A Fast Hierarchical Algorithm for 3-D Capacitance Extraction

Weiping Shi, Jianguo Liu†, Naveen Kakani and Tiejun Yu‡
Dept. of Computer Science, Univ. of North Texas, Denton, TX 76203
† Dept. of Mathematics, Univ. of North Texas, Denton, TX 76203
‡ Dept. of Mathematics, Univ. of North Carolina, Charlotte, NC 28223

Abstract

We present a new algorithm for computing the capacitance of three-dimensional perfect 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 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 us to multiply the coefficient matrix with any vector in $O(n)$ time. Finally, we use a generalized minimal residual algorithm 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, our algorithm is 100 to 40 times faster than FastCap, and uses 1/100 to 1/60 of the memory used by FastCap. The results computed by the new algorithm are within 2.7% from that computed by FastCap.

1 Introduction

In this paper, we study the capacitance extraction problem of three-dimensional perfect electrical conductors of complex structures. Fast and accurate capacitance estimation is important in the design of high-performance integrated circuits [8, 9, 10, 11, 12, 13, 15]. Two examples of three-dimensional complex structures for which capacitance strongly affects performance are DRAM cells and chip carriers used in high density packaging [10].

1.1 Integral Equation Approach

We assume the conductors are embedded in a homogeneous dielectric medium, though our techniques can be extended to multiple dielectrics by using multi-layered Green functions. Since our algorithm is kernel independent, there is no need to introduce dielectric-dielectric interface conditions and additional variable, as required by kernel-dependent algorithms such as FastCap [11].

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, and the non-diagonal entries C_{ij} are negative, representing the coupling capacitance between conductors i and j. To determine the j-th column of the capacitance matrix, we need only solve for the surface charges on each conductor produced by raising conductor j to unit potential while grounding 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 columns 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 be the induced charge in the original problem if σ satisfies the integral equation

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

where $\psi(x)$ is the known conductor surface potential, da' is the incremental conductor surface area, $x, x' \in \Re^3$, and $\|x - x'\|$ is the Euclidean distance between x and x'. The kernel of the integral equation is $1/\|x - x'\|$. For simplicity, we will temporarily omit the scale factor $1/4\pi\epsilon_0$ and reintroduce it later in Section 4.

We use Galerkin scheme to numerically solve (1) for σ. In this approach, the conductor surfaces are divided 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 $A_1, \ldots, A_n$. The result is a dense linear system

$$\mathbf{P}\mathbf{q} = \mathbf{v}, \tag{2}$$

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 where each entry

$$P_{ij} = \frac{1}{area(A_i)} \int_{x_i \in A_i} \frac{1}{area(A_j)} \int_{x_j \in A_j} \frac{1}{\|x_i - x_j\|} \, da_j da_i, \quad (3)$$

for panels A_i and A_j. Matrix **P** is known to be positive, symmetric and positive definite.

The linear system of (2) has to be solved to compute panel charges, and the capacitances are derived by summing the panel charges. Direct methods based on triangularization of matrix **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 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 sub-quadratic time. The FastCap algorithm of Nabors and White [10, 12] uses $O(n)$ time, and is based on the $O(n)$ time multipole algorithm for the n-body problem [6]. In this paper we propose a different $O(n)$ time algorithm, which is based on a different $O(n)$ time hierarchical algorithm for the n-body problem by Appel [1]. Efficient algorithms that are not based on the n-body problem include an FFT algorithm of Phillips and White [13], and the singular value decomposition (SVD) algorithm of Kapur and Zhao [9, 8]. The FFT algorithm and the SVD algorithm are about twice as fast as FastCap for test examples [13, 8], though the running time of both algorithms are $O(n \log n)$.

1.2 The n-Body Problem

In the n-body problem, each of the n particles exerts a force on all the other $n-1$ particles, implying $\binom{n}{2}$ pairwise interactions [1, 6]. The capacitance extraction problem shares many similarities with the n-body problem. Both the gravitational force and the electro-magnetic field decreases 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] (1985) and the multipole algorithm of Greenberg and Rohklin [5] (1987). 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 $O(n)$ [3]. At about the same time, Greengard and Rohklin [5] proposed the multipole algorithm, and proved its time complexity is $O(n)$. The capacitance extraction algorithm FastCap is related to the multipole algorithm, while our algorithm is related to Appel's algorithm [1], and a radiosity algorithm [7]. Hanrahan, Salzman and Aupperle [7] 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 of Appel [1] uses the following two key ideas to compute all the forces on a particle: 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, within the given precision, with a single term. However, 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 the panels into smaller and smaller panels. (We can also start with a set of given small panels and build a hierarchical data structure using the same idea.) Another difference is that the self-capacitance is very important in our problem while there is no such concept in the n-body problem.

