A Fast Hierarchical Algorithm for Three-Dimensional Capacitance Extraction

Weiping Shi, *Member, IEEE,* Jianguo Liu, Naveen Kakani, and Tiejun Yu, *Member, IEEE*

***Abstract*—The authors present a new algorithm for computing the capacitance of three-dimensional electrical conductors of complex structures. The new algorithm is significantly faster and uses much less memory than previous best algorithms and is kernel independent.**

The new algorithm is based on a hierarchical algorithm for the n-body problem and is an acceleration of the boundary element method (BEM) for solving the integral equation associated with the capacitance extraction problem. The algorithm first adaptively subdivides the conductor surfaces into panels according to an estimation of the potential coefficients and a user-supplied error bound. The algorithm stores the potential coefficient matrix in a hierarchical data structure of size $O(n)$, although the matrix is size n^2 if expanded explicitly, where n is the number of panels. The hierarchical data structure allows the multiplication of the coefficient matrix with any vector in $O(n)$ time. Finally, a generalized minimal residual algorithm is used to solve m linear systems each of size $n \times n$ in $O(mn)$ time, where m is the number of conductors.

The new algorithm is implemented and the performance is compared with previous best algorithms for the $k \times k$ bus example. The new algorithm is 60 times faster than FastCap and uses 1/80 of the memory used by FastCap. The results computed by the new algorithm are within 2.5% from that computed by FastCap. The new algorithm is 5 to 150 times faster than the commercial software QuickCap with the same accuracy.

***Index Terms*—Boundary element method, capacitance, parasitic extraction.**

I. Introduction

In THIS paper, we study the capacitance extraction problem of three-dimensional (3-D) electrical conductors of complex structures. Fast and accurate capacitance extraction is important in the design and verification of deep submicron very large scale integration (VLSI) circuits and packaging [9], [10], [13]–[15], [17] and MEMS [19]. Although there are a number of algorithms and commercial software for capacitance extraction, they are far from adequate. The 2-D/2.5-D algorithms are based on pattern matching. These algorithms are fast and capable of full chip extraction, but are inaccurate. The 3-D algorithms are based on integral equations or differential equations. These algorithms are accurate, some are capable of critical net and clock tree extraction, but are very slow. In addition, all 2-D/2.5-D algorithms use 3-D algorithms to build the extraction pattern library. A pattern library can easily contain tens of thousands of rules and take many hours of CPU time to generate. Therefore, a fast and accurate 3-D algorithm will have a great impact on the speed and accuracy of commercial extraction tools.

Manuscript received April 23, 2001. This paper was recommended by Associate Editor M. Sarrafzadeh.

W. Shi is with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: wshi@ee.tamu.edu).

J. Liu is with the Department of Mathematics, University of North Texas, Denton, TX 76203 USA (e-mail: jgliu@unt.edu).

N. Kakani is with the Department of Computer Science, University of North Texas, Denton, Texas 76203 USA.

T. Yu is with the Department of Mathematics, University of North Carolina, Charlotte, NC 28223 USA.

Publisher Item Identifier S 0278-0070(02)01782-7.

A. Integral Equation Approach

The capacitance of an m-conductor geometry can be summarized by an $m \times m$ capacitance matrix $\mathbf{C}$. The diagonal entries C_{ii} of $\mathbf{C}$ are positive, representing the self-capacitance of conductor i. The nondiagonal entries C_{ij} are negative, representing the coupling capacitance between conductors i and j. To determine the jth row of the capacitance matrix, we compute the surface charges on each conductor produced by raising conductor j to unit potential while grounding the rest of the conductors. Then C_{ij} is numerically equal to the charge on conductor i. This procedure is repeated m times to compute all rows of $\mathbf{C}$.

Each of the m potential problems can be solved using an equivalent free-space formulation where the conductor-dielectric interfaces are replaced by a charge layer of density σ. Assuming a homogeneous dielectric, the charge layer in the free-space problem will satisfy the integral equation

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

where $\psi(x)$ is the known conductor surface potential, da' is the incremental conductor surface area, $x, x' \in \Re^3$, $x' \in da'$, and $||x - x'||$ is the Euclidean distance between x and x'. The kernel of the integral equation is $1/||x - x'||$.

We use the Galerkin scheme to numerically solve (1) for σ. In this approach, the conductor surfaces are subdivided into n small panels and it is assumed that on each panel A_i, a charge q_i is uniformly distributed. Then for each panel A_i, an equation is written which relates the known potential on A_i, denoted by v_i, to the sum of the contribution of potential from charges on all n panels $A_1, A_2, \ldots, A_n$. The result is a dense linear system

$$\mathbf{Pq} = \mathbf{v} \tag{2}$$

where $\mathbf{q} \in \Re^n$ is the vector of panel charges, $\mathbf{v} \in \Re^n$ is the vector of known panel potentials, and $\mathbf{P} \in \Re^{n \times n}$ is the potential coefficient matrix. Each entry of $\mathbf{P}$ is defined as

$$P_{ij} = \frac{1}{\text{area}(A_i)} \times \int_{x_i \in A_i} \frac{1}{\text{area}(A_j)} \int_{x_j \in A_j} \frac{1}{4\pi\epsilon_0 ||x_i - x_j||} da_j da_i \tag{3}$$

for panels A_i and A_j. Matrix $\mathbf{P}$ is known to be positive, symmetric, and positive definite. Once the linear system (2) is solved, the capacitances are obtained by summing the panel charges.

