Parallelization of a Fast Multipole Boundary Element Method with Cluster OpenMP

André Buchau, Serge Mboonjou Tsafak, Wolfgang Hafla, and Wolfgang M. Rucker

Institute for Theory of Electrical Engineering, University of Stuttgart, 70569 Stuttgart, Germany

Parallelization of a boundary element method for the solution of problems, which are based on a Laplace equation, is considered. The fast multipole method is applied to compress the belonging linear system of equations. The well-known parallelization standard OpenMP is used on shared memory computers and the new standard Cluster OpenMP is used on computer clusters. Both standards are based on multithreading and exploit multicore processors very efficiently. Cluster OpenMP is an enhancement of OpenMP. There, multiprocessing on a computer cluster is hidden by virtual threads, which use a virtual shared memory on a distributed memory computer.

Index Terms—**Boundary element methods (BEMs), Cluster OpenMP, fast multipole method (FMM), parallelization.**

I. Introduction

NOWADAYS, boundary element methods (BEMs) are well established for the solution of electromagnetic field problems, which are based on a Laplace equation. Modeling and discretization of a problem are relatively easy. Furthermore, accuracy and efficiency of modern BEM software is excellent. One reason for this purpose is that the originally fully dense matrix of the underlying linear system of equations is compressed. The loss of accuracy of matrix compression techniques like hierarchical matrices [1] or fast multipole method (FMM) [2] is negligible. Even nonlinear problems can be taken into account, if the BEM is supplemented by volume integral equations [3].

Excellent numerical methods are necessary for a successful solution of electromagnetic field problems. Nevertheless, the used computer platform significantly influences the efficiency of the method. In the past, speed of processors was increased each year. Hence, software became faster without any modifications. Today, processor speed stagnates and multicore processors are introduced. However, software must be adapted to this new computer architecture by parallelization. Multicore computers are shared memory systems. Then, parallelization can be realized by multithreading. A well-established standard for parallelization based on multithreading is OpenMP [4]. It is supported by all modern compilers and is relatively easy to implement. Last year, Intel Corporation introduced the new Cluster OpenMP standard, which is an enhancement of OpenMP [5]. Cluster OpenMP transforms a computer cluster with distributed memory into a virtual shared memory system.

Cluster OpenMP is used to parallelize a boundary element method, which is based on the fast multipole method. Efficiency is studied for electrostatic field problems. The results of the novel Cluster OpenMP approach are compared with classical OpenMP.

II. Background

A. Boundary Element Method and Fast Multipole Method

Electrostatic field problems, which are based on a Laplace equation, are considered. All matter is assumed to be linear, isotropic and piecewise homogeneous. Electric potential on conductors is given. Normal component of dielectric displacement is continuous at interfaces between two dielectrics. A similar task is the solution of steady current flow field problems. There, potential at the ports of conductors is given. Current flow is continuous at interfaces between two conductors with different conductivity. In both cases, an indirect BEM formulation based on surface charge densities is applied [6].

If the fast multipole method is used, a sparse near-field matrix is computed. It takes into account all interactions between elements, which lie close together. This sparse matrix corresponds to the originally dense matrix, from which all far-field interactions were removed. Although only a sparse matrix must be computed, this operation is time consuming and requires a fast processor. A reason for this purpose is that singular and nearly singular integrals have to be evaluated, what results in a huge number of floating point operations [7].

The linear system of equations is solved iteratively. Then, a matrix vector product has to be computed in each iteration step. This product is accelerated by the fast multipole method. The estimated solution in the current iteration step is considered as sources on the elements and the field on the elements is computed. Here, the coupling of an element with all other elements is taken into account. The FMM summarizes information of sources of a group of elements and computes the field of these sources for another group of elements. The size of these groups depends on the distance between them. Note the number of operations, which must be evaluated, is relatively small. However, a relatively large amount of data must be interchanged [7].

