1

INTRODUCTION

1.0 SUMMARY

This chapter begins with a brief review of the several widely used numerical methods utilized in the field of parallel computational electromagnetics (CEM). This book is intended to provide a parallel frequency-domain solution to radiation and scattering for electromagnetic (EM) field problems. It is important to ensure that the parallel EM codes developed are compatible with all the typical computer platforms that are currently available. Therefore, computers with different architectures and operating systems have been used as benchmark platforms in this book. A brief description of the hardware used for this work is provided for easy reference. ScaLAPACK and PLAPACK, the two software library packages that have been employed for implementing parallel computations, are also introduced in this chapter.

1.1 A BRIEF REVIEW OF PARALLEL CEM

1.1.1 Computational Electromagnetics

Electromagnetic devices and systems, ranging from everyday office appliances to cellular phones, have become an integral part of modern life. The continued development of new technology greatly depends on the engineering analysis and synthesis of electromagnetic systems. These are based on obtaining accurate solutions of Maxwell's equations for the system of interest. The increase of research and development in a wide variety of applications, including antenna design, microwave circuits, photonics, ray tracing, wireless communication, electromagnetic compatibility/interference (EMC/EMI), and so on, have led to systems becoming more and more complex. In many cases, a single device has become a very complicated structure that includes a number of conductors, dielectric, and semiconductors of arbitrary shapes and of a complex physical nature. The expensive fabrication technologies preclude the possibility of modifying a device if its performance is not within the specifications of the designer. Therefore, methods capable of extremely accurate characterization of systems are required to analyze the structures of interest. It is challenging to obtain such methods because of the complex nature of any of the modern electromagnetic devices. The analysis of these devices does not lead to closed-form expressions.

Analytical solutions in closed form are known for only a limited number of special cases, which are hardly ever directly applicable to real-world applications. The failure to derive closed form solutions from Maxwell's equations for real-world applications has led to intensive research on numerical techniques. The application of numerical methods to electromagnetic field problems is known as *computational electromagnetics* (CEM). It is a branch of electromagnetics that deals with computational methods and is a natural extension of the analytical approach to solving the Maxwell equations [1,2].

For a student of engineering electromagnetics or a researcher just starting to work in this area, few textbooks can furnish initial guidance to CEM. So, a short bibliography of the most widely used numerical methods is reviewed in this chapter. Among the methods to be described, the method of moments (MoM) is considered to be one of the most popular integral equation–based numerical methods. It coincides formally with the weighted-residual method because the sources (unknowns) are expanded by a sum of certain basis functions multiplied by unknown coefficients. The residual of the integral equation is weighted using a suitable inner product and a set of weighting functions. This results in a set of linear equations, which can be solved in the usual way. Regarding MoM, R. F. Harrington worked out a systematic, functional-space description of electromagnetic problems, starting from integral equations and using the reaction concept. Harrington summarized his work in a book published in 1968, which is still considered to be a classic textbook even today [3]. For analysis of radiation from conducting structures, it is difficult to surpass the accuracy and efficiency of an integral equation–based computational methodology. For composite structures containing both conductors and dielectrics, where the dielectric inhomogenity is not severe, the integral equation methodology is still an excellent numerical procedure. Besides MoM, a number of variations of the integral equation–based numerical solution procedure have been developed. For example, the *fast multipole method* (FMM) and *multilevel fast multipole algorithm* (MLFMA) have been proposed to accelerate the matrix–vector product that arises in the iterative solution of the MoM equations [4].

The *finite element method* (FEM) is a numerical method that is used to solve boundary-value problems characterized by a partial differential equation and a set of boundary conditions. The first book completely committed to finite elements in electromagnetics was by P. P. Silvester and R. L. Ferrari and was published in 1983 [5]. The introductory text by J. M. Jin in 1993 was particularly directed to scattering and antenna problems [6]. The book by M. Salazar-Palma et al. [7] introduced an iterative and self-adaptive finite-element technique for electromagnetic modeling in 1998, with a review of the history of numerical methods in electromagnetics and a categorization of these methods as presented in the first chapter [7]. A selected bibliography for FEM in microwaves was also provided by R. Coccioli et al. [8].

The *finite-difference time-domain* (FDTD) method is a finite-difference solution for electromagnetic problems. The method was developed by K. S. Yee in 1966 [9]. With the advent of low-cost, powerful computers and improvement of the method itself, FDTD has become one of the most widely used methods for solving Maxwell's equations for scientific and engineering applications. K. L. Shlager and J. B. Schneider published a selective survey of the FDTD literature in 1995 [10]. In the same year, A. Taflove authored a comprehensive textbook that he and S. C. Hagness subsequently expanded and updated to a third edition in 2005 [11].

MoM, FEM, and FDTD, the methods mentioned so far, are the most frequently used algorithms in CEM. Many more books are also available for these and other numerical methods, which will not be discussed further.

As S. M. Rao states: "CEM has evolved rapidly during the past decade to a point where extremely accurate predictions can be made for very general scattering and antenna structures" [12]. Numerical simulation technology is the most cost-effective means of meeting many technical challenges in the areas of electromagnetic signature processing, antenna design, electromagnetic coupling, microwave device design and assembly, and so on. With the rapid increase in the performance of desktop computers, it has become feasible for designers to apply computational electromagnetics to determine the electromagnetic behavior of many systems. However, when performing an electromagnetic simulation associated with a large-scale configuration, it is difficult to achieve a high level of accuracy when using conventional computing platforms. The numerical capability of CEM can be enhanced substantially by using scalable parallel computer systems, which will be discussed next.

1.1.2 Parallel Computation in Electromagnetics

Large-scale CEM simulations have encountered computational limitations in terms of physical memory space and central processing unit (CPU) time of the computer. Parallel computing on clusters and/or computers with multicore processors continues to be the method of choice for addressing modern engineering and scientific challenges that arise from the extremely complicated real-life applications.

Parallel computing is a form of computational methodology in which many instructions are carried out simultaneously [13]. Parallel computing operates on the principle that large problems can almost always be divided into smaller ones, which may be carried out concurrently ("in parallel"). In the simplest sense, parallel computing is the simultaneous use of multiple computational resources to solve a numerical problem. In parallel computing, a problem is broken up into discrete parts that can be solved concurrently, and each part is further broken down into a series of instructions, and the instructions from each part are executed simultaneously on different CPUs (or cores). A very good tutorial on the topic of parallel computing is provided online [14].

The primary reasons for using parallel computing are that it saves time (wall clock time), solves larger problems, and provides concurrency (multiple tasks at the same time). A Lawrence Livermore Laboratory report [14] also points out additional reasons such as taking advantage of nonlocal computer resources available on a wide area network, or even the Internet when local resources are scarce. Also, cost saving can be a consideration for using multiple "cheap" computing resources instead of buying computational time on a supercomputer. Another reason for using parallel computing is to overcome storage constraints. For large-scale problems, using the memory and hard disk of multiple computers may overcome the limited resources of a single computer.

Parallel computing has been used for many years, mainly in high performance computing (HPC), but interest has been growing more recently mainly because of physical constraints preventing high-frequency scaling. Parallelism has become the dominant paradigm in computer architecture, mainly in the form of multicore processors. Since 2002 or so, the trends indicated by ever faster networks, distributed systems, multicore CPUs, and multi-processor computer architectures clearly show that parallelism is the future of computing [14].

The last two decades (1990s–2000s) witnessed significant developments in both computer hardware capabilities and implementations of fast and efficient algorithms. The performance of parallel computations on any computer system is closely tied to communication and latency. These factors, particular to each individual system, are introduced by the communication protocol and operating system implementation, both of which have profound influence on the performance of a parallel code. Therefore, a judicious tradeoff between a balanced workload and interprocessor communication is needed to efficiently use distributed-memory, multinode computers. Such a need is intrinsically related to the numerical algorithms and hardware architectures. A synergism of the relatively new numerical procedures and high-performance parallel computing capability opens up a new frontier in electromagnetics research [15].

Table 1.1 lists some selected publications on parallel CEM code development since 1998, from which one can see that parallel computing techniques have penetrated into mainstream numerical methods. Generally speaking, CEM codes are currently run on all typical computer platforms. Note that the works cited in Table 1.1 do not indicate the evolutionary process in parallel CEM, but only show that research has touched on different CEM methods to date.