To solve (2), direct methods based on triangularization of matrix $\mathbf{P}$, such as Gaussian elimination and Cholesky factorization, require $O(n^3)$ operations. Iterative algorithms normally require $O(n^2)$ operations per iteration. These approaches are inefficient if the number of panels is more than several thousands. In fact, any algorithm that uses the explicit representation of matrix $\mathbf{P}$ must use at least $\Omega(n^2)$ time and memory since the matrix is size n^2.

Several algorithms have been proposed to solve (2) in subquadratic time. The capacitance extraction algorithm FastCap of Nabors and White [13] uses $O(n)$ time. It is based on the multipole algorithm for the n-body problem by Greengard and Rohklin [7]. In this paper we propose a different $O(n)$ time algorithm, which is based on a different hierarchical algorithm for the n-body problem by Appel [1]. Efficient algorithms that are not based on the n-body problem include the precorrected fast Fourier transform (FFT) algorithm of Phillips and White [17], the singular value decomposition (SVD) algorithm of Kapur and Long [9], [10], etc. The running times of the precorrected FFT algorithm and SVD algorithm are both $O(n \log n)$. Le Coz and Iverson [11] proposed a Monte Carlo algorithm and successfully turned it into the popular commercial software QuickCap.

B. Kernel Independence

In deep submicron technology, multilayer dielectric with different permittivity is common. When this happens, the kernel of integral (1) is no longer $1/\|x-x'\|$. Instead, it is a complicated multilayer Green's function, which is often approximated by a series [12].

A capacitance extraction algorithm is kernel dependent if the algorithm is written for a specific kernel. FastCap is kernel dependent, since all multipole expansion theorems are derived specifically for the $1/\|x-x'\|$ kernel [7]. To use FastCap in multilayer dielectric, it is necessary to introduce additional panels on the dielectric–dielectric interface and additional variables and equations, so that within each layer, the kernel is still $1/\|x-x'\|$[14]. As a result, the running time is drastically increased.

On the other hand, a capacitance extraction algorithm is kernel independent if the algorithm works for any kernel. Therefore, when applied to a multilayer dielectric, kernel-independent algorithms are much more efficient than kernel-dependent algorithms. Our algorithm is kernel independent because it treats Green's function as a black box; our algorithm provides the coordinates of two points in 3-D space and the black box returns the value of the Green's function. The precorrected FFT algorithm [17], SVD algorithm [9], and QuickCap [11] are also kernel independent.

C. The n-Body Problem

The n-body problem is to compute the gravitational force between n particles in 3-D space, where each particle exerts a force on all the other particles, implying $\binom{n}{2}$ pairwise interactions [1], [7]. The capacitance extraction problem shares many similarities with the n-body problem; both the gravitational field and the electro-magnetic field decrease as the distance increases and the principle of superposition applies to both problems. The principle of superposition states that the potential due to a cluster of particles is the sum of the potential due to each individual particle.

There are two types of algorithms for the n-body problem: the hierarchical algorithm of Appel [1] and the multipole algorithm of Greenberg and Rohklin [6]. When Appel first published his paper, he thought the time complexity was $O(n \log n)$. Later, a careful analysis shows the time complexity is $O(n)$ [4]. At about the same time, Greengard and Rohklin [6] proposed the multipole algorithm and proved its time complexity is $O(n)$. FastCap is based on the multipole algorithm, while our algorithm is based on Appel's algorithm [1] and a radiosity algorithm [8]. Hanrahan *et al.* [8] used the ideas in Appel's algorithm to compute the reflection of light among a set of objects in computer graphics. Their experience shows Appel's algorithm is not only fast, but also very easy to implement.

The hierarchical algorithm uses the following key ideas to speedup the computation: 1) For practical considerations, the forces acting on a particle need only be calculated to within the given precision. 2) The force due to a cluster of particles at some distance can be approximated with a single term.

There are several differences between the capacitance extraction problem and the n-body problem, which prevent us from blindly adopting the n-body solution. One major difference is that in the n-body problem the objects are particles, while in the capacitance extraction problem, the objects are continuous conductor surfaces. Therefore, the hierarchical data structures are formed differently. The n-body algorithm begins with n particles and clusters them into larger and larger groups. Our algorithm begins with a set of panels and subdivides them into smaller and smaller panels. Another difference is that the self-capacitance is very important in our problem while there is no such concept in the n-body problem.

