A Parallel Implementation of A Fast Multipole Based 3-D Capacitance Extraction Program on Distributed Memory Multicomputers *

Yanhong Yuan and Prithviraj Banerjee
Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208
{yuanyh, banerjee}@ece.nwu.edu

Abstract

Very fast and accurate 3-D capacitance extraction is essential for interconnect optimization in ultra deep submicro designs (UDSM). Parallel processing provides an approach to reducing the simulation turn-around time. This paper examines the parallelization of the well known fast multipole based 3-D capacitance extraction program FASTCAP [4], which employs new preconditioning and adaptive techniques. To account for the complicated data dependencies in the unstructured problems, we propose a generalized cost function model, which can be used to accurately measure the workload associated with each cube in the hierarchy. We then present two adaptive partitioning schemes, combined with efficient communication mechanisms with bounded buffer size, to reduce the parallel processing overhead. The overall load balance is achieved through balancing the load at each level of the multipole computation. We report detailed performance results using a variety of standard benchmarks on 3-D capacitance extraction, on an IBM SP2.

1. Introduction

Accurate estimation of the capacitances in complicated three dimensional (3-D) interconnects is becoming increasingly important for determining the final circuit speeds or functionality in the ultra deep sub-micron design (UDSM) of integrated circuits. Examples of complicated 3-D structures for which capacitances strongly affect performance are illustrated in Figure 1 [4, 5]. Hierarchical approximation methods, such as the fast multipole algorithm [1] and Appel's algorithm [2] have been shown to be very effective and have been used in several extractors solving for the capacitance matrix of complicated 3-D structures [4, 7]. Among them, the fast multipole based 3-D capacitance extraction program FASTCAP [4] has been most widely used in both industry and academia for benchmark purposes. Although 3-D simulation tools can help designers find signal integrity problems, even the fastest of these tools running on a single scientific workstation are too slow to allow a designer to quickly investigate a variety of conductor layouts. Reducing the run time of the extraction tools is crucial in UDSM design. Parallel processing offers a tool for effectively speeding up the computation.

Parallel formulations of hierarchical methods have been extensively studied in the context of *particle simulations*. If particle distribution is uniform across the domain, it is easy to balance the load and optimize communication by partitioning the domain [12, 13, 14]. For non-uniform distributions, these objectives are hard to achieve, due to the highly irregular nature of both computation and communication. Works on unstructured problems can be found in [15, 16, 17]. Though implementing the adaptive algorithms for the non-uniform problems on a shared-memory parallel computer is quite straightforward, an efficient distributed-memory implementation proved to be quite challenging.

In 3-D capacitance extraction, we encounter highly irregular panel charge distributions. This paper presents our work on the parallelization of the preconditioned, adaptive, fast multipole accelerated 3-D capacitance extraction program FASTCAP [4]. To compute the costs of cubes for load balancing, we propose a *generalized cost function model.* We then present two *adaptive partitioning schemes* based on the model, i.e. adaptive cyclic mapping and adaptive block mapping. We show that the total load can be well balanced among processors by adaptively mapping cubes at every hierarchical level. The communication mechanisms under the adaptive partitioning schemes are characterized by efficient *collective communication operations* with *bounded buffer size*. This is in contrast to the very fine grain communication strategy based on send-receive pairs using small messages. Another advantage of our partitioning and com-

*This research was supported in part by the Advanced Research Projects Agency under contract DAA-H04-94-G0273 administered by the Army Research Office.

munication strategy is that, it involves minimum changes to the original serial code. We also derive an upper bound for the parallel formulation. The parallel algorithm has been developed using the Message Passing Interface(MPI) so that it is $portable$ on a variety of parallel platforms. We report detailed experimental results on an IBM SP2.

Prior reports of parallel algorithms for 3-D capacitance extraction will be reviewed in Section 2. Section 3 presents background on 3-D capacitance extraction and algorithms used in FASTCAP. The partitioning and communication strategies are discussed in Section 4. The parallel algorithms are presented in Section 5. Experimental results are reported in Section 6. Section 7 gives the conclusions.