1.3 Our Contribution

In Section 2, we show how to build a hierarchical data structure for the potential coefficient matrix by adaptively subdividing the conductor surfaces into subpanels according to the estimation of the coefficient and a user-supplied error bound. This is different from the multipole algorithm where the conductor surfaces are divided in a pre-processing 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 contains $O(n)$ block entries, where n is the number of panels.

In Section 3, we show how to multiply the potential coefficient matrix **P** with any vector in $O(n)$ time, using the hierarchical data structure. The hierarchical data structure is of size $O(n)$, but it implicitly stores the whole $n \times n$ potential coefficient matrix. As a result, the multiplication can be done in $O(n)$ time through three traversals of the hierarchical data structure. Then we use the Generalized Minimum Residual (GMRES) method to solve the linear systems.

In Section 4, we present the experimental results that show the new algorithm is significantly faster than the multipole algorithm FastCap and uses significantly less memory, with a small difference in the result compared with FastCap.

2 Potential Matrix Approximation

This section describes the recursive refinement procedure which decomposes a large panel into a hierarchy of small panels, and builds a hierarchical representation of the potential 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) AND (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);
  }
}
```

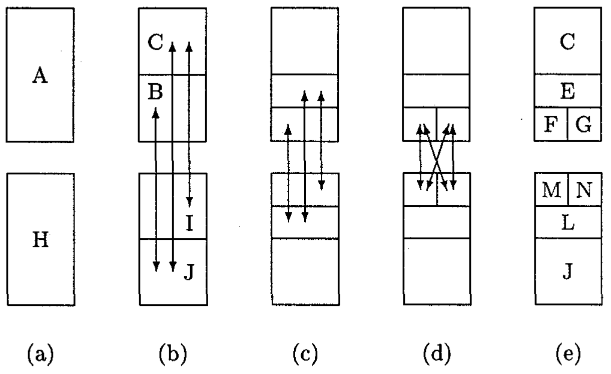

Figure 1: Partition conductor surfaces into panels.

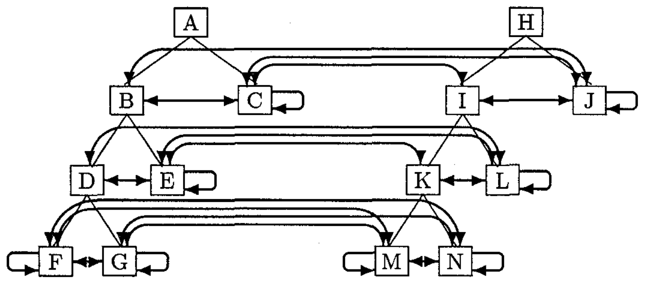

Figure 2: Potential coefficient matrix stored as links.

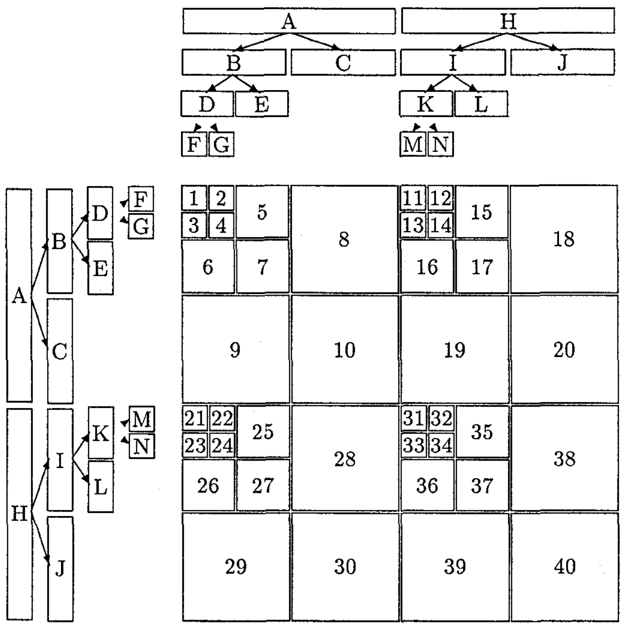

Figure 3: Potential coefficient matrix with block entries.

Procedure PotentialEstimate returns an estimate of the potential factor from panel Ai to panel Aj defined by (3). We use the closed form expressions derived by Wilton, et al [16] and numerical integration to compute the potential factor. If the estimated potential factor is less than the user provided error bound Peps (P_ϵ), then the panels are allowed to interact at this level of detail. The recursion is terminated and the interaction is recorded between the two panels. However, if the potential factor estimate is too large, 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 new panels. The subdivision hierarchy is stored in a binary tree with each node having two pointers left and right pointing to the two subpanels. Since a panel may be refined against many panels, the actual subdivision of a panel may have occurred previously. When this happens, Subdivide does not perform any task.

Figure 1 shows the refine process of two conductor surfaces. We start with two conductor surfaces A and H in (a). Assume the coefficient estimates between A and H are greater than the user provided error bound P_ϵ, we subdivide A into $B \cup C$, and then subdivide H into $I \cup J$ in (b). Now assume the estimates BJ, CI and CJ are less than P_ϵ, but estimate BI is greater than P_ϵ. Therefore, we record interactions BJ, CI and CJ at this level, while further subdivide panels B and I. The final panels are shown in (e). We also compute the self-capacitance at this time. Note that if we use uniform grid partition, then there would be a total of 16 panels. Also note that we consider mainly the interaction with other panels, instead of the geometry.

Figure 2 shows the hierarchical data structure produced by Refine, and associated potential coefficients stored as links between the nodes in the hierarchy. Each binary tree represents one conductor surface, and each leaf node represents one final panel that is not further divided in the discretization. The combination of all the leaf nodes completely cover all surfaces of the conductors. Each horizontal link represents one pair of potential coefficients defined in (3). Each self-link represents one self-potential coefficient.