D. Our Contribution

In Section II, we show how to build a hierarchical data structure for the potential coefficient matrix by adaptively subdividing the conductor surfaces into panels according to the estimation of the coefficient and a user-supplied error bound. This is different from FastCap where the conductor surfaces are divided in a preprocessing stage according to the geometry of each conductor individually. Our new algorithm subdivides the surfaces into panels of variable sizes and it is guaranteed that all coefficients are calculated to the same precision. More importantly, the coefficient matrix $\mathbf{P}$ of size $n \times n$ is stored as $O(n)$ block entries in the hierarchical data structure, where n is the number of panels.

In Section III, we show how to multiply the potential coefficient matrix $\mathbf{P}$ with any vector in $O(n)$ time, using the hierarchical data structure. The hierarchical data structure is of size $O(n)$, or more precisely $O(n+m^2)$, where m is the number of conductors and is much less than n. Then we use the generalized minimum residual (GMRES) method to solve the linear systems.

In Section IV, we present the experimental results that show the new algorithm is significantly faster than FastCap and uses significantly less memory. We also compared our algorithm with QuickCap and other algorithms. Our algorithm is much faster as well.

Section V is the conclusion. We also discuss how to apply our algorithm to a multilayer dielectric using multilayer Green's functions. Since our algorithm is kernel independent, there is no need to introduce dielectric–dielectric interface conditions and additional variables as required by kernel-dependent algorithms such as FastCap [14].

II. Potential Matrix Approximation

This section describes the recursive refinement procedure that subdivides a large panel into a hierarchy of small panels and builds a hierarchical representation of the potential coefficient matrix. We first describe the procedure, then estimate the number of interactions that need to be considered and finally analyze the error in the resulting potential matrix.

```
Refine(Panel Ai, Panel Aj)
{
    Pij = PotentialEstimate(Ai, Aj);
    Ri = longest side of Ai;
    Rj = longest side of Aj;
    if ((Pij*Ri < Peps) && (Pij*Rj < Peps))
        RecordInteraction(Ai, Aj);
    else if (Ri > Rj)) {
        Subdivide(Ai);
        Refine(Ai.left, Aj);
        Refine(Ai.right, Aj);
    } else {
        Subdivide(Aj);
        Refine(Ai, Aj.left);
        Refine(Ai, Aj.right);
    }
}
```

Procedure `PotentialEstimate` returns an estimate of the potential coefficient for two panels defined in (3). We use the closed form expressions derived by Wilton *et al.* [20] and numerical integration to compute the potential factor. If the estimated coefficient is less than the user provided error bound $\texttt{Peps}(P_\epsilon)$, then the panels are allowed to interact at this level. The recursion is terminated and the interaction is recorded between the two panels by procedure `RecordInteraction`. However, if the estimate is greater than P_ϵ, then the estimate may not be accurate. In this case, the panel with the larger area, say `Ai`, is to be subdivided into `Ai.left` and `Ai.right`. The procedure `Refine` is called recursively. Procedure `Subdivide` subdivides a panel into two small panels. The subdivision hierarchy is stored in a binary tree where each node has two pointers, `left` and `right`, pointing to the two small panels.

Since a panel may be refined against many other panels, the actual subdivision of a panel may have occurred previously. When this happens, `Subdivide` uses the same subdivision. By sharing some subdivisions, we can reduce the amount of memory usage significantly. For example, assume when `Ai` is refined against `Aj`, `Ai` is subdivided into `Ai.left`, `Ai.right` and `Ai.left` is further subdivided. Later, assume `Ai` is also refined against `Ak`. This time, `Ai` is subdivided into `Ai.left`, but `Ai.right` is further subdivided. Therefore, the two refinements share some subdivisions, but as a whole they are different. In general, since the locations and sizes of the panels are different, the refinements are different.

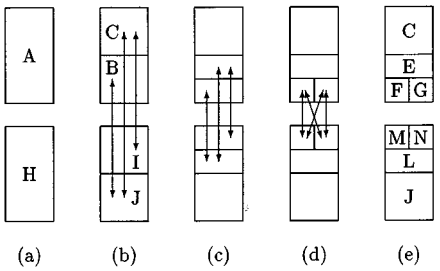

Fig. 1. Partition conductor surfaces into panels.

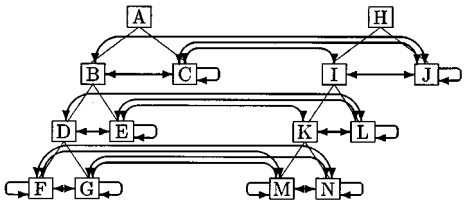

Fig. 2. Hierarchical data structure and potential coefficients.

Fig. 1 shows the refine process of two conductor surfaces. We start with two conductor surfaces A and H in (a). Assume the estimated coefficient between A and H is greater than the user provided error bound P_ϵ, so we subdivide A into $B \cup C$ and then subdivide H into $I \cup J$ in (b). Now assume the estimates between BJ, CI, and CJ are less than P_ϵ, but estimate BI is greater than P_ϵ. Then we record interactions BJ, CI, and CJ at this level, while we further subdivide panels B and I. The final panels are shown in (e). We compute the self-potential coefficient P_{ii} at this time.