B. OpenMP and Cluster OpenMP

The standard mode of execution of software for numerical field computations is to run the software in a single process and a single thread on one central processing unit (CPU). All commands are processed one by one. Of course, no data must be exchanged and no conflicts can happen. Modern CPUs even vectorize some operations to accelerate computations with large arrays.

In recent years, the speed of CPUs grew steadily. Unfortunately, clock speed of CPUs stagnates now. Latest developments of CPUs go to multicore processors. Both Intel and AMD, the two most important manufacturers for standard computer CPUs, introduced so-called quad-core processors. Each of these processors possesses four cores. A dual-processor computer can execute eight independent operations in parallel. This trend makes parallelization of software not optional but indispensable.

Digital Object Identifier 10.1109/TMAG.2007.916262

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

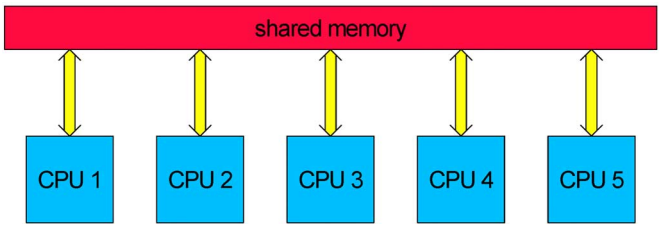

Fig. 1. Communication between threads on a shared memory computer.

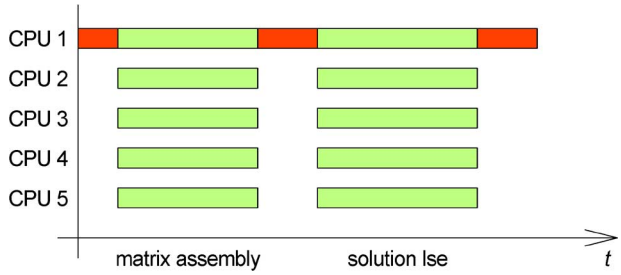

Fig. 2. Processes and threads in the case of OpenMP.

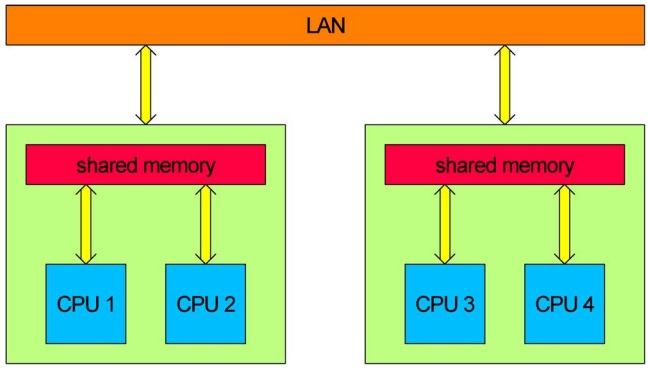

Fig. 3. Communication between threads on a distributed memory computer.

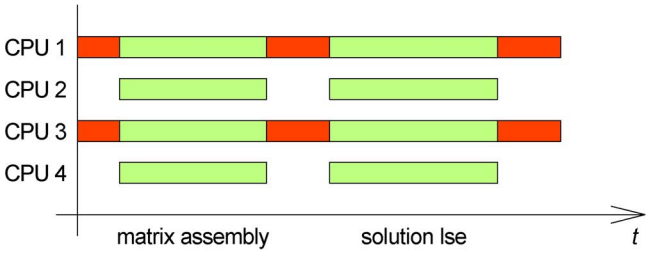

Fig. 4. Processes and threads in the case of Cluster OpenMP.

