Parallelization of BEM Electrostatic Solvers Using a PC Cluster

Hua-Kun Tang and Yao-Joe Yang
Department of Mechanical Engineering
National Taiwan University,
No. 1 Roosevelt Rd. Sec. 4, Taipei, Taiwan, ROC
pfldbl@mems.me.ntu.edu.tw; yjy@ntu.edu.tw

ABSTRACT

In this paper, the application of parallel-processing techniques for electrostatic calculation is presented. Evaluating the pre-conditioner used for inverting the system matrix of a boundary-element-method (BEM) electrostatic solver (Fastlap [1]) is one of the most time consuming steps during the whole computation process. Therefore, we apply parallelization techniques to accelerate this step using a PC cluster [2,3], with minimum modification on the original source code. The basic idea is to parallelize the inversions of the sub-matrices which are extracted from the original system matrices. Then the inverted sub-matrices will be assembled into the pre-conditioner. The efficiency of the parallelization is also compared between different methods of distributing sub-matrices for inverting. The preliminary results for a testing sphere structure show that the computing time can be effectively reduced using the parallel processing without any compromise in accuracy. The speed-up is close to 4 by using a four-node cluster.

Keywords: BEM, pre-conditioner, fast multipole method, PC cluster, parallelization, MPI

1 INTRODUCTION

Electrostatic computations, such as capacitance and electrostatic field/force calculations, are vital for MEMS as well as semiconductor device modeling. Since the computational domains for electrostatic problems are usually the open space, the boundary element method is widely used to implement electrostatic solvers [4-7]. Various BEM electrostatic solvers have been developed and commercialized during the past few years. Most of these solvers employ the *multipole* or the *FFT* algorithms [1,6,7,8] for constructing sparse-like system matrices, and generate the corresponding pre-conditioner matrices for accelerating the inversion of the system matrices using iterative methods [9,10].

There have been many works about parallelizing *FFT* or *multipole* algorithm. These parallelization techniques focus on mapping cubes to processors [11], or parallelizing pre-corrected *FFT* algorithm [12]. Because of the complicated relationship between cubes, mapping cubes to different processors usually results in very intensive communication overhead, and requires complicated coding and algorithm. Furthermore, the computational bottleneck often occurs at the stage of generating the pre-conditioning matrix, rather than at the stage of generating system matrix. Instead of applying parallelization techniques for the *multipole* or the *FFT* algorithm, in this work, we focus on parallelizing the process of creating the pre-conditioners that will be used for accelerating the inversion of the system matrices generated by the *multipole* and the *FFT* algorithms.

Both the *multipole* and the *FFT* algorithms require partitioning the whole computational domain into many small sub-domains during the process of constructing the system matrices. The performance of creating the pre-conditioner heavily depends on the total number of the partitioned sub-domains. Typically, if the whole computational domain is divided into a smaller number of sub-domains (i.e., a lower partitioning depth), the larger the sub-matrices which is required to be inverted for generating the pre-conditioner matrix. Therefore, the overall computational bottleneck frequently occurs at the step of creating the pre-conditioner with relatively small partitioning depths. On the other hand, with a higher partitioning depth, the efficiency for creating the pre-conditioner usually increases significantly. However, higher partitioning depths usually result in worse accuracy because some of the BEM panels might be larger than (or comparable to) the size of the smallest sub-domains.

In this work, we develop a methodology that applies a parallel computing technique to efficiently create the pre-conditioner. Especially, the advantage of this methodology is very effective for low partitioning-depth cases. We modify the BEM electrostatic solver Fastlap with MPI functionality [13] for parallelization. The computation is performed using a PC cluster running Linux operation system. The preliminary results of a testing sphere structure show that the computing time can be effectively reduced using the parallel processing without any compromise in accuracy.