Fig. 2 shows the hierarchical data structure produced by `Refine` and associated potential coefficients produced by `RecordInteraction`. The panels are stored as nodes in the tree and the coefficients are stored as links between the nodes. The value of each coefficient is stored as a floating-point number associated with the link. Each tree represents one conductor surface, each nonleaf node represents one panel further subdivided, and each leaf node represents one panel not further subdivided. The union of all the leaf nodes completely covers the surfaces of the conductors. Each horizontal link represents one pair of potential coefficients defined in (3). Each self-link represents one self-potential coefficient.

Fig. 3 shows the block matrix represented by the links of Fig. 2. Each block entry represents one interaction between panels. Note that there are total eight panels for the two conductors, so an explicit representation of the coefficient matrix would require 64 entries. However, the block matrix has only 40 entries. Furthermore, if we use uniform grid discretization, then there would be a total of 16 panels and 256 entries.

Esselink proved the number of interactions in the above algorithm is $O(n)$ [4]. Callahan and Kosaraju show that under a

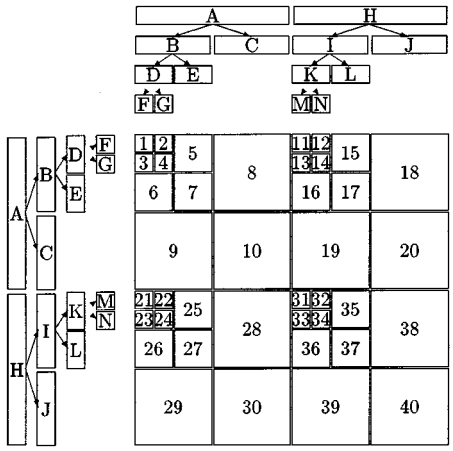

Fig. 3. Potential coefficient matrix with block entries.

very general condition, algorithms similar to ours contain $O(n)$ interactions [2]. We now give an intuitive explanation of why the block matrix contains $O(n)$ block entries. For simplicity, consider n panels of about the same size on the surface and a binary tree constructed above the panels by merging adjacent panels recursively. Then, panels on the same level of the tree are the same size. The error criterion in `Refine` says that two panels can interact directly only if `Pij*R` $<$ `Peps` where `R` is the length of the longest side of the two panels. Since P_{ij} is asymptotically $1/r$ where r is the distance between the two panels, the two panels can interact only if $R/r < P_\epsilon$, or $r > R/P_\epsilon$. For any fixed P_ϵ, this criterion requires that two panels at the same level in the tree can interact only if there are k other panels between them on that level, for some fixed constant k. If panels A_i and A_j are too far, the ancestor of A_i would have interacted with the ancestor of A_j. Therefore, A_i and A_j cannot interact either. This argument applies to all levels of the tree. Therefore, each node in the tree interacts with a constant number of other nodes, regardless of their level in the tree. So, the total number of block entries is $O(n)$. Large panels that are far apart interact directly, in the same way that small panels near each other interact directly. Notice that the size of the block in the potential factor matrix depends on the level in the tree the panel interacts with. The higher the level, the bigger the block.

For m conductors, results of Esselink [4] and Callahan and Kosaraju [2] still hold, except that there will be an additive overhead of $O(m^2)$. For the application of building the extraction library, m is typically less than 30, while n can easily reach tens of thousands. Therefore, $O(n + m^2) = O(n)$.

Our algorithm is kernel independent because for different kernels all we need to modify is the subroutine `PotentialEstimate`. The rest of the algorithm is the same. To write `PotentialEstimate` for multilayer Green's function, see [12] and [16].

Finally, we analyze the relationship between the termination criteria and the accuracy of the computed coefficients. We will show that the termination criterion places an upper bound on the error associated with the potential coefficient integral between any two interacting panels. Furthermore, all coefficients are approximated to the same precision controlled by P_ϵ.

Consider a panel A subdivided into two small panels A_1 and A_2 of similar shape and size. Let the radius of the smallest sphere that contains A be R. Consider a point x of distance r from the center of the sphere, for some $r > R$. The potential at x due to the charge on panels A_1 and A_2, with uniform charge densities σ_1 and σ_2, respectively, is

$$\int_{x' \in A_1} \frac{\sigma_1}{4\pi\epsilon_0 ||x' - x||} da' + \int_{x' \in A_2} \frac{\sigma_2}{4\pi\epsilon_0 ||x' - x||} da'. \quad (4)$$

If we treat A_1 and A_2 as a single panel A with uniform charge density $(\sigma_1 + \sigma_2)/2$, then the potential at x will be

$$\int_{x' \in A} \frac{\sigma_1 + \sigma_2}{2} \frac{1}{4\pi\epsilon_0 ||x' - x||} da'. \quad (5)$$