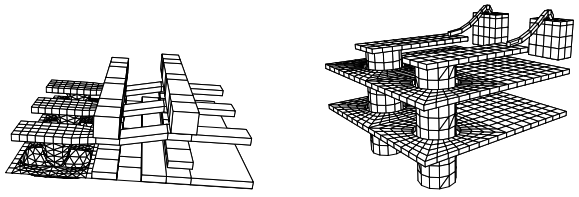
2. Related Work

Wang, Yuan, and Wu [9] developed a parallel fast multipole accelerated capacitance extraction program on a Transputer network. It was based on the adaptive scheme of recursive subdivision of dense regions [8]. There was no preconditioner for the multipole algorithm. The approach to load balancing was to map cubes of different sizes to processors. However, the cost associated with each cube was simply assumed to be one, and the interactions with other cubes were not counted. Due to that, serious imbalance may happen at the computation of a coarser level. Typical result included a speedup of 5 on 8 processors.

Aluru, Nadkarni and White [10] presented a parallel capacitance extraction program based on the precorrected Fast Fourier Transform (FFT) on an IBM SP2. It used a single decomposition which fits only one step of the precorrected FFT algorithm, i.e., the convolution step, and the decomposition algorithm simply allocates equal number of grid planes to each processor. Consequently, when the convolution algorithm is about 30% of the total CPU time, the parallel efficiencies are only about 40%. The overall speedups are about 3 to 5 on 8 processors.

The parallel algorithm for the direct boundary element method (BEM) based capacitance extraction using adaptive Gaussian quadrature was studied by Yuan and Banerjee [11]. For sequential computing, the direct BEM method results in dense linear system and generally requires more CPU time and memory than the hierarchical methods such as the fast multipole method. By recursive ordering of elements in the partition and mapping boundary elements to processors, a general speedup of 13 on 16 processors was reported on an IBM SP2.

A *generalized cost function model* for accurately measuring the computation cost associated with each cube is presented in this paper. This model is an extension and generalization of the methods used in [15]. However, there is a major difference in computing the cost of the internal cubes (or cells). In Singh's approach [15], the cost of an internal cube is not just the sum of costs of all $leaves$ within it, but the sum of the costs of all cubes (leaf or internal) within it (all descendents) plus its own cost. In our cost model, the cost of an internal cube is the sum of all the *interaction costs* of the cube's *interaction cubes*. This is primarily due to the reason that, in the fast multipole method, all the computations are performed level by level. A cube at the current computation level only interacts with its *interaction cubes*, which are not necessarily all the descendents of that cube. Therefore, there is no need to transfer the cost of all the descendents of a cube to that cube at current computation level. In addition to that, our model is more flexible to balance the load at each level of the multipole computation, therefore can achieve well balanced overall load balance. With some modifications, the cost model presented in this paper is also applicable to balance the load in other well known hierarchical N-body methods, such as the Barnes-Hut method [3] and the Appel's algorithm [2, 7].



(a) A DRAM cell (b) A via structure

Figure 1. Examples of 3-D interconnect structures.

3. Background

3.1. The Capacitance Problem

The capacitance of an m-conductor geometry is represented by an $m \times m$ symmetric matrix C. The j-th column of the capacitance matrix is determined by finding the surface charges on each conductor by raising conductor j to one volt and grounding the rest. The charge on each conductor can be determined by solving the integral equation [18]

$$\psi(x) = \int_{surfaces} \sigma(x') \frac{1}{4\pi\epsilon_0 \|x - x'\|} da', \qquad x \in surfaces, \tag{1}$$

where $\psi(x)$ is the known conductor surface potential, da' is the incremental conductor surface area, x, $x' \in \mathbf{R}^3$, and $\|x\|$ is the usual Euclidean length of x given by $\sqrt{x_1^2 + x_2^2 + x_3^2}$.

A standard approach to numerically solving (1) for σ is to use a piece-wise constant collocation scheme [19]. The

result is a dense linear system,

$$Pq = \overline{p}, \tag{2}$$

where $P \in \mathbf{R}^{N \times N}$, q is the vector of panel charges, $\overline{p} \in \mathbf{R}^N$ is the vector of known panel potentials. When an iterative algorithm such as GMRES [20] is used to solve (2), the major cost of the algorithm is $O(N^2)$ operations required to form the dense matrix P and $O(N^2)$ operations to compute the dense matrix-vector products for each GMRES iteration.