2 MATHEMATICAL PRELIMINARIES

2.1 Problem Formulation using BEM

Consider a system of m ideal conductors in free space. The capacitance of m conductors can be formulated as a $m \times m$ symmetric matrix C. The value of C_{ij} is the charge of the *i-th* conductor with the *j-th* conductor connected with 1 volt, and the rest of the conductors are

grounded. The charge on each conductor can be determined by solving the integral equation [7]:

$$\psi(x) = \int_{surfaces} \sigma(x') \frac{1}{4\pi\varepsilon_0 \|x - x'\|} da', x \in surfaces. \quad (1)$$

where $\psi(x)$ is the well-known surface potential, σ is the surface charge density, da' is incremental conductor surface area, $x, x' \in R^3$, and $\|x\|$ is the usual Euclidean length of x given by $\sqrt{x_1^2 + x_2^2 + x_3^2}$. By using piece-wise constant collocation scheme, Equation 1 can be formulated as

$$Pq = \overline{p} \quad (2)$$

where $P \in R^{n \times n}$, q is the vector of panel charges, $\overline{p} \in R^n$ is the vector of known panel potentials, and

$$P_{ij} = \frac{1}{a_j} \int_{panel_j} \frac{1}{4\pi\varepsilon_0 \|x_i - x'\|} da' \quad (3)$$

where x_i is the center of the *i-th* panel and a_j is the area of the *j-th* panel. The surface charge density σ can be obtained by solving the inversion matrix of P.

2.2 Multipole Algorithm and Pre-conditioner

The total cost to form the dense matrix P in Equation 3 is $O(N^2)$. The key in Fastlap is to use multipole acceleration method [7]. The multipole algorithm approximates the potential ψ contributed by the charge of a collection of distant panels using a multipole expansion, based on the assumption that the effects of distant panels are relatively insignificant. Therefore, it is unnecessary to find P_{ij} explicitly for distant panels. Besides, the cost of solving P^{-1} by Gauss elimination is $O(N^3)$ because P is a dense matrix, which becomes computationally intractable if the number of panels exceeds several hundreds. Therefore, Fastlap solve the inversion problem using iterative method with pre-conditioner matrix C. By multiplying pre-conditioner C in Equation 2,

$$CPq = C\overline{p} \quad (4)$$

The pre-conditioner matrix C is computed by inverting a sequence of sub-matrices extracted from the P matrix, as shown in Figure 1. Typically, the C matrix is close to P^{-1} [7]. The size of each sub-matrix is strongly dependent on the partitioning depth and panel location. Since the cost of inverting these sub-matrices is $O(N_S{}^3)$, where N_S is the size of each sub-matrix, the overall computational cost will be detrimentally affected if some of N_S are relatively large. On the other hand, as the partitioning depth increases, the efficiency of inverting sub-matrices can be increased, while the error of iteratively solving Equation 4 is also increases.

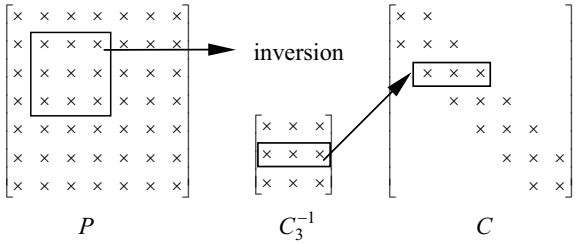

Figure 1 Forming the pre-conditioner matrix C from system matrix P. The matrix C will approach to P^{-1}.

3 SIMULATION METHOD

3.1 Modeling

We parallelize the BEM electrostatic solver Fastlap [1] using the functions of the MPI library. The simulation case in this work is the same as the example case provided in the Fastlap tutorial. Figure 2 shows the schematic of the model. The dotted line in Figure 2(a) indicates the sphere with the prescribed Dirichlet-type boundary conditions. Figure 2(b) shows one of the simulated 3-D BEM models divided by equal subdivision of the polar and azimuthal angles. The solution is known in closed form. Figure 3 shows the relationship between the partitioning depth and the maximum error, for the cases of four different numbers of panels.