A standard computer consists of a multicore CPU with a shared memory. This means that all processors access the same memory (Fig. 1). Data in one memory cell is accessible to all processors. The software is executed in a single process, which is split into multiple threads. Normally, one thread per processor is chosen. Multithreading is supported by most compilers with the well-known OpenMP standard. OpenMP is independent of platform and operating system. Only time-consuming parts of the software are parallelized. In the case of a BEM in combination with the FMM, these are assembly of the near-field matrix and matrix vector products during solution of the linear system of equations (Fig. 2). Matrix assembly means in practice that singular and nearly singular integrals are computed. Computing these integrals for two elements is completely independent. Care must only be taken during storage of the results in the matrix. Nodes belong to multiple elements. Hence, data of one node is computed by multiple element integrations. To avoid memory writing conflicts, only one thread is allowed to write into the matrix at a time. During the matrix vector product, the FMM operations are distributed among the threads. In many cases, multiple threads have to read the same data from memory. Then, data must be ensured to be up-to-date. More difficult is to avoid writing conflicts and data race during the storage of the results. Since the storage size of the output vector is relatively small in comparison to the remaining memory requirements, an output vector for each thread is used and these vectors are added at the end of the parallel section. Then all threads are completely independent, too. A big advantage of OpenMP is that load distribution is done dynamically during runtime. The threads are created and managed by the compiler and the operating system. OpenMP is, therefore, easy to implement.

If a computer cluster is used, parallelization is more complicated. CPUs and memory are distributed among the computing nodes (Fig. 3). It is necessary to start a process on each node (Fig. 4). Since it is a distributed memory system, which is often interconnected via a local area network (LAN), data must be exchanged between the processes. Of course, this interchange causes an overhead. Furthermore, a LAN is very slow in comparison to the bus in a computer. Hence, communication between the nodes should be kept to a minimum.

A classical approach to realize parallelization with multiple processes on a distributed memory computer is the MPI standard [8]. MPI is very powerful and allows the software developer full control over the execution of the program. Nevertheless, MPI has some disadvantages. The complete program must be parallelized and not only time-consuming parts. In practice, this is often ignored and some parts of the program are executed on all nodes with the same data. Then, conflicts in file access or the like must be avoided. The main disadvantage is that data exchange between the processes must be implemented by the software developer and load balance must be provided, too. Especially, this point is, in many cases, very difficult.

In May 2006, Intel Corporation introduced the new Cluster OpenMP standard. It is based on OpenMP and supports distributed memory computers. In a first step, it appears to the software developer in the same way as classical OpenMP. One process is started and threads are created in the parallel section of the program. These threads are distributed among the computing nodes. For this purpose, processes on the nodes are created by MPI calls. However, this is hidden by the compiler. All threads see the same virtual shared memory. For efficiency purposes, data must be declared sharable or not. If a thread writes to a memory page, all other threads on the other nodes are noticed about this action. If a thread reads a memory page, it is checked to be up-to-date and data is exchanged where required. To keep data transfer to a minimum, it is necessary to read and write data in a very ordered way.

In principle, load distribution can be done dynamically during runtime by the compiler. However, this often results in a large data transfer between the threads. A better strategy is to implement a load distribution strategy. Data should be kept local. That

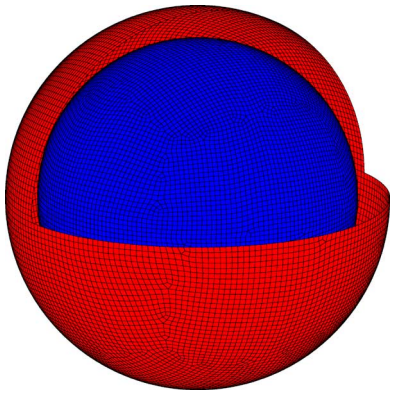

Fig. 5. Discretized model of a coated conducting sphere.

means, data should be read and written on the same node. A goal is to store data in such a way in memory that data of a memory page belongs only to one thread. Of course, if such a strategy is applied, simplicity of Cluster OpenMP is a little bit lost and programming is similar to a MPI approach.