Figure 3 shows the block matrix represented by the links of Figure 2. Each block entry represents one interaction between panels. Note that there are total 8 panels for the two conductors, so a traditional coefficient matrix would require 64 entries. However, the block matrix has only 40 block entries, each stored as one floating-point number.

We now show the block matrix always contains $O(n)$ block entries, while the traditional matrix that explicitly lists all pairwise interactions must contain n^2 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. Therefore, panels on the same level of the tree are of 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 is equivalent to the criterion 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. On the other hand, 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 any level of the tree. Therefore, each node in the tree interacts with a constant number of other nodes, and the total number of block entries is $O(n)$. Esselink did a detailed analysis on the number of interactions by Appel's algorithm for the general case, and compared it with the multipole algorithm [3].

In Figure 2, it shows that each panel undergoes a constant number of interactions with other panels, regardless of their level in the tree. Large panels that are far apart interact directly, in the same way that small panels near each other interact directly.

Finally, we analyze the relationship between the termination criteria and the accuracy of the computed potential factors. The termination criteria causes the potential coefficient corresponding to each interaction to have approximately the same magnitude. This is because if an estimated potential factor were larger, the panels would be subdivided. More importantly, the termination criteria also places an upper bound on the error associated with the potential factor integral between any two interacting panels.

Consider a panel A which contains two sub-panels A_1 and A_2 of similar shape and size. Let the radius of the smallest sphere that contains both A_1 and A_2 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

$$\int_{x_1 \in A_1} \frac{\sigma_1}{||x' - x_1||} da_1 + \int_{x_2 \in A_2} \frac{\sigma_2}{||x' - x_2||} da_2. \tag{4}$$

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:

$$\int_{x \in A} \frac{\sigma_1 + \sigma_2}{2} \frac{1}{||x' - x||} da. \tag{5}$$

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

$$\begin{aligned}
& \frac{\sigma_2 - \sigma_1}{2} \left(\int_{A_1} \frac{1}{||x' - x_1||} da_1 - \int_{A_2} \frac{1}{||x' - x_2||} da_2 \right) \\
\leq \; & \frac{\sigma_2 - \sigma_1}{2} \int_{A_1} \left(\frac{1}{||x' - x_1||} - \frac{1}{||x' - x_1|| + R} \right) da_1 \\
\leq \; & \frac{\sigma_2 - \sigma_1}{2} \frac{R}{r} \int_{A_1} \frac{1}{||x' - x_1||} da_1.
\end{aligned}$$

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 procedure `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 0, the error goes to 0. Also, all entries in the coefficient matrix are approximated to the same precision, which is a constant times P_ϵ.

In summary, our algorithm computes the potential factor matrix to within a fixed error tolerance. In the process it organizes the potential factor matrix into blocks. The estimated potential factor associated with each block has the same error as other blocks. Furthermore, the total number of blocks grows linearly in the number of elements.

3 Solve Linear Systems

3.1 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, please keep in mind $\mathbf{P}$ is the coefficient matrix, $\mathbf{q}$ is the given charge vector, and the product $\mathbf{Pq}$ is the potential vector which we want to compute, though the algorithm works for any matrix and vector.

In the following pseudo-codes, each panel `A` has two pointers `left` and `right`, pointing to the two sub-panels, and two fields `charge` and `potential`. For readers not familiar with recursive tree traversal, please see [2].

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