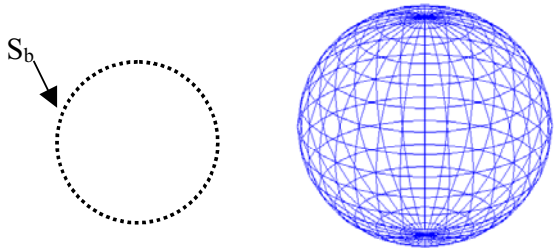

Figure 2 (a) Boundary conditions of the model for our work. (b) 3-D boundary element mesh plot of the ball simulated.

The partitioning depth is an integer that specify to which the computational domain will be hierarchically decomposed. As the partitioning depth equal to 1, there is only one sub-matrix (i.e., equal to the original *P*), which implies that parallelization will not be applicable. Therefore, these curves start with partitioning depth of 2. As shown in the figure, the errors decrease with the partitioning depth.

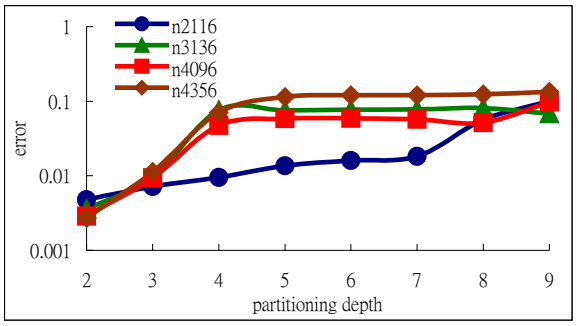

Figure 3 Maximum errors vs. partitioning depth for different numbers of panels.

3.2 Parallel Approach

The pre-conditioner is in fact a collection of the inverted sub-matrices that are originally extracted from the system matrix. Inverting those sub-matrices is the most expensive step in the process of creating pre-conditioner. Using the MPICH communication library [13], the Fastlap is accelerated by parallelizing the inversion calculations of those sub-matrices. Figure 4 shows the schematic of a four-node PC cluster used in this work. One of the nodes also serves as the cluster server that is responsible for distributing sub-matrix to other nodes.

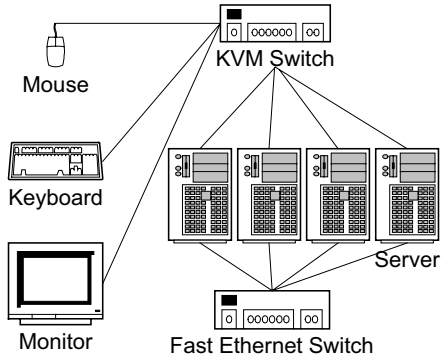

Figure 4 The schematic of the cluster: four nodes with a server for submitting jobs to the other nodes.

Since the sizes of sub-matrices are usually different, the computing time for inverting those sub-matrices differs significantly. In order to achieve load balance, we rearrange the order of those sub-matrices according to their sizes before distributing to each node. Figure 5 shows that for a system with 6 sub-matrices under different parallelization conditions. Figure 5(a) indicates that only one processor (P0) is used to compute the sub-matrices (i.e., without parallelization). The total required computational cost is 12 units. Figure 5(b) indicates that if the sub-matrices are distributed to the two nodes without rearranging their orders, the required computing time will not be optimal. Figure 5(c) shows the most ideal case when the sub-matrices are rearranged and equally distributed, the performance can reach the maximum condition.