The subject of reviewing parallel CEM is too broad to be covered extensively in several pages, and therefore only a synoptic view is given here. Even though the topic of this book is parallel frequency-domain MoM, parallel implementations of the other most widely used frequency-domain and time-domain numerical methods, like FEM and FDTD, are also briefly introduced here to provide an overview on the subject.

TABLE 1.1. Several Publications on Parallel CEM

Method	Research Topic	Year
	Frequency Domain	
MoM	A parallel implementation of NEC for the analysis of large structures [16]	2003
	Highly efficient parallel schemes using out-of-core solver for MoM [17]	2007
FMM	10 million unknowns: Is it that big? [18]	2003
	Massively parallel fast multipole method solutions of large electromagnetic scattering problems [19]	2007
FEM	Open-region, electromagnetic finite-element scattering calculations in anisotropic media on parallel computers [20]	1999
	Time Domain	
TD-FEM	Implementing a finite-element time-domain program in parallel [21]	2000
FDTD	A parallel FDTD algorithm using the MPI library [22]	2001
	Study on the optimum virtual topology for MPI-based parallel conformal FDTD algorithm on PC clusters [23]	2005
	A robust parallel conformal finite-difference time-domain processing package using the MPI [24]	2005
TDIE	A parallel marching-on-in-time solver [25]	2008

MoM is the most well known technique among integral equation (IE)-based methods. As the most popular frequency-domain integral equation method, MoM is an extremely powerful and versatile numerical methodology for discretizing an integral equation to a matrix equation. Because of the many advantages of MoM, the parallel implementation of the method itself, the hybrid methods and fast algorithms related to MoM, have all been significant research topics during the more recent decades and will continue to be popular for years to come.

The parallel MoM code was developed at the beginning of the 1990s [26–28]. J. E. Patterson programmed and executed the numerical electromagnetics code (NEC) [29] in a parallel processing environment, which was developed at Lawrence Livermore National Laboratory, in 1990. T. Cwik, J. Partee, and J. E. Patterson used MoM to solve scattering problems in parallel in 1991. Later, working with Robert A. van de Geijn, the author of PLAPACK [30], T. Cwik, developed a parallel MoM code using the Rao–Wilton–Glisson (RWG) basis function in 1994 [31,32]. In 1998, parallel MoM employing the RWG basis functions, using ScaLAPACK library package [33] as the solver, was implemented for the Cray T3E system [34].

Since the mid-1990s, research has been carried out on parallel implementations of standard production-level MoM codes. One of the frequently studied parallel implementations of existing codes is the widely used NEC. Successfully implemented in 2003, parallel NEC is portable to any platform that supports message-passing parallel environments such as the *message passing interface* (MPI) [35] and the *parallel virtual machine* (PVM) [36]. The code could even be executed on heterogeneous clusters of computers programmed with different operating systems [16].

The bottleneck of this traditional parallel MoM partly comes from the memory storage requirements. One remedy to overcome this disadvantage is to parallelize the hybrid of MoM and such high-frequency methods as *uniform geometrical theory of diffraction* (UTD) [37], *physical optics* (PO) [38], and so on. For example, the hybrid MoM-UTD method, which greatly extends the capability of MoM, is implemented in SuperNEC [39], and is then parallelized. The parallel iterative MoM and PO hybrid solver for arbitrary surfaces using the RWG basis functions also has been implemented [40,41].

Fast algorithms can also be employed to reduce the overall memory and time requirements. Some of the frequently used fast algorithms are the conjugate gradient (CG) fast Fourier transform (CG-FFT) [42], adaptive integral method (AIM) [43], fast multipole method (FMM) [44], and precorrected FFT [45]. The FMM, arguably the most popular of these methods, was even further improved via the multilevel fast multipole algorithm (MLFMA) to further improve the time taken for calculation of a matrix–vector product [46,47]. While difficult to implement, the MLFMA has become the algorithm of choice when solving large-scale scattering problems arising in electromagnetics. Therefore, the parallelization of MLFMA on distributed memory computing clusters [48–56] and shared memory multiprocessors [57] has always been a topic of great interest.

While some success has been demonstrated in parallelizing MLFMA in distributed memory environments [49,50,55,56], it requires sophisticated load distribution strategies involving shared and distributed partitions of the MLFMA tree. Thus, parallel scaling of the MLFMA algorithm has been limited to a handful of processors as addressed in [19]. An FFT extension of the conventional FMM, known as FMM-FFT, lowers the matrix–vector multiplication time requirement of the conventional algorithm, while preserving the propensity for parallel scaling of the single-level FMM (since it does not employ the tree structure of MLFMA). The research published in 2007 has demonstrated that a parallel FMM-FFT algorithm is quite attractive (when compared to the MLFMA) in the context of massively parallel distributed-memory machines [19].

These methods have been implemented mainly in commercially available software [58]. However, the use of hybrid methods and fast algorithms sacrifice accuracy to accommodate the solution of large problems. Also, the disadvantage of the MoM with subdomain basis functions, which is implemented in NEC or in commercial software, is that it generates much more unknowns than the MoM formulation using the entire-domain or larger subdomain basis functions, thus leading to large memory requirements. Although 64-bit computers these days theoretically have "huge" memory capacity, the development and applications of in-core solvers are limited by the amount of RAM available. Using virtual memory of the computer is not a good alternative because the significant degradation in performance results in increased time for generating a solution.

On the other hand, as an example, a 10-node Dell 1855 cluster equipped with 20 CPUs and 80 gigabytes (GB) of memory could complete a problem that occupies all the available memory using an in-core solver for a dense linear matrix equation in less than 3 hours. This suggests that the processing power of high-performance machines is underutilized and much larger problems can be tackled before runtimes become prohibitively long.

For this reason, one optional method to overcome the memory storage bottleneck of MoM while maintaining its accuracy is using an out-of-core solver, rather than using hybrid methods or fast algorithms, which may incur accuracy problems in some cases, e.g., coupling analysis during antenna design. Since the matrix generated by MoM is a full dense matrix, the LU decomposition method used to solve the matrix equation is computationally intensive when compared with the read/write process of the matrix elements from/into the hard disk [59]. Therefore, it is natural to introduce an out-of-core solver to tackle large, dense, linear systems generated using the MoM formulation. Furthermore, to solve the large problem out-of-core as quickly as possible, the code can be designed in a parallel way to run on a cluster.

Another way to overcome the bottleneck in storing the matrix elements in the memory is to apply the higher-order basis functions defined over a large domain to MoM [60-62] as this requires fewer numbers of unknowns (by a factor of ≥ 10 involving canonical planar structures). In two publications in 2007, the authors presented a parallel in-core implementation of MoM using higher-order basis functions [63,64], and the details of this methodology will be further discussed in this book along with the parallel out-of-core implementation [65]. The most important difference between the parallel in-core solver and the parallel out-of-core solver is that the latter has the ability to break the random access memory (RAM) restriction and thus can go far beyond the limit of the physical memory in any computer or cluster to solve a large MoM matrix equation within a reasonable amount of time.

Besides MoM, another well-established frequency-domain technique is the finite element method (FEM), which is an effective means for analyzing electromagnetic problems. The principal attribute of FEM is that it efficiently models highly irregular geometries as well as penetrable and inhomogeneous material media. Publications on parallel implementations of FEM on supercomputers, workstations, and PCs during the period of 1985–1995 were summarized in a paper published in 1996 [66]. In a massively parallel environment, traditional sequential algorithms will not necessarily scale and may lead to very poor utilization of the architecture associated with the multiprocessor. A domain decomposition method based on the finite-element tearing and interconnecting (FETI) algorithm was considered to be more scalable and efficient on parallel platforms than traditional iterative methods such as a preconditioned conjugate gradient algorithm when solving large matrices [67].

Research on parallel implementation of time-domain numerical methods has also been of interest for many years, accompanied by the development in computer technology. Two popular time-domain methods in the analysis of electromagnetic phenomena are the FDTD and FETD (finite-element time-domain) schemes. FDTD is, by nature, essentially data-parallel. A. Taflove et al. started the effort in 1988 on a parallel implementation of FDTD on CM-1 and CM-2 supercomputers [68]. Parallelization of the FDTD method using distributed computing was done in 1994 by V. Varadarajan and R. Mittra [69], who suggested rules and tolerances for implementation on a cluster of computers. They used the parallel virtual machine (PVM) message-passing protocol over TCP/IP on a cluster of eight HP computers. Since then, many problems have been solved using FDTD on various hardware platforms, processor configurations, and software. Z. M. Liu et al. implemented parallel FDTD on a CM-5 parallel computer in 1995 [70]. A. D. Tinniswood et al. ran parallel FDTD on 128-node IBM SP-2 cluster later in 1996 [71]. Note that the reported maximum speedup factor from different publications varies significantly and is not even a relative measure of the performance of the underlying algorithm. The works described so far have been restricted by the available technology, software systems, or factors under examination, as well as differences in the size of the domain of the selected problem [72].