3.2. Review of Sequential Algorithms in FASTCAP

The key in FASTCAP is to use the fast multipole algorithm [1] for fast evaluation of each matrix-vector product in time $O(N)$ in an iterative solver, such as GMRES. In this section, we briefly review the fast multipole procedure used in FASTCAP. Detailed discussion is available in [4, 5, 6].

The Fast Multipole Algorithm The fast multipole algorithm uses a divide and conquer strategy to cluster particles at various levels, and then uses multipole and local expansions to evaluate the interactions between distant clusters. Direct application of pairwise force law is computed for the interactions in the near field. This reduces the matrix-vector product from $O(N^2)$ to $O(N)$. In FASTCAP, once the 3-D conductor geometry is discretized into panels, a hierarchical oct-tree of cubes is constructed to cluster the panels.

The Adaptive Algorithm FASTCAP proposed a new adaptive scheme for the multipole method. In contrast to the adaptive scheme of non-uniform domain partitioning based on successive subdivision of dense regions [8], the new scheme maintains a uniform domain partitioning. For example, the potential due to panel charges in a cube is always evaluated directly, rather than with a multipole approximation, whenever the number of expansion coefficients would exceed the number of panels. However, this scheme introduces workload imbalance for cubes at the same level, due to the different ratio of the number of coefficients for a cube to the number of terms in a vector.

The Preconditioning Algorithm The GMRES iterative method can be significantly accelerated by preconditioning. Since the system matrix is never explicitly constructed, preconditioners are derived from the hierarchical domain representation. The preconditioner is formed by inverting a sequence of reduced P matrices, one associated with each finest level cube. The computational cost in computing the preconditioner is only in inverting small P^i matrices, which are related to only the panels contained in a cube and its neighbors.

4. Decomposition, Load Balancing and Communication

The fast multipole algorithm works on the unit of *cube* at various levels, rather than on *panels*. Therefore, it is more efficient and natural to map *cubes* to processors, rather than to map *panels*. However, there are strong correlations among cubes of different levels in different computation passes. To account for the complicated data dependencies at each level of computation and the imbalance resulted from the new adaptive scheme, we introduce a *generalized cost model* to measure the computation load associated with a cube at each computation level in the adaptive fast multipole method.

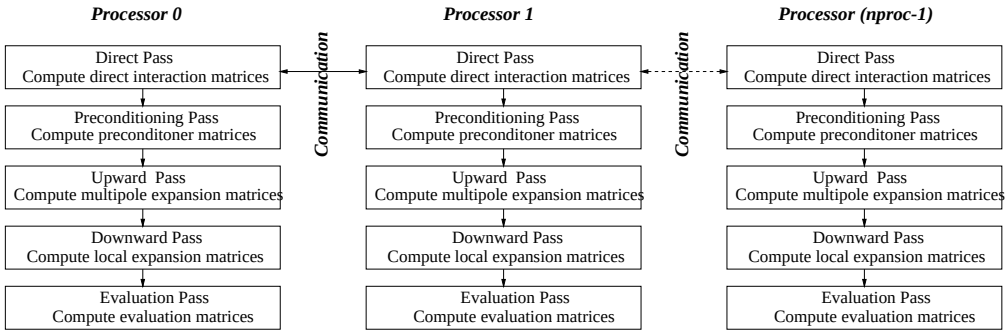
4.1. The Cost Function Model

Let us first introduce several notations. An *interaction cube* j of a cube i is the cube involved in the computation of cube i. For example, in the direct pass, a cube's interaction cubes will be its non-empty neighbors. In the upward pass, a cube's interaction cubes will be its non-empty children. And in the downward pass, a cube's interaction cubes are those non-empty ones in the cube's *interaction list*. The *interaction cost* of cube j is the cost contribution to the computation of cube i. It is also known that, for an r^{th}-order multipole expansion, the number of multipole coefficients is $(r+1)^2$. Furthermore, let k_j be the number of terms in a cube j's charge density expansion coefficients vector or its collocation point potentials vector. Based on these notations, the *total computation cost* or *cost* of a cube i at level l, $c(l,i)$, is defined as follows:

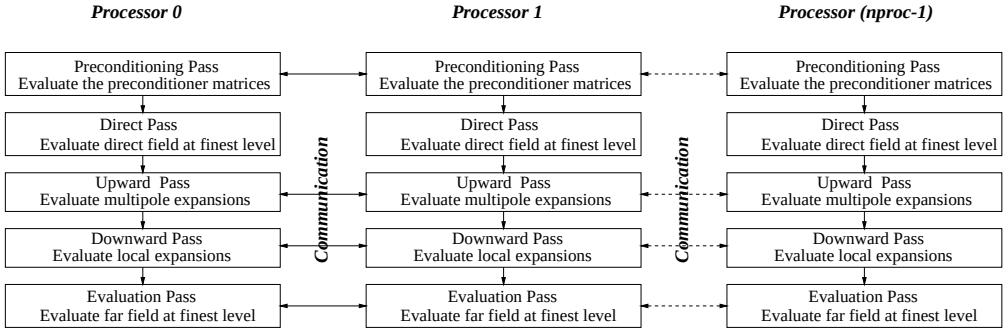
$$c(l,i) = K_d \sum_{j}^{\#nbrs} d_j + K_m \sum_{j}^{\#kids} m_j + K_l \sum_{j}^{\#icubes} l_j. \tag{3}$$

K_d, K_m and K_l are the *weight factors*, associated with the *direct pass*, *upward pass (multipole expansion)* and *downward pass (local expansion)*, respectively. Their values can be set in the range of [0,1]. d_j is the interaction cost of the cube j in cube i's neighbors. m_j is the interaction cost of the cube j in cube i's *adaptive* children. Similarly, l_j is the interaction cost of the cube j in cube i's interaction list. Under the adaptive scheme in FASTCAP, these interaction costs are defined as $k_j/(r+1)^2$. They reflect the computation work required based on the ratio of the number of coefficients for a cube to the number of terms in a vector.

Although $c(l,i)$ is the measurement of the total computation cost of cube i, it is also convenient to obtain the cost of a cube in each pass of the computation. This can be done easily by adjusting the weight factors. In fact, this is the way the cost function $c(l,i)$ is used in our balancing scheme,



(a) Parallel matrix construction.



(b) Parallel matrix-vector product.

Figure 2. An overview of the parallel approach to FASTCAP.

since it can balance the load at each pass, and therefore balance the total workload.

4.2. Adaptive Partitioning and Load Balancing

With the objective of *minimizing changes* to the original code and *maximizing the efficiency*, we present two partitioning schemes based on the above cost model. The strategy is to assign equal amount of computation work to each processor by mapping cubes to processors, at each hierarchical level. In the implementation, the partitioning can be done by either *adaptive cyclic mapping* or *adaptive block mapping* of cubes to processors in each of the *direct list*, *precondition list*, *multipole list* and *local list*, which are used to record the cubes at each hierarchical level in FASTCAP.

Assume there are p processors. Each processor maintains a cost queue, which records the total cost of cubes assigned to each processors. To map cubes to processors, the following procedure is performed simultaneously by all processors. First, the cost of each cube in each list is computed. In the adaptive cyclic mapping, the first p cubes in a list will be assigned to each processor cyclically. For each succeeding cube, it will be assigned to the processor which currently has the least total cost assigned to it. In the adaptive block mapping, the total cost of cubes in a list is computed. Then it is evenly distributed among processors. This is done by traversing the list and assigning the first portion of the cost to processor 0, and the second portion to processor 1, and so on. In FASTCAP implementation, the empty cubes are still generated and stored in the hierarchy, although they don't hold any actual data for field evaluation. These empty cubes will be jumped over in the adaptive mappings.

4.3. Communication with Bounded Buffer Size

The communication mechanisms used are characterized by the collective communication operations, i.e. *one-to-all broadcast* and *all-reduce* operation. This is quite important, especially for the matrix-vector product, which will be evaluated a number of times in GMRES to solve for the full capacitance matrix. Using fine grain send-receive pairs in

such case will often result in deadlock or contention.

For message passing formulations, it is important to bound the size of the buffer for communication. Let N be the number of panel charges, k be the average number of panel charges in each cube at finest level (constant), r be the order of multipole expansions, and p be the number of processors used. We demonstrate that the buffer size on each processor is bounded by $(r+1)^2 N/kp$. All the collective communications are performed either on the potential vector, or on the expansion coefficients at each level, except the one in the construction stage, which communicates on partial direct matrices of a cube's neighbors. The detailed parallel algorithms and analysis will be presented in Section 5.