```
AddCharge(Panel Ai)
{
  if (Ai is leaf)
    Ai.charge = Qi;
  else {
    AddCharge(Ai.left);
    AddCharge(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 adding up the product of potential coefficient P_{ij} and 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 blocks in the coefficient matrix.

```
CollectPotential(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 interior nodes to the leaves. This is done by another depth first traversal of the tree which propagates potential down to the leaf nodes. Each interior 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;
      DistributePotential(Ai.left);
      DistributePotential(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 $n-1$ interior nodes. Therefore, the time is $O(n)$, where n is the number of panels that are not further subdivided.

3.2 Generalized Minimum Residual Method

We use the Generalized Minimum Residual (GMRES) method with restart [14] to solve the system of equations. The basic idea behind the GMRES method for solving an $n \times n$ linear system 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 [4], 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. Since we can multiply $\mathbf{P}$ with any vector in $O(n)$ time, each iteration can be computed in $O(n)$ time.

4 Experimental Results

The new algorithm is implemented and simulation results are reported in Tables 1 to 3. Both the new algorithm and the FastCap algorithm (for expansion order 0 and 2) are compiled and executed on a SUN Ultra SPARC Enterprise 4000. FastCap(0) is the fastest in the FastCap package, and is about twice as fast as FastCap(2). However FastCap(0) has 5% to 10% relative error with respect to FastCap(2).

The test examples are $k \times k$ bus crossing conductors for $k = 2$ to 6, taken from the FastCap paper [10]. Each bus in $k \times k$ 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 $111.27pF/m$, according to [10]. Our GMRES reduces the two-norm of the residual to 1% of the initial residual, the same condition used by the conjugate residual algorithm in FastCap.

The difference or 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 difference is estimated in the Frobenius norm [4]: $||\mathbf{C} - \mathbf{C}'||/||\mathbf{C}||$. This is the standard way to measure the difference between two matrices.

Table 1 compares the new algorithm with FastCap, and Table 2 and 3 show the first row of the capacitance matrix computed by the new algorithm and by FastCap. Here is a summary of the comparison:

1) $P_\epsilon = 0.01$. Compared with FastCap(2), the new algorithm is 100 to 40 times faster, and uses 1/100 to 1/60 of the memory. The error is less than 2.7% with respect to FastCap(2). Compared with FastCap(0), our algorithm is 40 to 14 times faster, and is three times more accurate. (The memory usage for FastCap(0) appears to be incorrect.)

2) $P_\epsilon = 0.003$ and compared with FastCap(2). the new algorithm is 5 to 3 times faster, and uses 1/6 to 1/8 of the memory. The error is less than 0.8% with respect to FastCap(2).

Table 1: Comparison for the bus problem. Time is seconds, memory is MB, and error is with respect to FastCap(2).

	Test Problems				
	2x2	3x3	4x4	5x5	6x6
			FastCap (0)		
Time	4.3	11.0	41.3	39.0	84.4
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	10.1	22.9	51.1	161.8	231.9
Memory	5.6	16	26	46	62
Panel	792	1620	2736	4140	5832
			New Algorithm ($P_\epsilon = 0.01$)		
Time	0.1	0.4	0.9	2.2	5.8
Memory	0.06	0.2	0.3	0.5	0.9
Panel	160	408	576	720	1584
Error	2.1%	1.7%	1.8%	1.7%	2.7%
			New Algorithm ($P_\epsilon = 0.003$)		
Time	2.0	7.0	19.2	45.2	83.6
Memory	0.9	2.0	3.9	6.4	9.4
Panel	1888	3504	6720	10960	13152
Error	0.4%	0.2%	0.8%	0.6%	0.4%