With the development of MPI programming technology, a parallel FDTD implementation using the MPI Cartesian 2D topology to simplify and accelerate the algorithm was presented in 2001 and it allowed noncontiguous locations in memory to be associated with the data type [22]. Furthermore, suggestions on improved domain partitioning conditions based on the estimated processing time for different types of media [dispersive, perfectly matched layer (PML), and other boundary conditions] were given, and good performance results have been obtained. These aspects appear to be evolutionary for the FDTD computation for the following reasons. MPI is a widely embraced standard; the operations occur only at initialization; the methods are scalable and extensible; and there is no impact due to runtime load balancing to yield an improvement in overall performance. In the same year, an MPI Cartesian 3D topology was used for parallel FDTD, and different MPI communication patterns were investigated [73]. Later, the optimum MPI topology for 3D parallel FDTD was studied on a PC cluster [23,74,75]. The influence of different virtual topology schemes on the performance of a parallel FDTD code was discussed in detail, and the general rules were presented on how to obtain the highest efficiency of the parallel FDTD algorithm by optimizing the MPI virtual topology. In 2005, an MPI-based parallel conformal FDTD package was developed with a friendly graphical user interface (GUI) [24]. Parallel implementation of FDTD using MPI is continuing to find even wider use today.

The parallel *finite-element time-domain* (FETD) package using the MPI message-passing standard was developed in 2000 [76]. Different mathematical backgrounds of FETD and FDTD methods have led to different properties. The numerical performance of the distributed implementations of the FDTD and FETD methods was compared in [77] with respect to scalability, load balancing, and speedup.

Another promising time-domain method used to analyze the scattering and radiation from arbitrary structures is the time-domain integral equation (TDIE) based marching-on-in-time (MOT) method or marching-on-in-degree (MOD) method [78,79]. Note that the MOT method may sometimes suffer from its late-time instability. This can be overcome by using the associated Laguerre polynomials for the temporal variation. As such, the time derivatives can be handled analytically and stable numerical results can be obtained even for late times in the MOD method. A parallel time-domain simulator for analyzing the electromagnetic scattering and radiation phenomena has been developed by the authors of this book and partly presented in a previous reference [25].

To summarize, parallel CEM has been a fervently pursued research topic, and what was reviewed in the preceding paragraphs provide just a glimpse of its development. There is much room left for future study and improvement, specifically in the parallel out-of-core implementation of MoM.

In the next section, the computer hardware and software that supports the research on parallel CEM presented in this book are introduced.

1.2 COMPUTER PLATFORMS ACCESSED IN THIS BOOK

Current computers generally have a single processor and use the IA-32 (Intel architecture, 32-bit) [80] instruction set, which is one of the instruction sets of Intel's most successful microprocessor generically termed x86-32. IA-32 is the 32-bit extension of the original Intel x86 processor architecture that defines the instruction set installed in most personal computers (PC) in the world. The limitation of the 32-bit architecture is that it can address only 2^{32} bits at most. This restricts the size of the variable to be stored in the memory, which cannot exceed 4 GB. A typical 2 GB of RAM can store a matrix size of approximately 15,000×15,000 when the variable is defined in single-precision (complex) arithmetic or a matrix size of approximately 11,000×11,000 when using double-precision (complex) arithmetic. In other words, one cannot deal with an integral equation solver in a MoM context if the number of unknowns exceeds 15,000 for single precision and 11,000 for double precision using LAPACK [81] for serial codes or ScaLAPACK [82]/PLAPACK [83] for parallel codes.

The IA-32 structure was extended by Advanced Micro Devices (AMD) Corporation in 2003 to 64 bits. The first family of processors that AMD built was the AMD K8 processors, which AMD subsequently named AMD64. This was the first time any company other than Intel Corporation had any significant additions to the 32-bit architecture. Intel Corporation was forced to do something, and they came up with the IA-32e, or the NetBurst family of processors. Later, Intel called them EM64T (extended-memory 64-bit technology). These 64-bit AMD and Intel processors are backward-compatible with the 32-bit code without any loss in the performance.

In addition, there is the IA-64 (Intel architecture, 64-bit), which is a true 64-bit processor architecture developed cooperatively by Intel and Hewlett-Packard and was implemented in the Itanium and the Itanium 2 processors. A 64-bit computer can theoretically address up to $2^{64} = 16.8$ million terabytes (TB) directly, which is sufficient to store a $10^9\times10^9$ matrix in single precision or a $0.7\times10^9\times0.7\times10^9$ matrix in double precision, provided enough physical memory and virtual memory are available.

While manufacturing technology continues to improve, breaking the physical limits of semiconductor-based microelectronics has become a major design concern. Some effects of these physical limitations can cause significant heat dissipation and data synchronization problems. The demand for more capable microprocessors causes CPU designers to try various methods of increasing performance. Some instruction-level parallelism (ILP) methods, like superscalar pipelining, are suitable for many applications, but are inefficient for others that tend to contain a difficult-to-predict code. Many applications are better suited for thread-level parallelism (TLP) methods. Utilizing multiple independent CPUs is one common method used to increase a system's overall TLP. A combination of increased available space due to refined manufacturing processes and the demand for increased TLP is the logic behind the creation of multicore CPUs [84].

A multicore CPU [or chip-level multiprocessor (CMP)] combines two or more independent cores into a single package composed of a single integrated circuit (IC), called a *die*, or more dies packaged together. A dual-core processor contains two cores, and a quad-core processor contains four cores. A multicore processor implements multiprocessing in a single physical package. Cores in a multicore device may share a single coherent cache at the highest on-device cache level (e.g., L2 for the Intel core 2) or have separate caches (e.g., current AMD dual-core processors). The processors also share the same interconnect to the rest of the system, like the L2 cache and the interface to the frontside bus (FSB). Each core independently implements optimizations such as superscalar execution, pipelining, and multithreading. Software benefits from multicore architectures where a code can be executed in parallel. Multicore processors can deliver significant benefits in performance for multithreaded software by adding processing power with minimal latency, given the proximity of the processors.

The general trend in processor development has been from multicore to many-core. AMD was the first x86 processor manufacturer to demonstrate a fully functioning dual-core processor on a shipping platform [85]. Intel built a prototype of a processor with 80 cores that delivered TFLOPS (tera-floating-point operations per second) performance in 2006.

The compute resources for parallel computing may include a single computer with multiple processors or one processor with multiple cores, an arbitrary number of computers connected by a network, or a combination of both.

Research efforts in this book involve all three types of computing resources and have employed 18 different computer platforms to observe how the performance of the in-core and out-of-core parallel solvers scales in various hardware and software environments.

Table 1.2 provides a summary of the various computing platforms that the authors have had access to during this research on frequency-domain parallel IE solvers. Here, the term *node* is used to characterize a piece of hardware with a network address. A node may contain multiple cores or CPUs for processing.

The platforms listed in Table 1.2 cover a range of computer architectures, including single-core, dual-core, and quad-core CPUs from both of the primary CPU manufacturers, Intel and AMD. The two common operating systems (OS) for computational platforms, Linux and Windows, and the three system manufacturers, Dell, IBM, and HP, are all represented in this study. Detailed information on the various platforms can be found in Appendix A.