Assume without loss of generality $\sigma_2 \geq \sigma_1$, then the difference between (4) and (5) is

$$\begin{aligned} &\frac{\sigma_2 - \sigma_1}{2} \frac{1}{4\pi\epsilon_0} \left(\int_{A_1} \frac{1}{||x' - x||} da' - \int_{A_2} \frac{1}{||x' - x||} da' \right) \\ \leq &\frac{\sigma_2 - \sigma_1}{2} \frac{1}{4\pi\epsilon_0} \int_{A_1} \left(\frac{1}{||x' - x||} - \frac{1}{||x' - x|| + R} \right) da' \\ \leq &\frac{\sigma_2 - \sigma_1}{2} \frac{R}{r} \int_{A_1} \frac{1}{4\pi\epsilon_0 ||x' - x||} da'. \end{aligned}$$

Therefore, the relative error is at most a constant times R/r. Since P_{ij} is in proportion with $1/r$, the condition `Pij` $*$ `R` $<$ `Peps` in `Refine` implies that the error of the approximation for every entry is bounded by a constant times P_ϵ. Therefore, as P_ϵ goes to zero, the error goes to zero. Also, all entries in the coefficient matrix are approximated to the same precision, which is a constant times P_ϵ.

III. Solving Linear Systems

A. Fast Matrix-Vector Multiplication

To solve the linear system $\mathbf{Pq} = \mathbf{v}$, an iterative algorithm requires the multiplication of the coefficient matrix $\mathbf{P}$ with a vector, which normally takes $O(n^2)$ time. However, because our $\mathbf{P}$ is represented by $O(n)$ blocks, each matrix-vector multiplication can be done in $O(n)$ time. The multiplication proceeds in three steps. To help understand the algorithm, the reader can imagine $\mathbf{P}$ as the coefficient matrix, $\mathbf{q}$ as the given charge vector, and the product $\mathbf{Pq}$ as the potential due to charge $\mathbf{q}$, though the algorithm works for any matrix and vector.

In the following pseudocodes, each panel `A` has two pointers `A.left` and `A.right`, pointing to the two children panels that are the subdivision of `A`. Panel `A` also has two fields `A.charge` and `A.potential` which we will explain later. For readers not familiar with recursive tree traversal, please see [3].

The first step computes the charge of all panels in the tree. The charge of a leaf panel `Ai` is given by `Qi` from $\mathbf{q}$. The charge of an nonleaf panel is the sum of the charge of its children panels. This calculation can be done in a single depth-first traversal of the tree, propagating the charge upward. In other words, to compute the charge for each panel, the charges of its children panels are computed first and then the charge of the panel equals the sum of the charge of its children panels. The time for computing the charge for all panels is linear in terms of the number of panels in the tree.

```
    AddCharge(Panel Ai)
    {
    if (Ai is leaf)
        Ai.charge = Qi;
     else {
        Add Charge(Ai.left);
        Add Charge(Ai.right);
        Ai.charge =
        Ai.left → charge + Ai.right → charge;
    }
}
```

The second step computes for each panel A_i the potential due to its interacting panels. This can be computed by summing up the product of potential coefficient P_{ij} with charge at A_j, for all A_j that has interaction with A_i. The time for computing the charge for all nodes is linear in terms of the number of links in the tree.

```
    Collect Potential(Panel Ai)
    {
    for all Aj such that AiAj has interaction {
            Ai.potential =
                Ai.potential + Aj.charge * Pij;
            if (Ai is not leaf) {
                CollectPotential(Ai.left);
                CollectPotential(Ai.right);
            }
        }
    }
```

The third step distributes the potential from the nonleaf nodes to the leaf nodes. This is done by another depth-first traversal of the tree that propagates potential down to the leaf nodes. Each nonleaf node adds its accumulated potential to its children's potential, recursively. The time of this step is linear in terms of the number of nodes in the tree.

```
DistributePotential(Panel Ai)
{
    if (Ai is not leaf) {
        Ai.left - > potential =
            Ai.left - > potential + Ai.potential;
        Ai.right - > potential =
            Ai.right - > potential + Ai.potential;
        Distribute Potential(Ai.left);
        Distribute Potential(Ai.right);
    }
}
```

The total time for the matrix-vector product is linear in terms of the number of nodes and links. It is well known that for any binary tree with n leaves, there are exactly $n-1$ nonleaf nodes. Therefore, the time is $O(n)$, where n is the number of leaf panels.

B. Generalized Minimum Residual Method

We use the GMRES method with restart [18] to solve the $n \times n$ system of equations. The basic idea behind the GMRES method is to project the problem onto a Krylov subspace $\mathcal{K}_k$ of dimension $k < n$ using the orthonormal basis constructed by a scheme due to Arnoldi [5], solve the k-dimensional subproblem using a standard approach, and then recover the solution of the original problem from the solution of the projected problem. The Arnoldi scheme involves the coefficient matrix only multiplicatively. The dimension of the Krylov subspace is usually small.