5. Parallel Algorithms for FASTCAP

In this section, we present the parallel algorithms for matrix construction and matrix-vector using the fast multipole method, with an emphasis on the communication mechanisms. We also analyze the complexity of our parallel algorithms. An overview of the parallel approach to FASTCAP is illustrated in Figure 2. Either cyclic mapping or block mapping can be applied. The same collective communication pattern is applicable to both mappings.

5.1. Matrix Construction

As shown in Figure 2 (a), each pass of the construction stage can be performed in parallel. There is only one communication step, which is required for the preconditioning pass, as the construction of a cube's preconditioning matrix involves the cube's neighbors' direct matrices, which may be stored in a remote processor. One way to exchange the data is to use *send-receive* pairs. However, this very fine grain communication mechanism often resulted in deadlock in our early experiments. To solve this problem, we resort to a collective communication mechanism. A processor will pack the direct matrices of the neighbors of all the cubes it owns. This package will be *broadcast* to all other processors. Experimental results show that this mechanism eliminates the contention and deadlock problem, and is in fact very efficient. We obtain excellent speedups in the matrix construction stage. Let k be the number of panels in a finest level cube, and p be the number of processors. There are at most $O(N/k)$ finest level cubes, each with storage requirement of $O(k^2)$ for its direct matrix, so the buffer size is bounded by $O(kN/p)$ on each processor.

5.2. Matrix-Vector Product

More communication steps are required in the matrix-vector evaluation stage (see Figure 2 (b)). In the upward pass, there are correlations between different levels, since the computation of multipole expansions $\{M_c\}$ at level l requires those at level $l+1$. There are also correlations between the upward and downward pass, since computing the local expansions $\{L_c\}$ requires to convert some cubes' multipole expansions $\{M_c\}$. In addition, there are correlations between the upward and the evaluation pass. In the evaluation pass, the evaluation of the far field is done by evaluating either the local expansions or the multipole expansions (implicit in vector $\{evalvect_c\}$), using the new adaptive scheme. Due to the partitioning of the direct list cubes, the required local and multipole expansions may not be available locally. It is quite complicated to use fine grain *send-receive* pairs due to the complicated correlations. Furthermore, this fine grain scheme will also result in deadlock.

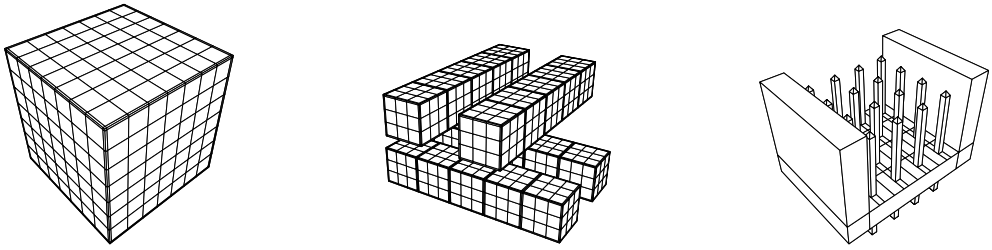
To deal with this problem, we use a scheme which packs all the computed multipole expansions $\{M_c\}$ at a finer level into a buffer and broadcasts it to all other processors, using the same protocol described in the construction stage. After this, the computation at the next (coarser) level can be performed in parallel with local data. Since for an r^{th}-order multipole expansion, the number of multipole coefficients is $(r+1)^2$, the size of the buffer will be bounded by $(r+1)^2 N/kp$ on each processor.

Similar to the upward pass, there are strong correlations between the downward and the evaluation pass. There are also correlations between different levels within the downward pass itself. Therefore, we use a communication scheme similar to the upward pass to exchange the local expansions $\{L_c\}$ among processors at each level. The buffer employed in the upward pass can be reused here. The communication scheme in other passes is quite simple. An *all-reduce* operation is performed in both the preconditioning and the evaluation passes. Note that there is no communication in the direct pass.

It can be demonstrated that the overall complexity for the matrix-vector product in FASTCAP is: $\frac{aN}{p} + b \log_8 p + c(N,p)$, where N is the number of panels, L is the number of levels, p is the number of processors, a and b are constants determined by the floating point speed and the requested precision, and c is a lower order term which includes things like the communication or synchronization overhead.