Cluster OpenMP is used to distribute the threads among the computing nodes. If a node has multiple processors or multicore CPUs, classical OpenMP distributes the load of a Cluster OpenMP thread with classical threads among the processors of a node.

III. Numerical Results

Two numerical examples were studied to analyze the efficiency of OpenMP and Cluster OpenMP in the context of a BEM with the FMM. The first example is an electrostatic field problem. A conducting sphere, which is coated with a dielectric, is considered (Fig. 5). The configuration of this example is very simple and an analytical solution is available to test the accuracy of the method. Of course, parallelization makes sense only for large problems. That is why the sphere was discretized with 27 228 second-order, quadrilateral elements. An indirect BEM formulation was applied to solve this electrostatic field problem. A linear system of equations with 81 688 unknowns is obtained. A classical BEM approach would result in memory requirements of 50 GB. The FMM compresses the matrix to 500 MB. The potential along a radial was computed. It shows excellent agreement with the analytical solution (Fig. 6).

Steady currents inside a printed circuit board (PCB) are investigated as second numerical example (Fig. 7). A dual-layer printed circuit board is equipped with two connectors, which are loaded with several hundred amperes (Fig. 8). Typical fields of application for such PCBs are agricultural machines. The aim of this investigation was to get the influence of the connection between connector and the conducting layers of the board. For a numerical solution, an indirect BEM formulation was chosen. Then, the current flow field inside the conductors and the electric field in the substrate and the surrounding (Fig. 9) are obtained within a single solution step. The PCB was discretized with second-order quadrilateral elements, too. The resulting linear system of equations for the 86453 unknowns would require 56 GB with a classical BEM. The application of the fast multipole method reduces this amount to 900 MB.

Two different computers were used for the numerical solution of both examples. One computer was equipped with 16 Intel Itanium 2 processors with a clock speed of 1.3 GHz. The computer

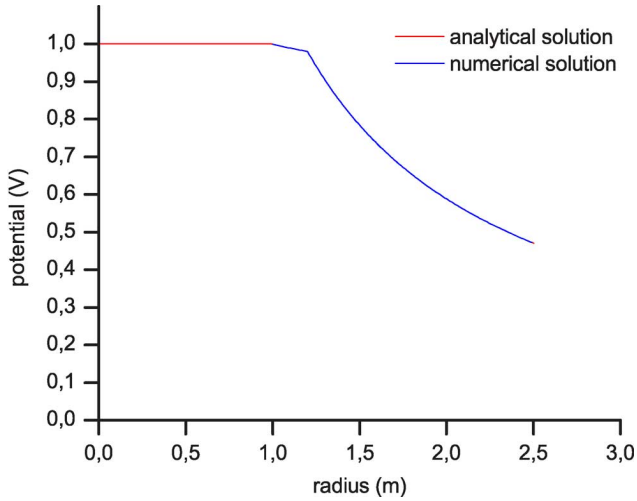

Fig. 6. Potential along a radial line compared with the analytical solution.

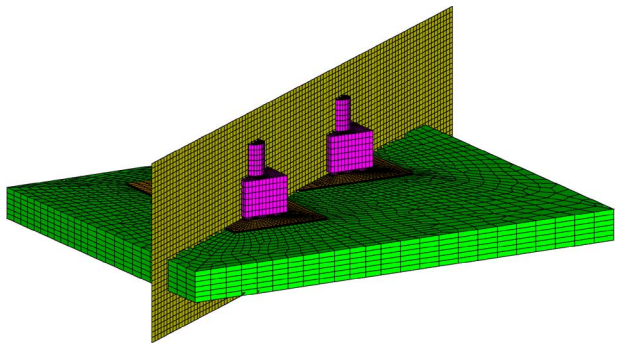

Fig. 7. Dual-layer printed circuit board with two connectors.