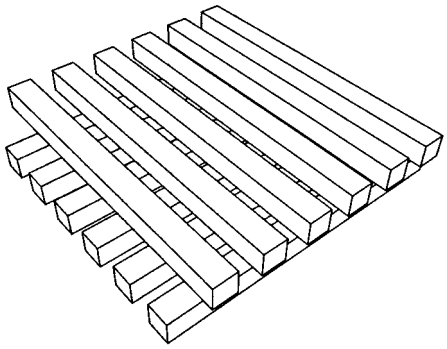

Fig. 4. Test problem 6×6 bus.

TABLE I
COMPARISON FOR THE BUS PROBLEM. TIME IS CPU SECONDS, MEMORY IS MB, AND ERROR IS WITH RESPECT TO FASTCAP(2)

	Test Problems				
	2x2	3x3	4x4	5x5	6x6
	FastCap (0)				
Time	2.1	6.2	22.3	22.3	50.7
Memory	12	16	86	24	97
Panel	792	1620	2736	4140	5832
Error	8.6%	5.0%	5.2%	8.5%	10.0%
	FastCap (2)				
Time	3.2	13.4	24.8	54.4	140.0
Memory	5.6	16	26	46	62
Panel	792	1620	2736	4140	5832
Error	—	—	—	—	—
	QuickCap				
Time	9	12	13	12	12
Error	2.0%	1.5%	1.8%	1.9%	2.3%
	New Algorithm ($P_\epsilon = 0.01$)				
Time	0.06	0.22	0.47	0.95	2.3
Memory	0.06	0.2	0.3	0.5	0.8
Panel	160	408	576	720	1584
Error	2.1%	1.7%	1.8%	1.7%	2.5%
	New Algorithm ($P_\epsilon = 0.003$)				
Time	0.99	3.9	10.6	25.1	47.4
Memory	0.9	1.9	3.8	6.3	9.1
Panel	1888	3504	6720	10960	13152
Error	0.4%	0.2%	0.8%	0.6%	0.4%

The GMRES method with restart [18] can be briefly described as follows:

ALGORITHM: GMRES(k).

1. *Start*: Let e_1 denote the first column of the $n \times n$ identity matrix. Choose x_0 and $r_0 = \mathbf{v} - \mathbf{P}x_0$, $\beta = \|r_0\|$ and $v_1 = r_0/\beta$.
2. *Iterate*: For $j = 1, 2, \cdots, k$ do: $h_{i,j} = (\mathbf{P}v_j,\ v_i)$, $i = 1, 2, \cdots, j$, $\hat{v}_{j+1} = \mathbf{P}v_j - \Sigma_{i=1}^{j} h_{i,j} v_i$, $h_{j+1,j} = \|\hat{v}_{j+1}\|$ and $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.

TABLE II
COMPARISON FOR 4 × 4 BUS PROBLEM. ERROR IS WITH RESPECT TO FASTCAP(2)

	First Row of Capacitance Matrix (pF)								Error	Time	Mem
	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}		(sec)	(MB)
FastCap(0)	394.5	-124.0	-0.175	-2.471	-52.15	-43.39	-43.08	-52.92	5.2%	22.3	86
FastCap(2)	405.2	-137.8	-11.91	-8.079	-48.36	-40.09	-40.01	-48.45	—	24.8	26
QuickCap	413	-142	-14.1	-6.6	-48.1	-39.8	-41.7	-49.5	1.8%	13	NA
New Algo $P_\epsilon = 0.01$	401.6	-139.6	-9.180	-6.480	-48.46	-37.80	-37.53	-48.46	1.8%	0.47	0.3

TABLE III
COMPARISON FOR 5 × 5 BUS PROBLEM. ERROR IS WITH RESPECT TO FASTCAP(2)

	First Row of Capacitance Matrix (pF)										Error	Time	Mem
	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	$C_{1,10}$		(sec)	(MB)
FastCap(0)	505.5	-142.8	-10.48	-20.31	2.21	-54.51	-50.13	-46.04	-50.32	-55.01	8.5%	22.3	24
FastCap(2)	484.5	-166.1	-13.62	-6.17	-6.54	-48.84	-40.12	-40.12	-40.21	-48.90	—	54.4	46
QuickCap	486	-162	-14	-8.34	-5.93	-51.8	-40.8	-42	-38.9	-48.7	1.9%	12	NA
New Algo $P_\epsilon = 0.01$	476.5	-168.2	-12.97	-1.91	-4.48	-47.37	-38.52	-37.36	-38.41	-47.69	1.7%	0.95	0.5

3. *Form the approximate solution*:
   $\bar{H}_k = [h_1 h_2 \cdots h_k]$,
   where $h_j = [h_{1,j} h_{2,j} \cdots h_{j+1,j}]^T$.
   $V_k = [v_1 v_2 \cdots v_k]$ and $x_k = x_0 + V_k y_k$,
   where y_k minimizes $\|\beta e_1 - \bar{H}_k y\|$, $y \in \Re^k$.
4. *Restart*:
   Compute $r_k = b - \mathbf{P} x_k$.
   If satisfied then stop,
   else compute $x_0 = x_k$, $\beta = \|r_k\|$, $v_1 = r_k/\beta$ and goto 2.

