used to compute scattering effects of a sphere whose radius is 1 meter. The wavelength is set to be 1 meter for obtaining all the dimensions in wavelength. The surface of the sphere is divided into 3692 elements, with 5538 unknowns and 1848 nodes.

Let the variable rank represent the number of processors and the first processor in a group of processors is numbered as p_0 while the other processors in this group are numbered as p_1, p_2, $\cdots$, p_n, respectively. Then the algorithm of the second way can be described as follows using pseudo code (by scanning a 3-D object):

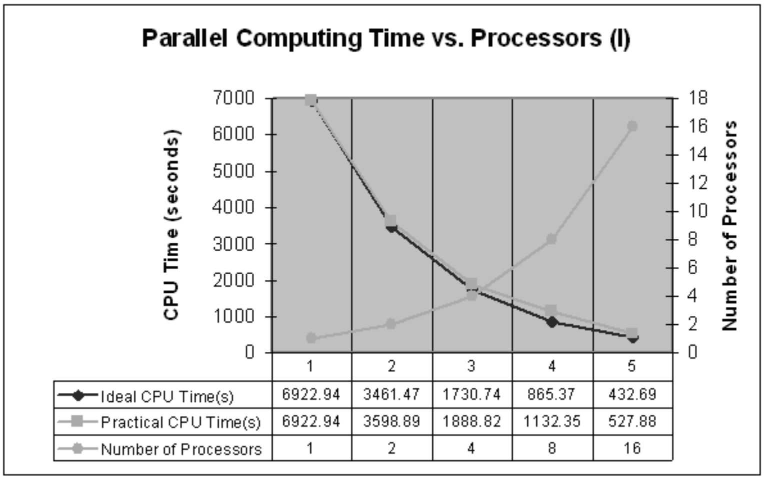
```
IF (rank .eq. $p_0$) THEN
!Project the panel charges to the grid charges
CALL MPI_Scatter( ) !scatter data from $p_0$ to $p_0 - p_n$
ENDIF

Project the panel potentials to the uniform grids !  $p_0$ to
```

```
$p_0 - p_n$

!start to compute convolution

! $p_0 - p_n$, compute FFT
CALL 1-D FFT( ) for $m$ times !  along axis $x$
CALL 1-D FFT( ) for $n$ times !  along axis $y$
CALL 1-D FFT( ) for $k$ times !  along axis $z$

! $p_0 - p_n$, compute FFT$^{-1}$
CALL 1-D FFT( ) for $m$ times !  along axis $x$
CALL 1-D FFT( ) for $n$ times !  along axis $y$
CALL 1-D FFT( ) for $k$ times !  along axis $z$

!end of convolution

Interpolate the grid potentials to the panels ! $p_0$ to $p_0 - p_n$

Correction(Compute nearby interactions) !  $p_0$ to $p_0 - p_n$

IF (rank .eq. $p_0$) THEN
CALL MPI_Gather( ) !gather data from $p_0 - p_n$ to $p_0$
ENDIF
```

4. THE EXPERIMENTAL RESULTS

4.1. Parallelization of the First Way

In this way, the electromagnetic scattering by a metallic spheric model is considered and its scattering cross sections are computed. The wavelength λ is set to be 1 meter, so all the other dimensions are given in terms of the wavelength. The surface of the sphere whose radius is 1 meter is divided into 3692 elements (also called triangles), 5538 unknowns and 1848 nodes. The grids spacing is set to be 0.167λ as P-FFT can use less number of grids (generally of size ranging from 0.15λ to 0.3λ) than AIM in order to reach good enough accuracy [1, 5, 6]. So the grids that are used to represent the long-range potentials are $28\lambda \times 28\lambda \times 28\lambda$. Obviously, only one incident angle needs to be computed for a sphere. However, 16 angles from 180° to 165° are utilized in computation in order to show the effectiveness of the parallel algorithm. Fig. 3 shows the experimental results of parallelization of the first layer. In each column, the value of CPU Time corresponds to the number of processors used in an experiment. Here, the ideal CPU



	1	2	3	4	5
◆ Ideal CPU Time(s)	6922.94	3461.47	1730.74	865.37	432.69
■ Practical CPU Time(s)	6922.94	3598.89	1888.82	1132.35	527.88
● Number of Processors	1	2	4	8	16

Figure 3. Parallel computing time.