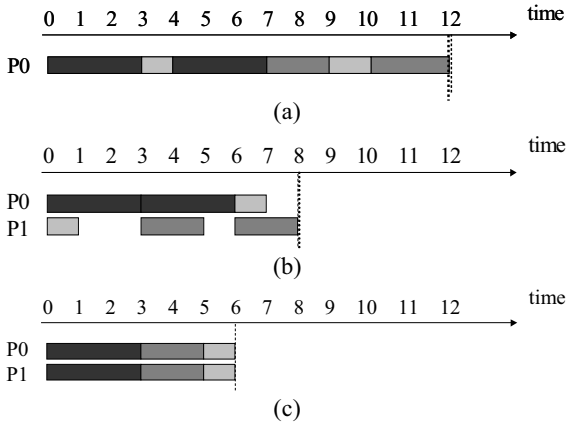

Figure 5 (a) There are six sub-matrices of different sizes (different required computing times). It costs 12 time units by one processor. (b) and (c) show the total computing time by two nodes without and with rearranging the distributing sequences.

4 RESULT AND DISCUSSION

Table 1 lists the computing times for different depths and different numbers of nodes. As indicated in Figure 3, although the cases with lowest d (d=2) provide the most accurate results, they require the largest computing times. This table also shows that our parallelized code can effectively increase the computation efficiency. Figure 6 and 7 show the speed-up and the parallelization efficiency in different partitioning depth, respectively. They are defined as

$$Speedup = \frac{T(1)}{T(N_p)} \tag{5}$$

$$Efficiency = \frac{Speed-up}{N_p} \times 100\% \tag{6}$$

where N_P is the number of processors, $T(N_P)$ and $T(1)$ is the time for the parallel analysis calculated by N_P processors and by one processor, respectively. Figure 6 demonstrates that the speed-up is almost 4 for a 4-node cluster with d=2. Figure 7 shows that parallelization efficiency increases as the partitioning depth decreases. Notice that when d becomes higher, speed-up and efficiency both decrease rapidly. This is because the calculation time of small sub-matrix for cases of higher depths is faster than data transmission time through the local network of the PC cluster. Furthermore, the error will also increases significantly with the partitioning depth. It also has to be emphasized that the parallelization technique not only effectively reduces the computing time, but also provides the results without any compromise in accuracy.

Table 2 shows that the calculated results by the parallelized and un-paralleled (original) Fastlap are identical.

Computing time (n3136)			
partitioning depth	***d*=2**	***d*=3**	***d*=4**
1 node	1141.6 s	45.88 s	5.639 s
2 nodes	974.87 s	41.63 s	5.667 s
3 nodes	475.82 s	25.30 s	5.736 s
4 nodes	315.46 s	21.02 s	5.821 s

Table 1 The computing time calculated by original Fastlap and our paralleled Fastlap. Although the computing time decreases as *d* increases, the error increases with *d*.

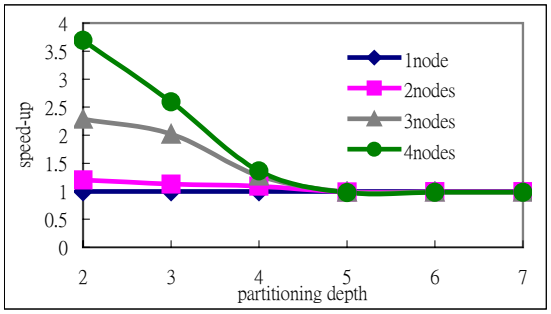

Figure 6 Speed-up in different number of nodes derived by different partitioning depth. When the partitioning depth decreases, the higher speed-up we get. There are 4096 panels here.

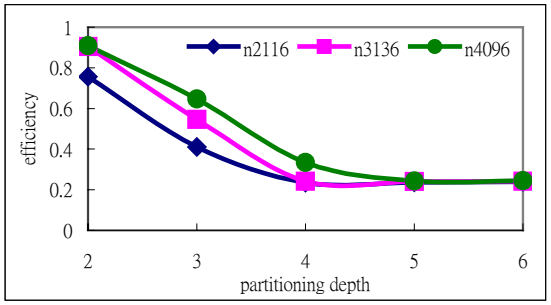

Figure 7 This figure shows the efficiency in different partitioning depth. Efficiency increases when partitioning depth decreases.