6. Experimental Results

We have implemented the parallel algorithm for FASTCAP using MPI. The experiments are conducted on an IBM SP2. There are 16 processors in the IBM SP2 distributed memory multiprocessor. Each SP2 node is an IBM RS6000 workstation with 128 MB memory and running at 66 MHz. The available memory to users is about 100 MB. The nodes on the SP2 are interconnected through a high performance



(a) Cubic capacitor (b) 2×2 bus crossing (c) 4×4 connector

Figure 3. Benchmark capacitance problems.

Table 1. Runtimes (seconds) / speedups on an IBM SP2. Star (*) means the problem does not fit on the specified processors. The speedups are estimated by extrapolating the running time to one processor.

problem	via	dram	connector3	connector4
#panels	6120	6129	5944	10096
1 proc	84.9/1.0	340.8/1.0	139.6/1.0	*
2 procs	45.6/1.9	178.0/1.9	72.7/1.9	*
4 procs	25.2/3.4	100.5/3.4	40.7/3.4	143.3/3.5
8 procs	14.5/5.8	56.7/6.0	22.5/6.2	88.2/6.5
16 procs	10.0/8.5	40.0/8.5	15.2/9.2	60.7/9.2

switch. The unidirectional communication bandwidth available at a node is approximately 40 MB/second.

6.1. Solution of Capacitance Problems

The benchmark problems are shown in Figures 1 and 3 [4, 5, 21]. We run FASTCAP in its default mode, that is, two-term multipole expansions and a GMRES with a tolerance of 1%. The default preconditioner is set to use the overlapped preconditioning.

Tables 1 summarizes the performance results on the extraction of the via problem (via), the DRAM cell (dram), a 3×3 backplane connector (connector3) and the 4×4 backplane connector (connector4). Runtimes and speedups under the cyclic mapping are reported on the IBM SP2. The performance of the block mapping and that of the cyclic mapping are quite close.

It can be seen that on the SP2, good speedups are obtained on a variety of problems using various density of discretization. Typical results include a speedup of 6 on 8 processors and about 9 on 16 processors. The speedup on two processors is about 1.9 and on four processors it is about 3.5. These results are better than that reported by the parallel FFT based FASTCAP [10], and also better than that of [9] using a different adaptive scheme, where the typical speedup is about 3 to 5 on 8 processors. Some problems with large number of panels, required either by the complicated problem itself or by the accuracy can not be fit on one and two processors (marked with *), as the program could allocate only about 100 MB of memory per processor on the IBM SP2. However, using the memory scalable parallel algorithm, they can now be solved on multiple processors. (Their speedups are estimated by extrapolating the runtime to one processor.)

6.2. Efficient Hierarchical Matrix-Vector Product

We now discuss the performance of the preconditioned adaptive fast multipole accelerated matrix-vector product, and the GMRES solver implemented on top of it.

Matrix-Vector Product It should be pointed that, in a generic fast multipole algorithm, the multipole and local expansion matrices are computed, combined with the potential evaluation. Therefore, the *actual* matrix-vector product includes both the matrices construction and product evaluation. Nevertheless, in FASTCAP, these two steps are separated. Therefore, to obtain the correct running time for an *actual* matrix-vector product in FASTCAP, we need to extract the runtime for one iteration of product evaluation, and add it to the runtime for computing the matrices at the beginning.

The speedups and efficiencies are depicted in Figure 4. It can be seen that our parallel formulation yields excellent performance. A speedup of about 15 is observed on 16 processors, which reflects an efficiency of 94%. Note that the speedups obtained here are comparable to that of [12], where a parallel fast multipole algorithm for *uniform*

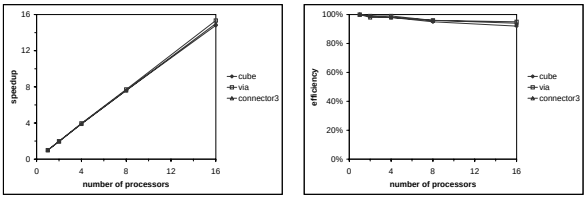


Figure 4. Speedups (left) and efficiencies (right) for one matrix-vector product on an IBM SP2, for three problems: the cubic capacitor (panel = 5400), the via problem (panel = 6120), and the the 3x3 backplane connector (panel = 5944).