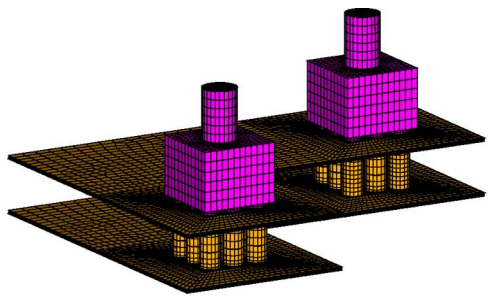

Fig. 8. All electrical components of the printed circuit board in Fig. 7.

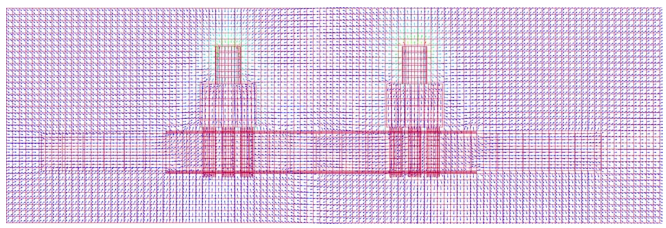

Fig. 9. Electric field strength in the plane, which is depicted in Fig. 7.

cluster consists of eight computing nodes. Each node has two AMD Opteron processors with a clock speed of 2.2 GHz. To determine the speedup of parallelization, first a serial program run was executed (Table I).

Matrix assembly consists of a huge number of floating point operations. Its CPU time is mainly influenced by the power of the CPU. During the solution of the linear system of equations,

TABLE I
CPU TIME OF SERIAL PROGRAM EXECUTION

	Itanium		Opteron	
	Matrix assembly	Solution lse	Matrix assembly	Solution lse
Sphere	1466 s	3765 s	791 s	1603 s
PCB	4275 s	6759 s	2323 s	3067 s

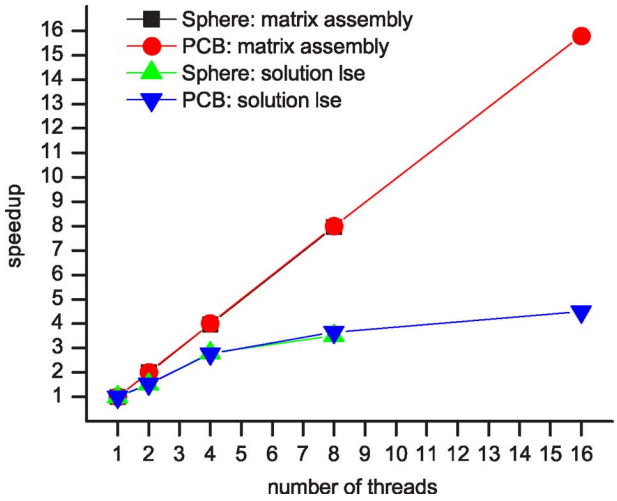

Fig. 10. Speedup of matrix assembly and solution of linear system of equations in the case of OpenMP.

the computational costs are mainly caused by matrix vectors products. A huge amount of data must be transferred between memory and processor. Hence, the speed of the bus between memory and processor is very important. Vectorization, which is standard in modern CPUs, is even not possible for many FMM operations.

In Fig. 10, the speedup on the shared memory computer is depicted. The speedup for matrix assembly is excellent. Even for 16 threads the speedup is 15.8. Unfortunately, the speedup for solution of the system of linear equations is much worse. Memory bandwidth is only one factor. To avoid memory conflicts and data races, threads have to wait at some points in the FMM algorithm for each other. Dense vector operations like the near-field matrix are very efficient and parallelization is very efficient, too. However, to reduce the total computational costs, memory access to the coefficients of the FMM must be improved, for instance by a new version of the FMM algorithm, which is trimmed to parallel execution.