TABLE 1.2. A Summary of the Computer Platforms Used for This Research

Processor Architecture	Platform (Manufacturer)	CPU	Total Nodes	Total Cores	Operating System
IA-32	CEM-1	Intel single-core	1	8	Windows
	CEM-5	Intel single-core	6	6	Linux
EM64T	CEM-2 (Dell)	Intel dual-core	1	4	Windows
	CEM-4 (Dell)	Intel single-core	10	20	Linux
	CEM-7 (Dell)	Intel quad-core	2	16	Linux
	CEM-8 (Dell)	Intel quad-core	1	8	Windows
	CEM-9 (IBM)	Intel quad-core	2	16	Linux
	CEM-10	Intel dual-core	1	2	Windows
	Cluster-1 (HP)	Intel dual-core	40	160	Linux
	Cluster-2 (HP)	Intel quad-core	65	520	Linux
	Cluster-3 (HP)	Intel quad-core	20	160	Linux
	Cluster-4 (HP)	Intel quad-core	20	160	Linux
	Cluster-5 (HP)	Intel dual-core	40	160	Linux
	Cluster-6 (HP)	Intel dual-core	60	240	Linux
	Cluster-7 (IBM)	Intel quad-core	14	112	Linux
	Cluster-8 (IBM)	Intel quad-core	8	64	Linux
AMD64	CEM-6 (IBM)	AMD dual-core	2	16	Linux
IA-64	CEM-3 (HP)	Intel single-core	1	4	Windows

The portability of the codes developed in this book has been verified by executing them on the various platforms listed in Table 1.2. Portability is a characteristic of a well-constructed program (code) that is written in such a way that the same code can be recompiled and run successfully in different environments, i.e., different operating systems, different processor architectures, different versions of libraries, and the like. Portability is important to long-term maintenance of the code by ensuring its robustness to different computational platforms.

1.3 PARALLEL LIBRARIES EMPLOYED FOR THE COMPUTATIONS

Choice of the appropriate software (operating system and the relevant scientific subroutine package libraries) is very important to achieve an optimum efficiency for parallel computation. There are several projects for the parallelization of numerical software. An overview of the public-domain libraries for high performance computing can be found at the HPCNetlib homepage at http://www.nhse.org/hpc-netlib. These libraries are generally very specialized and optimized for the particular hardware platform on which the problem is to be executed.

Basically, the libraries in use for parallel computation consist of mathematical subroutines and programs implementing the MPI protocols. Understanding the importance of matching the software with the hardware has led to various computer manufacturers developing their own libraries.

For example, as for math libraries, the AMD Core Math Library (ACML) released by AMD is specifically designed to support multithreading and other key features of AMD's next-generation processors. ACML currently supports OpenMP, while future releases will expand on its support of multiplatform, shared-memory multiprocessing [86]. Intel has come up with the Intel Math Kernel Library (Intel MKL), a math library highly optimized for Intel Itanium, Intel Xeon, Intel Pentium 4, and Intel Core 2 Duo processor-based systems. Intel MKL performance is competitive with that of other math software packages on non-Intel processors [87].

As for the MPI library, the HP-MPI has been developed by the Hewlett-Packard (HP) company for Linux, HP-UX, Tru64 UNIX, Microsoft Windows Compute Cluster Server 2003, and Windows XP Professional systems. It is a high-performance and production quality implementation of the message passing interface (MPI) standard for HP servers and workstations [88]. Intel MPI library from the Intel Company implements the high-performance MPI-2 specification on multiple fabrics. The Intel MPI library enables users to quickly deliver end-user maximum performance even when they change or upgrade to new interconnects, without requiring major changes to the software or to the operating environment. Intel also provides a free runtime environment kit for products developed with the Intel MPI library [89].

For the computations used in this book, Intel MKL is employed on the computational platforms with Intel CPUs, and AMD ACML is used on platforms with AMD CPUs. These math libraries along with ScaLAPACK and PLAPACK, two of the most general parallel library packages based on message passing with MPI, can be used to obtain a satisfactory parallel computational efficiency. While HP-MPI is used for compiling the CEM source codes and launching the parallel jobs on HP clusters with the Linux operating system, Intel MPI is used on other computer platforms also using the Linux operating system. For computer platforms that use the Windows operating system, different MPICH2 packages are downloaded from the Argonne National Laboratory webpage (http://www.mcs.anl.gov/research/projects/mpich2/index.php) and installed according to the system architecture.

In the following paragraphs, ScaLAPACK and PLAPACK will be introduced in some detail.

1.3.1 ScaLAPACK — Scalable Linear Algebra PACKage

ScaLAPACK is the largest and most flexible public-domain library with basic numerical operations for distributed-memory parallel systems to date. It can solve problems associated with systems of linear equations, linear least-squares problems, eigenvalue problems, and singular value decomposition. ScaLAPACK can also handle many associated computations such as matrix factorizations and estimation of the condition number of a matrix.

ScaLAPACK is a parallel version of LAPACK in both function and software design. Like LAPACK, the ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. The fundamental building blocks of the ScaLAPACK library are distributed-memory versions of the level 1, level 2, and level 3 BLAS (*b*asic *l*inear *a*lgebra *s*ubprograms [90]), called the *p*arallel BLAS (PBLAS) [91], and a set of *b*asic *l*inear *a*lgebra *c*ommunication *s*ubprograms (BLACS) [92] for communication tasks that arise frequently in parallel linear algebra computations. In the ScaLAPACK routines, the majority of interprocessor communication occurs within the PBLAS, so the source code of the top software layer of ScaLAPACK resembles that of LAPACK.

Figure 1.1 describes the ScaLAPACK software hierarchy [93]. The components below the dashed line, labeled “Local”, are called on a single process, with arguments stored on a single process only. The components above the dashed line, labeled “Global”, are synchronous parallel routines, whose arguments include matrices and vectors distributed across multiple processes. The components below the solid line are machine-specific, while those above the solid line are machine-independent. Each component in Figure 1.1 is described in the following with the various acronyms defined.

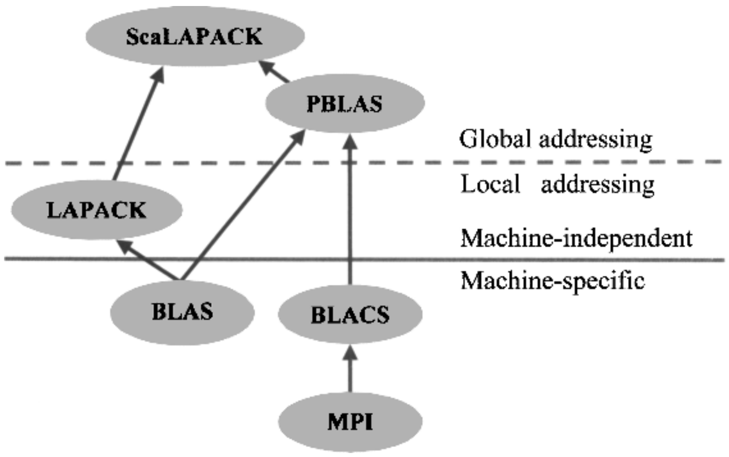

Figure 1.1. ScaLAPACK software hierarchy.

MPI. The message passing interface (MPI) standard is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users. It is a language-independent communications protocol used to program parallel computers. The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way, with language-specific syntax (bindings), plus a few features that are language-specific. MPI has such functions included, but are not limited to, point-to-point rendezvous-type send/receive operations, choosing between a Cartesian or graph-like logical process topology, exchanging data between process pairs (send/receive operations), combining partial results of computations (gathering and reduction operations), synchronizing nodes (barrier operations), as well as obtaining network-related information (such as the number of processes), neighboring processes accessible in a logical topology, and so on. MPI programs always work with processes, although usually people talk about processors. When one tries to achieve maximum performance, one process per processor/core is selected as part of the mapping activity. This mapping activity occurs at runtime, through the agent that starts the MPI program, normally called mpirun or mpiexec [94,95].

BLAS. BLAS (*b*asic *l*inear *a*lgebra *s*ubprograms [90]) include subroutines for common linear algebra computations such as dot product, matrix–vector multiplication, and matrix–matrix multiplication. An important aim of the BLAS is to provide a portability layer for computation. As is well known, using matrix–matrix operations (in particular, matrix multiplication) tuned for a particular architecture can mask the effects of the memory hierarchy (cache misses, *t*ranslation *l*ook-aside *b*uffer (TLB) misses, etc.) and permit floating-point operations to be performed near peak speed of the machine.

Optimized versions of the BLAS can be found at http://www.tacc.utexas.edu/resources/software/#blas.