Since we can multiply $\mathbf{P}$ with any vector in $O(n)$ time, each iteration can be computed in $O(n)$ time.

IV. EXPERIMENTAL RESULTS

The new algorithm is implemented and compared with FastCap 2.0 and QuickCap 3.1. All computation are done on the same computer, a SUN Ultra Enterprise 2 at Texas A&M University. It has two UltraSparc II processors at 296 MHz and 1GB memory. FastCap(0) is FastCap with expansion order 0. It is the fastest version in the FastCap package. FastCap (2) is FastCap with expansion order 2. It is the most accurate version in the FastCap package, within a reasonable amount of time. FastCap (0) is about twice as fast as FastCap (2). However FastCap(0) has 5% to 10% relative error with respect to FastCap (2). QuickCap is a widely used commercial software by Random Logic Corporation.

The test examples are $k \times k$ bus crossing conductors for $k = 2$ to 6, taken from the FastCap paper [13]. The 6×6 bus example is shown in Fig. 4. Each bus in the $k \times k$ example is scaled to $1m \times 1m \times (2k+1)m$. The distance between buses on the same layer is $1m$, and the distance between layers is $1m$. The constant $4\pi\epsilon_0$ is 111.27 pF/m, according to [13].

The error of capacitance matrices is defined as follows. Let the capacitance matrix computed by FastCap (2) be $\mathbf{C}$ and the capacitance matrix computed by another program be $\mathbf{C}'$. Then the error is estimated in the Frobenius norm [5]: $\|\mathbf{C} - \mathbf{C}'\|/\|\mathbf{C}\|$. This is the standard way to measure the difference between two matrices.

Our GMRES solver reduces the two-norm of the residual to 1% of the initial residual, the same condition used by the conjugate residual algorithm in FastCap.

Table I compares the new algorithm, FastCap and QuickCap. Tables II and III show the first row of the capacitance matrix computed by the three programs. The conductors are numbered from one side to the other side 1, 2, …, k (top layer) and then $k+1, \ldots, 2k$ (bottom layer). The following is a summary of the comparison.

1) Set $P_\epsilon = 0.01$ and compare with FastCap. Compared with FastCap(2), our algorithm is 52 to 60 times faster and uses 1/93 to 1/68 of the memory. The error is less than 2.5% with respect to FastCap(2). Compared with FastCap(0), our algorithm is 22 to 47 times faster and is three times more accurate. (Some memory usage numbers reported by FastCap(0) appear to be incorrect.)
2) Set $P_\epsilon = 0.01$ and compare with QuickCap. The running time of QuickCap will increase significantly if we require high accuracy, which is typical for any Monte Carlo algorithms. Therefore, we use the results of the first run of QuickCap. Our algorithm is 5 to 150 times faster than QuickCap and the accuracy is the same. It appears the running time of QuickCap is independent of the size of input, which we do not understand since we do not have access to the source code of QuickCap.
3) $P_\epsilon = 0.003$ and compare with FastCap(2). The new algorithm is 2 to 4 times faster and uses 1/6 to 1/8 of the memory. The error is less than 0.8% with respect to FastCap(2).

It is natural to ask how to choose the value of P_ϵ. Since P_ϵ gives an asymptotic error bound, similar to the expansion order of FastCap and running time/variance of QuickCap, P_ϵ does not translate directly to the accuracy of computed capacitance matrix C. The relationship between P_ϵ and error of $\mathbf{C}$ can only be measured for an actual implementation using a reference. For the current implementation and with FastCap as the reference, $P_\epsilon = 0.01$ is the default value.

We do not have access to the precorrected FFT algorithm [17] and the SVD algorithm [9]. Published results show the precorrected FFT algorithm and the SVD algorithm are about twice as fast as FastCap for the $k \times k$ bus examples [9], [17]. Therefore, based on their relative performance to FastCap, our algorithm is much faster than these algorithms as well.

V. Conclusion

This paper presents a hierarchical algorithm that is significantly faster than previous best algorithms and uses much less memory. The new algorithm is kernel independent and, therefore, is even more efficient when applied to multilayer dielectrics. The new algorithm does not require preprocessing to partition the conductor surface into panels; instead, it automatically partitions the panels according to a user supplied error bound. The new algorithm provides continuous tradeoff of time with precision by changing the error bound P_ϵ.

There are a number of differences between our algorithm and FastCap: 1) We include the discretization process into the algorithm. By doing so, we not only find a good discretization, but also get the hierarchy for free, which FastCap needs to rebuild from scratch. We experimented by feeding the results of our discretization to FastCap. It was discovered that our discretization gives better accuracy than the discretization that only considers the geometry of each conductor separately. 2) We use a single term to approximate the charge and potential, which is the same as multipole expansion order zero. However, we use far more panels to compensate the low expansion order. It can be seen from Table I that it is less expensive to use more panels than to use high expansion orders. Also, to use any expansion order higher than 0 will make the algorithm kernel dependent. 3) Our hierarchical data structure is more efficient than the array used by FastCap. Experimental results show this is a major source of improvement. Also, some features, such as the adaptive evaluation defined in FastCap [15], is free under our data structure.