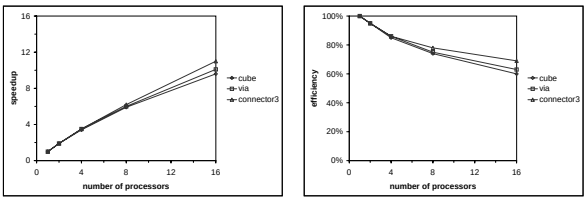


Figure 5. Speedups (left) and efficiencies (right) for the iterative GMRES on an IBM SP2, for the same problems.

particle distribution was implemented on a shared memory machine (Encore Multimax 320).

The GMRES Solver We now present results on the iterative GMRES solver. This corresponds to solving only one row of the capacitance matrix, i.e., computing the coupling capacitances of one conductor with all others. Figure 5 illustrates the corresponding speedup and efficiency curves.

It is shown that a speedup of 6 is obtained on 8 processors, and about 11 on 16 processors. The speedup of solving one column of the capacitance matrix is worse than that of solving one matrix-vector product. This is because a number of matrix-vector products will be evaluated for the convergence in GMRES, therefore more communication operations are involved. Similarly, the speedup of solving the full capacitance matrix is worse than that of solving one row of the matrix, since for each conductor, a GMRES solution is invoked, and there are generally a number of conductors in a 3-D interconnect problem. In all these cases, the matrix construction is performed only once in FASTCAP.

Table 2. Memory allocation (MB) for three benchmarks on the IBM SP2.

#processors	direct	multipole	local	misc	total
cube (panel = 9600)					
1	38.7	0.8	13.1	18.5	71.1
2	21.6	0.4	6.5	17.6	46.1
4	12.2	0.2	3.3	15.4	31.0
8	8.2	0.1	1.7	14.3	24.3
16	6.6	0.05	0.9	13.7	21.4
bus2x2 (panel = 4312)					
1	17.6	0.3	9.0	9.3	36.4
2	9.5	0.2	4.5	8.9	22.5
4	5.5	0.1	1.9	8.2	15.8
8	3.4	0.05	1.0	7.8	12.4
16	2.5	0.03	0.5	7.5	10.7
via (panel = 6120)					
1	28.3	0.5	10.9	12.5	52.3
2	16.7	0.3	5.5	12.0	34.5
4	10.6	0.1	2.7	11.0	24.5
8	7.6	0.07	1.4	10.1	19.0
16	6.1	0.04	0.7	9.5	16.5

6.3. Memory Scalability

We now examine the memory scalability of the parallel program. Table 2 presents memory allocation (MB) on any one processor of the IBM SP2 for three benchmarks with different degree of geometry complexities and discretization. The data is collected after solving for the full capacitance matrix , under the cyclic mapping scheme. The block mapping leads to approximately the same results. As our parallel program allocates no extra storage (and performs no extra computation) when using only one processor, the memory allocation for one processor in our parallel program is the same as the original serial code.

The *direct* column gives the amount of memory allocation which is related to the direct evaluation. It also includes the storage for the preconditioning pass. The *multipole* column shows the allocation corresponding to the evaluation of multipole expansions. Similarly, the allocation related to local expansion evaluation is presented in *local*. There are several sources for the miscellaneous memory allocation given in column *misc*. Among them, the storage required in the initialization, i.e., the input of the problem discretization and the construction of the cube hierarchy, accounts for about 50% of the allocation. The storage requirement of buffers for message passing is also counted here, which accounts for about 10% of the allocation. Other sources

include the storages in various initializations, bookkeeping and accountings in FASTCAP.

It can be seen that the parallel program has very good memory scalability in the critical portion of the computation, i.e. the multipole accelerated matrix construction and matrix-vector product (as shown in the *direct, multipole and local* columns). For the sequential FASTCAP, the initialization accounts only about 0.1% to 1% of the total CPU time for a variety of benchmarks, in parallel computing, however, it has become the bottleneck to achieve good memory scalability.

7. Conclusions

We presented a parallel formulation for the well known multipole accelerated 3-D capacitance extraction program FASTCAP. Experimental results were reported on an IBM SP2. Typical results included a speedup of about 9 on 16 processors. The approach presented here involved minimum changes to the serial code and we have reported very good efficiencies compared to other relevant researches. To the best of our knowledge, the algorithms presented in the paper is among the first parallel formulations for the fast multipole accelerated FASTCAP, with new preconditioning and adaptive features (note that the FASTCAP uses a different adaptive scheme in contrast to that in [13], and a different preconditioning method in contrast to [17]).