BLACS – BLACS (*b*asic *l*inear *a*lgebra *c*ommunication *s*ubprograms [92]) is a message-passing library designed for linear algebra. An important aim of the BLACS is to provide a portable, linear algebra-specific layer for communication. The computational model consists of a one- or two-dimensional *process grid*, where each process stores pieces of the matrices and vectors. The BLACS include synchronous send/receive routines to communicate a matrix or submatrix from one process to another, to broadcast submatrices to many processes, or to compute global reductions (sums, maxima, and minima). There are also routines to construct, change, or query the process grid. Since several ScaLAPACK algorithms require broadcasts or reductions among different subsets of processes, the BLACS permit a process to be a member of several overlapping or disjointed process grids, each one labeled by a context. Some message-passing systems, such as MPI, also include this context concept; MPI calls this a "communicator". The BLACS provide facilities for safe interoperation of system contexts and BLACS contexts.

LAPACK. LAPACK, or *l*inear *a*lgebra *pack*age [96], is a collection of routines for solving linear systems, least-squares problems, eigenproblems, and singular problems. High performance is attained by using algorithms that do most of their work in calls to the BLAS, with an emphasis on matrix–matrix multiplication. Each routine has one or more *performance tuning parameter*, such as the sizes of the blocks operated on by the BLAS. These parameters are machine-dependent and are obtained from a table defined when the package is installed and referenced at runtime.

The LAPACK routines are written as a single thread of execution. LAPACK can accommodate shared-memory machines, provided parallel BLAS are available (in other words, the only parallelism is implicit in calls to BLAS). More detailed information about LAPACK can be found at http://www.netlib.org/lapack/.

PBLAS. To simplify the design of ScaLAPACK, and because BLAS have proved to be useful tools outside LAPACK, the authors of ScaLAPACK chose to build a parallel set of BLAS, called PBLAS, which perform message passing and whose interface is as similar to the BLAS as possible. This decision has permitted the ScaLAPACK code to be quite similar, and sometimes nearly identical, to the analogous LAPACK code. Further details of PBLAS can be found in reference [91].

ScaLAPACK also contains additional libraries to treat distributed matrices and vectors. One is the tools library, which offers useful routines, for example, to find out which part of the global matrix a local process has in its memory or to identify the global index of a matrix element corresponding to its local index and vice versa.

The research done for this book has found some drawbacks associated with the commercial math library packages. For example, each process cannot address more than 2 GB RAM for LU decomposition when using the earlier version of

Intel Cluster Math Kernel Library (CMKL). Parallel computation on the Itanium 2 platform, using the Windows operating system, is not supported by Intel or HP [97]. Fortunately, these problems have so far been solved during the research work for this book by the authors and/or the vendors.

1.3.2 PLAPACK — Parallel Linear Algebra PACKage

The *p*arallel *l*inear *a*lgebra *pack*age (PLAPACK) is a prototype of a more flexible alternative to ScaLAPACK. Containing a number of parallel linear algebra solvers, PLAPACK is an MPI-based parallel linear algebra package designed to provide a user-friendly infrastructure for building parallel dense linear algebra libraries.

Figure 1.2 gives the layering of the PLAPACK infrastructure with the part concerning *m*anaged *m*essage *p*assing *i*nterface (MMPI) omitted from reference [83] since it is not used by the library employed in the solution of the problems related to this book. The components of PLAPACK shown in the bottom row (below the solid line) in Figure 1.2 are machine-dependent. The components that are above the solid line are machine-independent.

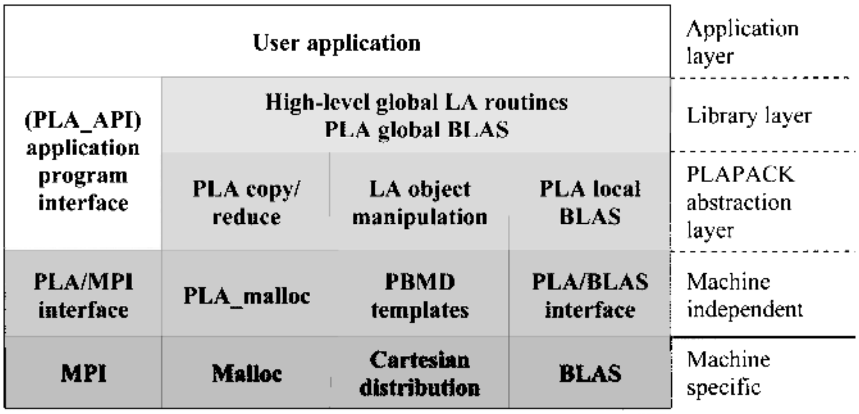

Figure 1.2. Layering of the PLAPACK infrastructure.

To ensure completeness of the book, a very brief introduction is given for each layer of the PLAPACK infrastructure, based on the literature [98] to which the reader may refer to for a detailed description.

Machine/distribution-dependent layer. To ensure portability, PLAPACK uses standardized components, namely, the MPI for communication, the standard memory management routines provided by the C programming language (*malloc*/*calloc*), and the BLAS that is generally optimized by the vendors. The author of PLAPACK also adds Cartesian matrix distributions to this layer, since they provide the most general distribution of matrices commonly used for implementation of parallel dense linear algebra algorithms.

Machine/distribution-independent layer. To achieve machine independence, PLAPACK offers a number of interfaces to the machine-dependent layer. Each interface is briefly described below.

PLAPACK-MPI interface. PLAPACK relies heavily on collective communications like scatter (MPI_scatter), gather (MPI_gather), collect (MPI_allgather), broadcast (MPI_bcast), and others. PLAPACK's developers created an intermediate layer that can be used to pick a particular implementation of such an operation.

PLAPACK-memory management interface. PLAPACK uses dynamic memory allocation to create space for storing data. By creating this extra layer, it provides a means for customizing memory management, including the possibility of allowing the user to provide all space used by PLAPACK.

Physically based matrix distribution (PBMD) and templates. The approach taken to describe Cartesian matrix distributions is fundamental to PLAPACK. In particular, PLAPACK recognizes the important role that vectors play in applications and thus all distribution of data starts with the distribution of the vectors [30]. The details of the distribution are hidden by describing the generic distribution of imaginary vectors and matrices (the template) and indicating how actual matrices and vectors are aligned to the template.

Physically based matrix distribution was proposed by the authors of PLAPACK as a basis for a set of more flexible parallel linear algebra libraries. It was claimed that developers of applications will not have to "unnaturally" modify the way they want to distribute the data in order to fit the distribution required by the format of the respective library. They have the freedom to distribute the vectors (which contain the data of "physical significance") across processors in a way that depends only on the application. The matrix (the operator) is then distributed in a conforming way that, in effect, optimizes matrix−vector products (MVPs) involving the vectors and the matrix.

PLAPACK-BLAS interface. Computation is generally performed locally on each processing node by the BLAS, which have a FORTRAN-compatible calling sequence. Since the interface between C and FORTRAN is not standardized, a PLAPACK-BLAS interface is required to hide these platform specific differences. To make the development of programs easier and to achieve better performance, PLAPACK includes the parallel version of the BLAS routines.

PLAPACK abstraction layer. This layer of PLAPACK provides the abstraction that frees users from details like indexing, communication, and local computation.

Linear algebra objects and their manipulation. All information that describes a linear algebra object, like a vector or a matrix, is encapsulated in an opaque object. This component of PLAPACK allows users to create, initialize, and destroy such objects. In addition, it provides an abstraction that allows users to transparently index into a submatrix or vector. Finally, it provides a mechanism for describing the duplication of the data.

Copy/reduce: duplication and consolidation of data. Communication in PLAPACK is not specified by explicit communication operations. Instead, linear algebra objects are used to describe how data are to be distributed or duplicated. Communication is achieved by copying or reducing from one (duplicated) object to another. This raises the level of abstraction at which communication can be specified.

PLAPACK local BLAS. Since all information about matrices and vectors is hidden in the linear algebra objects, a call to a BLAS routine on a given processor requires extraction of that information. Rather than exposing this, PLAPACK provides routines (the PLAPACK local BLAS) that extract the information and subsequently call the correct sequential BLAS routine on each processor.

Library layer. The primary intent for the PLAPACK infrastructure is to provide the building blocks for creating higher-level libraries. Thus, the library layer for PLAPACK consists of global (parallel) basic linear algebra subprograms and higher-level routines for solving linear systems and algebraic eigenvalue problems.

PLAPACK global BLAS. The primary building blocks provided by PLAPACK are the global (parallel) versions of the BLAS. These allow dense linear algebra algorithms to be implemented quickly without exposing parallelism in any form.