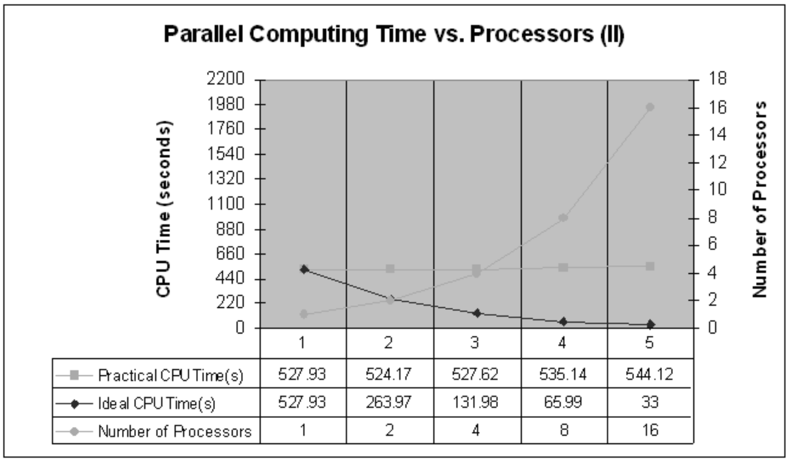
time is equal to the value of practical CPU time of a single processor divided by the number of multiple processors.

4.2. Parallelization of the Second Way (Only Parallelizing FFT)

As a second way of parallelization, we will parallelize the FFT first. Only one angle of 180° is computed in the second way of parallelization. Figure 4 shows the experiment result of parallelization of the second way. The grids spacing is set to be 0.167λ.

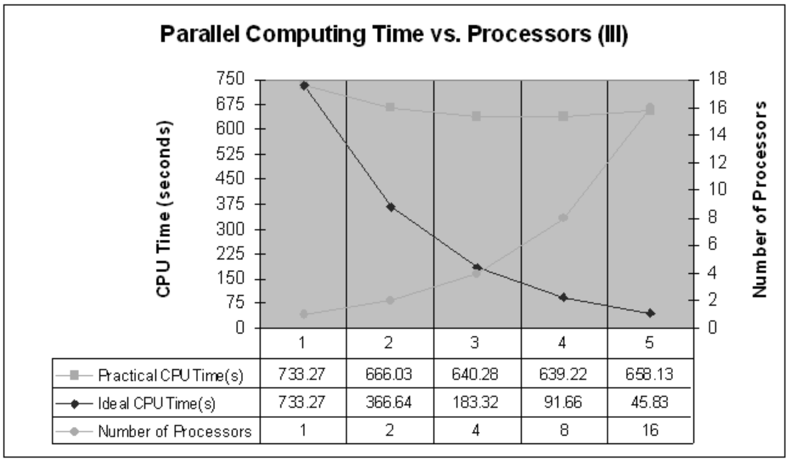
From Fig. 4, it is depicted that the efficiency is not improved. There are two factors contributing to this result. One is that FFT computation generally occupies 10 to 30 percent of total running time with appropriate grids spacing. The other is that the resultant computational time of FFT is small as compared with the communication time between processors. When the number of processors increases, the communication time also extends. So when the number of processors reaches 8, it spends more time than that with 1 processor.

Shown in Fig. 5 is another example. The grids spacing is set to be 0.077λ. It shows in Fig. 5 that the parallel FFT has good efficiency of reducing the execution time. This is because the FFT dimension changes from $28\lambda \times 28\lambda \times 28\lambda$ to $56\lambda \times 56\lambda \times 56\lambda$. The execution time of FFT is obviously far more than the time spent on communications between processors.



	1	2	3	4	5
■ Practical CPU Time(s)	527.93	524.17	527.62	535.14	544.12
◆ Ideal CPU Time(s)	527.93	263.97	131.98	65.99	33
● Number of Processors	1	2	4	8	16

Figure 4. Parallel computing time.