Acknowledgment

The authors wish to thank the following people for their help during this research: S. Tate and D. Zhou for discussions and references at an early stage of the research, J. White for suggestions on the experiments, Random Logic Corporation for providing them with QuickCap, a DAC reviewer for pointing out the kernel-independence feature of the algorithm, and two TCAD reviewers for improving the presentation.

References

[1] A. A. Appel, "An efficient program for many-body simulation," *SIAM J. Sci. Stat. Comput.*, vol. 6, no. 1, pp. 85–103, 1985.

[2] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with application to the k-nearest neighbor and n-body potential field," *J. Assn. Computing Machinery*, vol. 42, no. 1, pp. 67–90, Jan. 1995.

[3] T. H. Corman, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[4] K. Esselink, "The order of Appel's algorithm," *Inform. Processing Lett.*, vol. 41, pp. 141–147, 1992.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1989.

[6] L. Greengard and V. Rohklin, "A fast algorithm for particle simulations," *J. Comp. Phys.*, vol. 73, pp. 325–348, 1987.

[7] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*. Cambridge, MA: MIT Press, 1988.

[8] P. Hanrahan, D. Salzman, and L. Aupperle, "A rapid hierarchical radiosity algorithm," *Computer Graphics (Proc. SIGGRAPH'91)*, vol. 25, no. 4, pp. 197–206, July 1991.

[9] S. Kapur and D. E. Long, "IES^3: A fast integral equation solver for efficient 3-dimensional extraction," in *Proc. 1997 ICCAD*, Nov. 1997, pp. 448–455.

[10] S. Kapur and J. Zhao, "A fast method of moments solver for efficient parameter extraction of MCMS," in *Proc. 34th Design Automation Conf.*, June 1997, pp. 141–146.

[11] Y. L. L. Coz and R. B. Iverson, "A stochastic algorithm for high speed capacitance extraction in integrated circuits," *Solid State Electron.*, vol. 35, no. 7, pp. 1005–1012, 1992.

[12] K. A. Michalski and J. R. Mosig, "Multilayered media Green's functions in integral equation formulations," *IEEE Trans. Antennas Propagat.*, vol. 45, pp. 508–519, Mar. 1997.

[13] K. Nabors and J. White, "Fastcap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1447–1459, Nov. 1991.

[14] ——, "Multipole-accelerated 3-D capacitance extraction algorithms for structures with conformal dielectrics," in *Proc. 29th Design Automation Conf.*, June 1992, pp. 710–715.

[15] K. N. Nabors *et al.*, "Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory," *SIAM J. Sci. Comput.*, vol. 15, no. 3, pp. 713–735, May 1994.

[16] K. S. Oh, D. Kuznetsov, and J. E. Schutt-Aine, "Capacitance computation in a multilayerd dielectric medium using closed-form spatial Green's function," *IEEE Trans. Microwave Theory Techn.*, vol. 42, pp. 1443–1453, Aug. 1994.

[17] J. R. Phillips and J. White, "A precorrected FFT method for capacitance extraction of complicated 3-D structures," in *Proc. 1994 ICCAD*, pp. 268–271.

[18] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, July 1986.

[19] D. Teegarden, G. Lorenz, and R. Neul, "How to model and simulate microgyroscope systems," *IEEE Spectrum*, vol. 32, no. 7, pp. 66–75, July 1998.

[20] D. R. Wilton *et al.*, "Potential integration for uniform and linear source distributions on polygonal and polyhedral domains," *IEEE Trans. Antennas Propagat.*, vol. AP-32, pp. 276–281, Mar. 1984.

Weiping Shi (M'96) received the B.S. and M.S. degrees in computer science from Xi'an Jiaotong University, China, in 1982 and 1984, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1992.

He was with the University of North Texas and spent a one-year sabbatical with PDF Solutions before joining Texas A&M University, where he is now an Associate Professor in the Department of Electrical Engineering. His research interests include computer-aided design of VLSI circuits and design and analysis of algorithms.

Jianguo Liu received the B.S. and M.S. degrees in computational mathematics from Xiangtan University, China, in 1982 and 1984, respectively, and the Ph.D. degree in applied mathematics from Cornell University, Ithaca, NY, in 1994.

He is now an Associate Professor in the Department of Mathematics, University of North Texas. His research interests include numerical optimization and scientific computation.

Naveen Kakani, photograph and biography not available at the time of publication.

Tiejun Yu (M'99) received the M.S. and Ph.D. degrees in electrical engineering from Tsinghua University, China, in 1991 and 1996, respectively. From 1997 to 1999, he was a postdoc with the Department of Mathematics, University of North Carolina at Charlotte.

From 1999 to 2000, he was a Research Associate with Duke University. He is now a Senior R&D Engineer with Cadence Design Systems, San Jose, CA. His current research intersets include EM scattering, parameters extraction, and simulation for RF ICs and packages.