PLAPACK higher-level linear algebra routines. Higher-level algorithms can be easily implemented using the global BLAS. However, to ensure better performance, it is often desirable to implement these higher-level algorithms directly using the abstraction level of PLAPACK. Development of such implementations can often be attained by incrementally replacing calls to global BLAS with calls that explicitly expose parallelism by using objects that are duplicated in nature.

PLAPACK application interface. A highlight of the PLAPACK infrastructure is the inclusion of a set of routines that allow an application to build matrices and vectors in an application friendly manner.

PLAPACK does not offer as many blackbox solvers as ScaLAPACK, but is designed as a parallel infrastructure to develop routines for solving linear algebra problems. With PLAPACK routines, the user can create a global matrix, vectors, and multiscalars, multivectors, and may fill them with values with the help of an API (application programming interface).

Since PLAPACK is not as commercialized as ScaLAPACK, joint efforts have been made by the authors of this book and Robert A. van de Geijn, the author of PLAPACK, to make the library compatible to EM64T systems

[99,100]. The method for generating a Windows version of the PLAPACK library is also provided in Appendix B.

1.4 CONCLUSION

An introduction to parallel CEM is given with a special attention to a parallel MoM methodology to solve EM problems in frequency domain. As will be discussed in the later sections, this methodology can be a powerful tool to solve challenging computational radiation and scattering problems by using parallel out-of-core techniques. A summary of the various computing platforms and the software to parallelize the computations is also provided. The parallel computing platforms involve the most typical architectures available, including IA-32, EM64T, and IA-64 systems. Two popular parallel scientific libraries, ScaLAPACK and PLAPACK, are briefly described to familiarize the reader with them as they will be used later on for implementing parallel solvers for matrix equations.

REFERENCES

[1] A. E. Bondeson, T. Rylander, and P. Ingelström, *Computational Electromagnetics,* Springer, New York, 2005.

[2] R. A. Elliott, *Electromagnetics*, McGraw-Hill, New York, 1966.

[3] R. F. Harrington, *Field Computation by Moment Methods,* Macmillan, New York, 1968.

[4] W. C. Chew, J. Jin, E. Michielssen, and J. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, Norwood, MA, 2000.

[5] P. P. Silvester and R. L. Ferrari, *Finite Elements for Electrical Engineers,* Cambridge University Press, New York, 1983.

[6] J. M. Jin, *The Finite Element Method in Electromagnetics*, Wiley, New York, 1993.

[7] M. Salazar-Palma, T. K. Sarkar, L. E. Garcia-Castillo, T. Roy, and A. Djordjevic, *Iterative and Self-Adaptive Finite-Elements in Electromagnetic Modeling*, Artech House, Norwood, MA, 1998.

[8] R. Coccioli, T. Itoh, G. Pelosi, and P. P. Silvester, *Finite Element Methods in Microwaves: A Selected Bibliography*, University of Florence, Department of Electronics and Telecommunciations. Available at: http://ingfi9.die.unifi.it/fem corner/map/cont2_1.htm. Accessed July 2008.

[9] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Transactions on Antennas and Propagation*, Vol. 14, No. 3, pp. 302–307, May 1966.

[10] K. L. Shlager and J. B. Schneider, "A Selective Survey of the Finite-Difference Time-Domain Literature," *IEEE Antennas and Propagation Magazine*, Vol. 37, No. 4, pp. 39–56, Aug. 1995.

[11] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method,* 3rd ed., Artech House, Norwood, MA, 2005.

[12] S. M. Rao and N. Balakrishnan, "Computational Electromagnetics — A Review,"

Indian Academy of Science, Current Science Online, 2000. Available at: http://www.ias.ac.in/currsci/nov25/articles24.htm. Accessed July 2008.

[13] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*, Benjamin-Cummings, Redwood City, CA, 1989.

[14] Lawrence Livermore National Laboratory, "Introduction to Parallel Computing," *High Performance Computing Training*, 2007. Available at: https://computing .llnl.gov/tutorials/parallel_comp/#WhyUse. Accessed July 2008.

[15] J. S. Shang, "Computational Electromagnetics," *ACM Computing Surveys*, Vol. 28, No. 1, pp. 97–99, March 1996.

[16] A. Rubinstein, F. Rachidi, M. Rubinstein, and B. Reusser, "A Parallel Implementation of NEC for the Analysis of Large Structures," *IEEE Transactions on Electromagnetic Compatibility*, Vol. 45, No. 2, pp. 177–188, May 2003.

[17] Y. Zhang, T. K. Sarkar, P. Ghosh, M. Taylor, and A. De, "Highly Efficient Parallel Schemes Using Out-of-Core Solver for MoM," *IEEE Applied Electromagnetics Conference–AEMC*, Kolkata, India, Dec. 2007.

[18] S. Velamparambil, W. C. Chew, and J. Song, "10 Million Unknowns: Is It That Big?" *IEEE Antennas and Propagation Magazine*, Vol. 45, No. 2, pp. 43–58, April 2003.

[19] C. Waltz, K. Sertel, M. A. Carr, B. C. Usner, and J. L. Volakis, "Massively Parallel Fast Multipole Method Solutions of Large Electromagnetic Scattering Problems," *IEEE Transactions on Antennas and Propagation,* Vol. 55, No. 6, pp. 1820–1816, June 2007.

[20] G. Hennigan and S. Castillo, "Open-region, Electromagnetic Finite-element Scattering Calculations in Anisotropic Media on Parallel Computers," *IEEE Antennas and Propagation Society International Symposium,* Orlando, FL, 1999.

[21] D. H. Malan and A. C. Metaxas, "Implementing a Finite Element Time Domain Program in Parallel," *IEEE Antennas and Propagation Magazine*, Vol. 42, No. 1, pp. 105–109, 2000.

[22] C. Guiffaut and K. Mahdjoubi, "A Parallel FDTD Algorithm Using the MPI Library," *IEEE Antennas and Propagation Magazine*, Vol. 43, No. 2, pp. 94–103, 2001.

[23] Y. Zhang, W. Ding, and C. H. Liang, "Study on the Optimum Virtual Topology for MPI Based Parallel Conformal FDTD Algorithm on PC Clusters," *Journal of Electromagnetic Waves and Applications*, Vol. 19, No. 13, pp. 1817–1831, Oct. 2005.

[24] W. Yu, Y. Liu, T. Su, N. Huang, and R. Mittra, "A Robust Parallelized Conformal Finite Difference Time Domain Field Solver Package Using the MPI Library," *IEEE Antennas and Propagation Magazine*, Vol. 47, No. 3, 2005.

[25] Y. Zhang, A. De, B. H. Jung, and T. K. Sarkar, "A Parallel Marching-on-in-Time Solver," *IEEE AP-S & USNC/URSI Symposium*, San Diego, CA, July 2008.

[26] J. E. Patterson, T. Cwik, R. D. Ferraro, N. Jacobi , P. C. Liewer, T. G. Lockhart, G. A. Lyzenga, J. W. Parker, and D. A. Simoni, "Parallel Computation Applied to Electromagnetic Scattering and Radiation Analysis," *Electromagnetics*, Vol. 10, No. 1–2, pp. 21–39, Jan. –June 1990.

[27] T. Cwik, J. Partee, and J. E. Patterson, "Method of Moment Solutions to Scattering Problems in a Parallel Processing Environment," *IEEE Transactions on Magnetics*, Vol. 27, No. 5, pp. 3837–3840, Sept. 1991.

[28] D. B. Davidson, "Large Parallel Processing Revisited: A Second Tutorial," *IEEE Antennas and Propagation Magazine*, Vol. 34, No. 5, pp. 9–21, Oct. 1992.