Since a distributed memory computer causes communication overhead between the virtual threads, the efficiency of Cluster OpenMP must be worse than the efficiency of OpenMP. As can be seen from Fig. 11, it is the case for both examples. Even in the case of matrix assembly, the speedup for four threads, which run on two computing nodes, is only three. A reason is that after each writing memory access the other computing node is informed about this change in the virtual shared memory. The speedup for the solution of the linear system of equations is good in comparison to OpenMP. One reason, therefore, is that the FMM algorithm was slightly modified. Some fast operations were done in each thread. Hence, data is locally available and must not be transferred between the computing nodes.

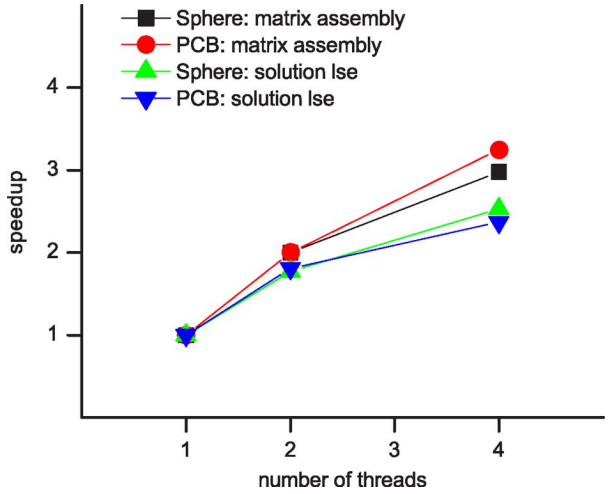

Fig. 11. Speedup of matrix assembly and solution of linear system of equations in the case of Cluster OpenMP.

IV. CONCLUSION

Today, parallelization is indispensable to use modern computers with multicore CPUs efficiently. A very good approach is then multithreading based on the OpenMP standard. Classical dense vector operations like near-field matrix assembly show an excellent speedup. Compression techniques like the fast multipole method have some problems. The algorithm itself is so efficient that CPUs cannot be used at their best. In addition, an efficient parallelization is difficult. A way out of this dilemma would be a new algorithm, which takes parallelization and memory access into account. Cluster OpenMP is very promising. Particularly, parallelization on a computer cluster is easy to implement. Both OpenMP and Cluster OpenMP will be a good choice for an easy and efficient parallelization of software. Especially the combination of parallelization and a fast and efficient method like the BEM with the FMM is very attractive.

REFERENCES

[1] M. Bebendorf, "Approximation of boundary element matrices," *Numer. Math.*, vol. 4, pp. 565–589, 2000.
[2] L. Greengard and V. Rokhlin, *The Rapid Evaluation of Potential Fields in Three Dimensions*, C. Anderson and C. Greengard, Eds. Berlin, Germany: Springer, 1987, vol. 1360, Lecture Notes Math., pp. 121–141.
[3] W. Hafla, A. Buchau, and W. M. Rucker, "Accuracy improvement in nonlinear magnetostatic field computations with integral equation method and indirect scalar potential formulations," *COMPEL*, vol. 25, no. 3, pp. 565–571, 2006.
[4] OpenMP Architecture Board, OpenMP C and C++ Application Program Interface, ver. 2.0 [Online]. Available: http://www.openmp.org, 2002
[5] J. P. Hoeflinger, "Extending OpenMP to clusters," White Paper, Intel Corporation, 2006.
[6] A. Buchau, W. Hafla, and W. M. Rucker, "Accuracy investigations of boundary element methods for the solution of Laplace equations," *IEEE Trans. Magn.*, vol. 43, no. 4, pp. 1225–1228, Apr. 2007.
[7] A. Buchau, W. Rieger, and W. M. Rucker, "Fast field computations with the fast multipole method," *COMPEL*, vol. 20, no. 2, pp. 547–561, 2001.
[8] Message Passing Interface Forum, MPI: A Message Passing Interface Standard [Online]. Available: http://www.mpi-forum.org June 12, 1995

Manuscript received June 24, 2007. Corresponding author: A. Buchau (e-mail: andre.buchau@ite.uni-stuttgart.de).