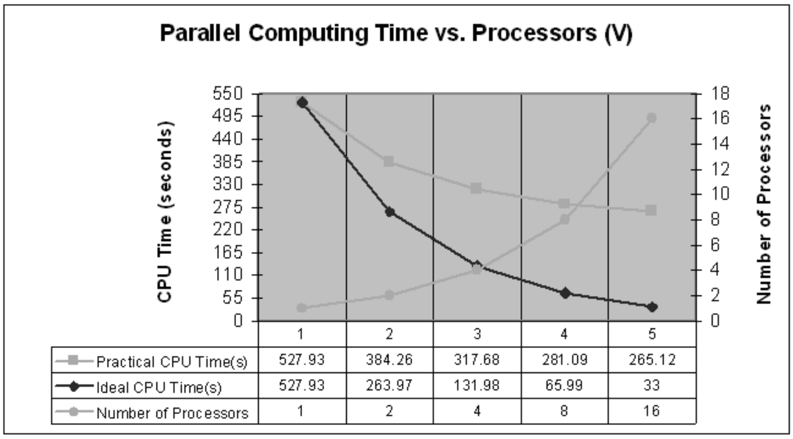


	1	2	3	4	5
■ Practical CPU Time(s)	733.27	666.03	640.28	639.22	658.13
◆ Ideal CPU Time(s)	733.27	366.64	183.32	91.66	45.83
● Number of Processors	1	2	4	8	16

Figure 5. Parallel computing time.

4.3. Parallelization of the Second Way (Parallelizing Correction and FFT)

As the last approach, we tried to consider an alternative way for the parallelization by parallelizing both correction and FFT in the P-FFT method. Again, only one angle of 180° is computed and the grids spacing is set to be 0.167λ. The corresponding results are provided and shown in Fig. 6.



	1	2	3	4	5
Practical CPU Time(s)	527.93	384.26	317.68	281.09	265.12
Ideal CPU Time(s)	527.93	263.97	131.98	65.99	33
Number of Processors	1	2	4	8	16

Figure 6. Parallel computing time.

5. CONCLUSION

In this work, we proposed several ways of making parallel codes of P-FFT algorithms. From the present analysis, it is found that the actual CPU time after parallelization can be close to our expected ideal time. Also, it shows that the parallelization of the first way is successful. Figures. 4-6 provide, however, dissatisfactory results to us as the real time is much longer that the expected time. There might be two aspects that incur this result. First, FFT and correction occupy approximately 10%-30% and 40%-60% of the total computation time respectively. So only part of the whole computation is parallelized in the second way while the whole part of the computation is parallelized in the first way. Second, there are more communications between processors in the second way. This contributes to extra execution time. The experimental results show that the running time is shortened greatly after the computation is parallelized. This proves that the algorithm proposed in this paper is efficient and can be utilized to reduce the CPU time and our other computational efforts.

ACKNOWLEDGMENT

The authors are grateful to Dr. Xiao-Chun Nie in Temasek Laboratories, and Mr. Wei-Bin Ewe in Dept. of Electrical and Computer Engineering both at National University of Singapore for his useful discussion and suggestions. This work is partially sponsored by a joint project of Temasek Laboratories and Institute of High Performance Computing, Singapore.

REFERENCES

1. Chew, W. C., J.-M. Jin, E. Michielssen, and J. Song, *Fast And Efficient Algorithms In Computational Electromagnetics*, Artech House, Norwood, MA, 2001.
2. Umashankar, K. and A. Taflove, *Computational Electromagnetics*, Artech House, Norwood, MA, 1993.
3. Phillips, J. R. and J. K. White, "A pre-corrected FFT method for electrostatic analysis of complicated 3-D structures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 10, 1059–1072, Oct. 1997.
4. Aluru, N. R., V. B. Nadkarni, and J. White, "A parallel precorrected FFT based capacitance extraction program for signal integrity analysis," *Proc. of 33rd Design Automation Conference*, DAC 96-06/96 Las Vegas, NV, USA.
5. Nie, X., L.-W. Li, N. Yuan, and Y. T. Soon, "Pre-corrected FFT algorithm for solving combined field integral equations in electromagnetic scattering," *Journal of Electromagnetic Waves and Applications*, Vol. 16, No. 8, 1171–1187, 2002.
6. Nie, X., L.-W. Li, and N. Yuan, "Fast analysis of scattering by arbitrarily shaped three-dimensional objects using the pre-corrected FFT method," *Microwave and Optical Technology Letters*, September 20, 2002.
7. Chu, E. and A. George, *Inside The FFT Black Box : Serial And Parallel Fast Fourier Transform Algorithms*, CRC Press, Boca Raton, Fla., 2000.
8. Nussbaumer, H. J., *Fast Fourier Transform And Convolution Algorithms*, Springer-Verlag, New York, 1981.
9. Guiffaut, C. and K. Mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas and Propagation Magazine*, Vol. 43, No. 2, 94–103, April 2001.
10. Gropp, W., E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming With The Message-Passing Interface*, 2nd ed., MIT Press, Cambridge, MA, 1999.