[29] T. Marshall, "Numerical Electromagnetics Code (Method of Moments)," *Numerical Electromagnetics Code NEC2 Unofficial Home Page*, 2002. Available

at: http://www.nec2.org/. Accessed Aug. 2008.
[30] R. A. van de Geijn, *Using PLAPACK— Parallel Linear Algebra Package*, MIT Press, Cambridge, MA, 1997.
[31] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic Scattering by Surfaces of Arbitrary Shape," *IEEE Transactions on Antennas and Propagation*, Vol. 30, No. 3, pp. 409–418, May 1982.
[32] T. Cwik, R. A. van de Geijn, and J. E. Patterson, "The Application of Parallel Computation to Integral Equation Models of Electromagnetic Scattering," *Journal of the Optical Society of America*, Vol. 11, No. 4, pp. 1538–1545, April 1994.
[33] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK User's Guide*, Society for Industrial Mathematics, Philadelphia, PA, 1997.
[34] U. Jakobus, "Parallel Computation of Electromagnetic Fields Based on Integral Equations," *Penn State Citeseer*, 1999. Available at: http://citeseer.ist.psu.edu /cachedpage/438895/9. Accessed Aug. 2008.
[35] Lawrence Livermore National Laboratory, "Message Passing Interface (MPI)," *High Performance Computer Training*. Available at: https://computing.llnl.gov /tutorials/mpi/. Accessed Aug. 2008.
[36] Oak Ridge National Laboratory, "PVM: Parallel Virtual Machine," *Computer Science and Mathematics*. Available at: http://www.csm.ornl.gov/pvm/. Accessed Aug. 2008.
[37] E. F. Knott and T. B. A. Senior, "Comparison of Three High-Frequency Diffraction Techniques," *Proceedings of the IEEE*, Vol. 62, No. 11, pp. 1468–1474, Nov. 1974.
[38] J. S. Asvestas, "The Physical Optics Method in Electromagnetic Scattering," *Journal of Mathematical Physics*, Vol. 21, No. 2, pp. 290–299, Feb. 1980.
[39] SuperNEC-EM Antenna Simulation Software, *SuperNEC–Parallel MoM User Reference Manual. Version 2.7*. Available at: http://www.supernec.com /manuals/snparurm.htm. Accessed Aug. 2008.
[40] J. Edlund, *A Parallel, Iterative Method of Moments and Physical Optics Hybrid Solver for Arbitrary Surfaces*, licentiate thesis, Uppsala University, 2001.
[41] Y. Zhang, X. W. Zhao, M. Chen, and C. H. Liang, "An Efficient MPI Virtual Topology Based Parallel, Iterative MoM-PO Hybrid Method on PC Clusters," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 5, pp. 661–676, 2006.
[42] T. K. Sarkar, E. Arvas, and S. M. Rao, "Application of FFT and the Conjugate Gradient Method for the Solution of Electromagnetic Radiation from Electrically Large and Small Conducting Bodies," *IEEE Transactions on Antennas and Propagation*, Vol. 34, No. 5, pp. 635–640, May 1986.
[43] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive Integral Method for Solving Large-Scale Electromagnetic Scattering and Radiation Problems," *Radio Science*, Vol. 31, No. 5, pp. 1225–1251, 1996.
[44] V. Rokhlin, "Rapid Solution of Integral Equations of Scattering Theory in Two Dimensions," *Journal of Computational Phyics*, Vol. 86, pp. 414–439, 1990.
[45] X. C. Nie, L. W. Li, and N. Yuan, "Precorrected-FFT Algorithm for Solving Combined Field Integral Equations in Electromagnetic Scattering," *IEEE Antennas and Propagation Society International Symposium*, San Antonio, TX, Vol. 3, pp. 574–577, June 2002.
[46] J. M. Song, C. Lu, and W. Chew, "Multilevel Fast Multipole Algorithm for

Electromagnetic Scattering by Large Complex Objects," *IEEE Transactions on Antennas and Propagation*, Vol. 45, No. 10, pp. 1488–1493, Oct. 1997.

[47] J. Song, C. Lu, W. Chew, and S. Lee, "Fast Illinois Solver Code (FISC)," *IEEE Antennas and Propagation Magazine,* Vol. 40, No. 3, pp. 27–34, June 1998.

[48] S. Velamparambil, J. Schutt-Aine, J. Nickel, J. Song, and W. Chew, "Solving Large Scale Electromagnetic Problems Using a Linux Cluster and Parallel MLFMA," *IEEE Antennas and Propagation Society International Symposium,* Vol. 1, pp. 636–639, Orlando, FL, July 1999.

[49] S. Velamparambil and W. Chew, "Analysis and Performance of a Distributed Memory Multilevel Fast Multipole Algorithm," *IEEE Transactions on Antennas and Propagation*, Vol. 53, No. 8, pp. 2719–2727, Aug. 2005.

[50] S. Velamparambil, J. Song, and W. C. Chew, "Parallelization of Multilevel Fast Multipole Algorithm on Distributed Memory Computers in Fast and Efficient Algorithms," *Computational Electromagnetics*, Chap. 4, Artech House, Boston, MA, 2001.

[51] E. J. Lu and D. Okunbor, "A Massively Parallel Fast Multipole Algorithm in Three Dimensions," *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing,* pp. 40–48, Aug. 1996.

[52] E. J. Lu and D. Okunbor, "Parallel Implementation of 3-D FMA Using MPI," *Proceedings of 2nd MPI Developer's Conference, IEEE Computer Society Technical Committee on Distributed Processing*, pp. 119–124, Notre Dame, IN., July 1996.

[53] P. Have, "A Parallel Implementation of the Fast Multipole Method for Maxwell's Equations," *International Journal for Numerical Methods in Fluids*, Vol. 43, No. 8, pp. 839–864, Nov. 2003.

[54] M. A. Stalzer, "A Parallel Fast Multipole Method for the Helmholtz Equation," *Parallel Processing Letters*, Vol. 5, No. 2, pp. 263–274, June 1995.

[55] S. Velamparambil, W. C. Chew, and M. L. Hastriter, "Scalable Electromagnetic Scattering Computations," *IEEE Antennas and Propagation Society International Symposium, 2002,* Vol. 3, pp. 176–179, San Antonio, TX, June 2002.

[56] S. Velamparambil, J. Song, W. Chew, and K. Gallivan, "ScaleME: A Portable Scaleable Multipole Engine for Electromagnetic and Acoustic Integral Equation Solvers," *IEEE Antennas and Propagation Society International Symposium, 1998*, Vol. 3, pp. 1774–1777, June 1998.

[57] M. Nilsson, *A Parallel Shared Memory Implementation of the Fast Multipole Method for Electromagnetics*, Technical Report 2003-049, Department of Information Technology, Scientific Computing, Uppsala University, Uppsala, Sweden, Oct. 2003.

[58] FEKO, "Comprehensive EM Solutions Field Computations Involving Objects of Arbitrary Shape," Available at: http://www.feko.info/. Accessed Aug. 2008.

[59] E. D'Azevedo and J. J. Dongarra, *The Design and Implementation of the Parallel Out-of-Core ScaLAPACK LU, QR and Cholesky Factorization Routines*, University of Tennessee, Knoxville, Citeseer, 1997. Available at: http://citeseer.ist.psu.edu/article/dazevedo97design.html. Accessed Aug. 2008.

[60] B. M. Kolundzija , J. S. Ognjanovic T. K. Sarkar, and R. F. Harrington, *WIPL: Electromagnetic Modeling of Composite Wire and Plate Structures — Software and User's Manual,* Artech House, Norwood, MA, 1995.

[61] B. M. Kolundzija, J. S. Ognjanovic, and T. K. Sarkar, *WIPL-D: Electromagnetic Modeling of Composite Metallic and Dielectric Structures, Software and User Manual*, Artech House, Norwood, MA, 2000.

[62] B. M. Kolundzija, J. S. Ognjanovic, and T. K. Sarkar, "Analysis of Composite

Metallic and Dielectric Structures — WIPL-D Code," *Proceedings of 17th Applied Computational Electromagnetics Conference*, Monterey, CA, pp. 246–253, March 2001.

[63] Y. Zhang, T. K. Sarkar, H. Moon, A. De, and M. C. Taylor, "Solution of Large Complex Problems in Computational Electromagnetics Using Higher Order Basis in MoM with Parallel Solvers," *IEEE Antennas and Propagation Society International Symposium*, pp. 5620–5623, Honolulu, HI, June 2007.

[64] Y. Zhang, T. K. Sarkar, H. Moon, M. Taylor, and M. Yuan, "The Future of Computational Electromagnetics for Solving Large Complex Problems: Parallel In-Core Integral Equation Solvers," *EMTS 2007 International URSI Commission B — Electromagnetic Theory Symposium*, Ottawa, ON, Canada, July 2007.

[65] Y. Zhang, J. Porter, M. Taylor, and T. K. Sarkar, "Solving Challenging Electromagnetic Problems Using MoM and a Parallel Out-of-Core Solver on High Performance Clusters," *2008 IEEE AP-S* & USNC/*URSI Symposium*, San Diego, CA, July 2008.