Acknowledgments The authors would like to thank Prof. J. White and his group at MIT for providing the FASTCAP program, the associated examples and documents, on which these experiments were based.

References

[1] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Physics.*, 73 (1987), pp.325-348.

[2] A. W. Appel, "An efficient program for many-body simulations," *SIAM J.Sci. Statist. Computing*, 6 (1985), pp.85-103.

[3] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, *Nature*, 324 (1986), pp.446-449.

[4] K. Nabors and J. White, "FASTCAP: A multipole-accelerated 3-D capacitance extraction program," *IEEE Trans. Computer Aided Design*, pp. 1447-1459, Nov. 1991.

[5] K. Nabors, S. Kim, and J. White, "Multipole-accelerated capacitance extraction algorithms for 3-D structures with multiple dielectrics," *IEEE Trans. Circuits and Systems*, vol.39, no.11, pp. 946-954, 1992.

[6] 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. Comput.*, no.15, pp.713-735, 1994.

[7] W. Shi, J. Liu, N. Kakani, and T. Yu, "A fast hierarchical algorithm for 3-D capacitance extraction," in *Proc. ACM/IEEE Design Automation Conference*, pp.212-217, June 1998.

[8] J. Carrier, L. Greengard, and V. Rokhlin, "A fast adaptive multipole algorithm for particle simulation," *SIAM J. Sci. Stat. Comput.*, vol.9, no.4, pp. 669-686, 1988.

[9] Z. Wang, Y. Yuan, and Q. Wu, "A parallel multipole accelerated 3-D capacitance simulator based on an improved model," *IEEE Trans. Computer Aided Design*, vol.15, no.12, pp. 1441-1450, Dec. 1996.

[10] N. R. Aluru, V. B. Nadkarni, and J. White, "A parallel precorrected FFT based capacitance extraction program for signal integrity analysis," *Proc. of ACM/IEEE Design Automation Conference*, June 1996.

[11] Y. Yuan and P. Banerjee, "A parallel 3-D capacitance extraction program," *Proc. of International Conference on High Performance Computing*, Calcutta, India, Dec. 1999.

[12] L. Greengard and W. D. Gropp, "A parallel version of the fast multipole method," in *Parallel Processing for Scientific Computing*, pp.213-222, 1987.

[13] J. F. Leathrum and J. A. Board, "Mapping the adaptive fast multipole algorithm into MIMD systems," in P.Mehrotra and J.Saltz, editors, *Unstructured Scientific Computation on Scalable Multiprocessors*, MIT Press, Cambridge, MA, 1992.

[14] F. Zhao and S. L. Johnsson, "The parallel multipole method on the connection machine," *SIAM J. Sci. Stat. Comput.*, 12 (1991), pp.1420-1437.

[15] J. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy, "Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-Hut, fast multipole, and radiosity," *Journal of Parallel and Distributed Computing*, 27 (1995), pp.118-141.

[16] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree N-body algorithm," in *Proc. Supercomputing*, 1993, pp.12-21.

[17] A. Grama, V. Kumar, and A. Sameh, "Parallel hierarchical solvers and preconditioners for boundary element method," *SIAM Journal of Scientific and Statistical Computing*, vol.20, no.1, pp.337-358, 1998.

[18] A. E. Ruehli and P. A. Brennan, "Efficient capacitance calculations for three-dimensional multiconductor systems," *IEEE Trans. Microwave Theory and Tech.*, vol.21, pp.76-82, 1973.

[19] S. Rao, T. Sarkar, and R. Harrington, "The electrostatic field of conducting bodies in multiple dielectric media," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-32, no. 11, pp. 1441–1448, November 1984.

[20] Y. Saad and M. H. Shultz, "GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems," *SIAM Journal of Scientific and Statistical Computing*, 7:856-869, July 1986.

[21] K. Nabors, S. Kim, J. White, and S. Senturia, "FASTCAP user's guide," Dept. of EECS, MIT, Cambridge, MA 02139, USA, Sept. 1992.