5 CONCLUSION

In this paper, we present the application of parallel-processing techniques for electrostatic BEM solvers that employ the *multipole* or *FFT* algorithms. The computations are performed on a four-node PC cluster. In order to efficiently generate the pre-conditioner matrix of BEM solvers, the sub-matrices, which are extracted from the original BEM system matrix, are distributed to the nodes of the PC cluster for inversion calculating. The comparison of different sequences of distributing sub-matrices for optimal load balance is also studied. The preliminary results show that the computing time can be effectively reduced without any compromise in accuracy, and also show that the best speed-up is about 3.7 on a 4-node cluster.

Results by the Fastlap (d=2)

Number of panel	512	1024	2116
maximum error	0.010746	0.008536	0.004781
average error	0.002627	0.002309	0.001137

Results by the paralleled Fastlap

Number of panel	512	1024	2116
maximum error	0.010746	0.008536	0.004781
average error	0.002627	0.002309	0.001137

Table 2 Maximum error and average error calculated by fastlap and our work. We can see that paralleled fastlap have the same result as it was.

ACKNOWLEDGMENT

The author would like to thank Professor W. F. Wu for supporting this work and Dr. D. W. Lin for helping construct the PC cluster.

REFERENCES

[1] T. Korsmeyer, K. Nabors, and J. White, Fastlap Version 2.0, Massachusetts Institute of Technology, Cambridge, MA, 1996

[2] T. Anderson, D. Culler, and D. Patterson, "A Case for Networks of Workstations (NOW)," IEEE Micro, Vol. 15, No. 1, Feb. 1995, pp. 54-64.

[3] J. Bruck, D. Dolev, C.-T. Ho, M.-C. Rosu, and R. Strong, "Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations," Journal of Parallel and Distributed Computing, Jan. 1997, pp. 19-34.

[4] W. S. Hall, The Boundary Element Method, Kluwer Academic Publishers, Boston, 1994.

[5] K. E. Atkinson, and M. Golberg, "A Survey of Boundary Integral Equation Methods for the Numerical Solution of Laplace's Equation in Three Dimensions," Numerical Solution of Integral Equations, Plenum Press, New York, 1990, pp. 1-34.

[6] CoventorWare2001 Reference Manual, Coventor, Inc., 2001

[7] K. S. Nabors, Efficient Three-Dimensional Capacitance Calculation, Ph.D. dissertation, Massachusetts Institute of Technology, May 1993.

[8] J. R. Phillips and J. K. White, "A Precorrected FFT Method for Electrostatic Analysis of Complicated 3-D Structures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 10, Oct. 1997, pp. 1059-1072.

[9] K. Nabors, F. T. Korsmeyer, F. T. Leighton and J. White, "Preconditioned, Adaptive, Multipole-accelerated Iterative Methods for Three-dimensional First-kind Integral Equations of Potential Theory," SIAM J. Sci. Comp., vol. 15, No. 3, May 1994, pp. 713-735.

[10] Y. Saad and M. H. Schultz, "GMRES: A Generalized Minimum Algorithm for Solving Nonsymmetric Linear Systems," SIAM J. Sci. Stat. Comp., vol. 7, July 1986, pp. 856-869.

[11] Y. Yuan and P. Banerjee, "A Parallel Implementation of A Fast Multipole Based 3-D Capacitance Extraction Program on Distributed Memory Multicomputers," IPDPS 2000, May. 2000, pp. 323-330.

[12] L. W. Li, Y. J. Wang and E. P. Li, "MPI-Based Parallelized Precorrected FFT Algorithm for Analyzing Scattering by Arbitrarily Shaped Three-Dimensional Objects," Journal of Electromagnetic Waves and Applications, vol. 17, no. 10, Oct. 2003, pp. 1489-1491.

[13] W. Gropp, E. Lusk, N. Doss and A. Skjellum, "A High-performance, Portable Implementation of The MPI Message Passing Interface Standard," parallel computing, 22 (6) Sep. 1996, pp.789-828.