[66] J. Mackerle, "Implementing Finite Element Methods on Supercomputers, Workstations and PCs a Bibliography (1985–1995)," *Engineering Computations*, Vol. 13 No. 1, pp. 33–85, 1996. Available at: http://www.emeraldin sight.com/Insight/ViewContentServlet?Filename=/published/emeraldabstractonly article/pdf/1820130103.pdf. Accessed Aug. 2008.

[67] C. T. Wolfe, U. Navsariwala, and S. D. Gedney, "A Parallel Finite-element Tearing and Interconnecting Algorithm for Solution of the Vector Wave Equation with PML Absorbing Medium," *IEEE Transactions on Antennas and Propagation*, Vol. 48, No. 2, pp. 278–284, Feb. 2000.

[68] W. P. Pala, A. Taflove, M. J. Piket, and R. M. Joseph, "Parallel Finite Difference-Time Domain Calculations," *International Conference on Computation in Electromagnetics*, pp. 83–85, Nov. 1991.

[69] V. Varadarajan and R. Mittra, "Finite-Difference Time-Domain Analysis Using Distributed Computing," *IEEE Microwave and Guided Wave Letters*, Vol. 4, pp. 144–145, 1994.

[70] Z. M. Liu, A. S. Mohan, T. A. Aubrey, and W. R Belcher, "Techniques for Implementation of the FDTD Method on a CM-5 Parallel Computer," *IEEE Antennas and Propagation Magazine*, Vol. 37, No. 5, pp. 64–71, 1995.

[71] A. D. Tinniswood, P. S. Excell, M. Hargreaves, S. Whittle, and D. Spicer, "Parallel Computation of Large-Scale FDTD Problems," *Computation in Electromagnetics, 3rd International Conference, (Conf. Publ. No. 420)*, Vol. 1, pp. 7–12, April 1996.

[72] H. Simon, *Parallel Rigorous Electrodynamic Simulation using the Finite-Difference Time Domain Method*, National Energy Research Scientific Computing (NERSC) Center, Computational Research Division. Available at: http://www.nersc.gov/~simon/cs267/hw0/hafeman/. Accessed Aug. 2008.

[73] U. Andersson, *Time-Domain Methods for the Maxwell Equations*, PhD thesis, KTH, Sweden, 2001.

[74] Y. Zhang, W. Ding, and C. H. Liang, "Analysis of Parallel Performance of MPI Based Parallel FDTD on PC Clusters," *Microwave Conference Proceedings, 2005, APMC 2005, Asia-Pacific Conference Proceedings*, Vol. 4, pp. 3, Dec. 2005.

[75] Y. Zhang, *Parallel Computation in Electromagnetics*, Xidian University Press, Xi'an, China, 2006.

[76] H. Malan and A. C. Metaxas, "Implementing a Finite Element Time Domain

Program in Parallel," *IEEE Antennas and Propagation Magazine*, Vol. 42, No. 1, pp. 105–109, Feb. 2000.

[77] B. Butrylo, C. Vollaire, and L. Nicolas, "Parallel Implementation of the Vector Finite Element and Finite Difference Time Domain Methods," *Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on*, pp. 347–352, Sept. 2002.

[78] S. M. Rao and T. K. Sarkar, "Numerical Solution of Time Domain Integral Equations for Arbitrarily Shaped Conductor/Dielectric Composite Bodies," *IEEE Transactions on Antennas and Propagation*, Vol. 50, No. 12, pp. 1831–1837, Dec. 2002.

[79] B. H. Jung, Z. Ji, T. K. Sarkar, M. Salazar-Palma and M. Yuan, "A Comparison of Marching-on-in-time Method with Marching-on-in-degree Method for the TDIE Solver," *Progress in Electromagnetics Research*, Vol. 70, pp. 281–296, 2007.

[80] Wikipedia contributors, "IA-32," *Wikipedia, The Free Encyclopedia*, July 2007. Available at: http://en.wikipedia.org/w/index.php?title=IA-32&oldid=146752470. Accessed Oct. 2007.

[81] Netlib Repository at UTK and ORNL, "LAPACK — Linear Algebra PACKage." Available at: http://www.netlib.org/lapack/. Accessed Aug. 2008.

[82] Netlib Repository at UTK and ORNL, *The ScaLAPACK Project*. Available at: http://www.netlib.org/scalapack/. Accessed Aug. 2008.

[83] G. Morrow and R. A. van de Geijn, *Half-day Tutorial on: Using PLAPACK: Parallel Linear Algebra PACKage*, Texas Institute for Computational and Applied Mathematics, University of Texas at Austin, Austin, TX. Available at: http://www.cs.utexas.edu/users/plapack/tutorial/SC98/. Accessed Aug. 2008.

[84] Wikipedia contributors, "Multicore," *Wikipedia, The Free Encyclopedia*. Available at: http://en.wikipedia.org/wiki/Multi-core_(computing)#Advantages. Accessed Aug. 2008.

[85] Advanced Micro Devices, Inc., *AMD Multi-core White Paper*. Available at: http://www.sun.com/emrkt/innercircle/newsletter/0505multicorewp.pdf. Accessed Aug. 2008.

[86] Advanced Micro Devices, Inc., *AMD Core Math Library (ACML)*, AMD Developer Central. Available at: http://developer.amd.com/cpu/libraries/acml/Pages/default.aspx. Accessed Aug. 2008.

[87] Intel®Software Network, *Intel®Math Kernel Library 10.0 Overview*. Available at: http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm. Accessed Aug. 2008.

[88] HP Technical Documentation, *HP-MPI User's Guide*. Available at: http://docs.hp.com/en/B6060-96022/index.html. Accessed Aug. 2008.

[89] Intel®Software Network, *Intel®MPI Library 3.1 for Linux or Microsoft Windows Compute Cluster Server (CCS)*. Available at: http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/308295.htm

[90] Netlib Repository at UTK and ORNL, *Basic Linear Algebra Subprograms (BLAS)*. Available at: http://www.netlib.org/blas/. Accessed Aug. 2008.

[91] Netlib Repository at UTK and ORNL, The ScalAPACK Project, *Parallel Basic Linear Algebra Subprograms (PBLAS) Home Page*. Available at: http://www.netlib.org/scalapack/pblas_qref.html. Accessed Aug. 2008.

[92] Netlib Repository at UTK and ORNL, *Basic Linear Algebra Subprograms (BLACS)*. Available at: http://www.netlib.org/blacs/. Accessed Aug. 2008.

[93] L. S. Blackford, *ScaLAPACK Tutorial*, Available at: http://www.netlib.org/scalapack/tutorial/sld053.htm. Accessed Aug. 2008.

[94] Wikipedia contributors, "Message Passing Interface," *Wikipedia, The Free*

Encyclopedia. Available at: http://en.wikipedia.org/wiki/Message_Passing_Interface. Accessed Aug. 2008.

[95] Message Passing Interface Forum, *MPI Documents*. Available at: http://www.mpi-forum.org/docs/docs.html. Accessed Aug. 2008.

[96] Netlib Repository at UTK and ORNL, *LAPACK — Linear Algebra PACKage*. Available at: http://www.netlib.org/lapack/. Accessed Aug. 2008.

[97] Y. Zhang, T. K Sarkar, A. De, N. Yilmazer, S. Burintramart, and M. Taylor, "A Cross-Platform Parallel MoM Code with ScaLAPACK Solver," *IEEE Antennas and Propagation Society International Symposium*, pp. 2797–2800, Honolulu, HI, June 2007.

[98] P. Alpatov, G. Baker, H. C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, and R. A. van de Geijn, "PLAPACK Parallel Linear Algebra Package Design Overview," *Supercomputing, ACM/IEEE 1997 Conference*, Nov. 1997.

[99] Y. Zhang, T. K. Sarkar, R. A. van de Geijn and M. C. Taylor, "Parallel MoM Using Higher Order Basis Function and PLAPACK In-Core and Out-of-Core Solvers for Challenging EM Simulations," *IEEE AP-S & USNC/URSI Symposium*, San Diego, CA, July 2008.

[100] M. C. Taylor, Y. Zhang, T. K. Sarkar, and R. A. van de Geijn, "Parallel MoM Using Higher Order Basis Functions and PLAPACK Out-of-Core Solver for a Challenging Vivaldi Array," *XXIX URSI General Assembly*, Chicago, IL, Aug. 2008.