We do not have access to the precorrected FFT algorithm [13], the MEI algorithm [15], and the SVD algorithm [8]. However, based on their relative performance to FastCap for the $k \times k$ bus problem, our algorithm is much faster than these algorithms as well.

5 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 multilayered 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. The new algorithm is very simple, and provides many rooms for further improvements.

There are several major differences between our hierarchical algorithm and the multipole algorithm FastCap. 1) We partition the conductor surfaces according to an estimation of the coefficients for the actual problem, while FastCap considers the geometry of each conductor separately. As a result, we use fewer panels than FastCap does, yet our precision is higher for the same expansion order. 2) To achieve high precision, we divide the conductor surfaces into smaller panels, while FastCap uses high order expansion terms. Experimental results show that it is less expensive to use more panels than to use more expansion terms. 3) We organize

Table 2: Comparison for 4x4 bus problem. Error is with respect to FastCap(2).

	First Row of Capacitance Matrix (pF)								Error	Time	Memory
	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}		(sec)	(MB)
FastCap											
$l = 0$	394.5	-124.0	-0.175	-2.471	-52.15	-43.39	-43.08	-52.92	5.2%	41.3	86
$l = 2$	405.2	-137.8	-11.91	-8.079	-48.36	-40.09	-40.01	-48.45		51.1	26
New Algo											
$P_\epsilon = 0.01$	401.6	-139.6	-9.180	-6.480	-48.46	-37.80	-37.53	-48.46	1.8%	0.9	0.3
$P_\epsilon = 0.003$	407.9	-135.7	-13.08	-8.720	-48.79	-41.20	-41.21	-48.72	0.8%	19.2	3.9

Table 3: Comparison for 5x5 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													
$l = 0$	505.5	-142.8	-10.48	-20.31	2.21	-54.51	-50.13	-46.04	-50.32	-55.01	8.5%	39.0	24
$l = 2$	484.5	-166.1	-13.62	-6.17	-6.54	-48.84	-40.12	-40.12	-40.21	-48.90		161.8	46
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%	2.2	0.5
$P_\epsilon = 0.003$	486.3	-165.5	-15.05	-6.77	-6.01	-48.78	-40.49	-40.52	-40.52	-48.83	0.6%	45.2	6.4

the coefficient matrix more efficiently using the hierarchical data structure. Some feathers, such as the adaptive evaluation defined in FastCap [12], is free under our data structure.

Acknowledgment The authors wish to thank Weishi Sun, Steve Tate and Dian Zhou for discussions and references, and anonymous referees for improving the presentation.

References

[1] A. A. Appel, "An efficient program for many-body simulation," *SIAM J. Scientific and Statistical Computing*, Vol. 6, No. 1, 1985, 85–103.

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

[3] K. Esselink, "The order of Appel's algorithm," *Information Processing Letters*, Vol. 41, 1992, 141–147.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, 1989.

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

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

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

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

[9] S. Kapur and J. Zhao, "A fast method of moments solver for efficient parameter extraction of MCMs," *Proc. 34th DAC*, June 1997, 141–146.

[10] K. Nabors and J. White, "FastCap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. CAD*, Vol. 10, No. 11, Nov. 1991, 1447–1459.

[11] K. Nabors and J. White, "Multipole-accelerated 3-D capacitance extraction algorithms for structures with conformal dielectrics," *Proc. 29th DAC*, 1992, 710–715.

[12] K. 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, May 1994, 713–735.

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

[14] 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, July 1986, 856–869.

[15] W. Sun, W. W.-M. Dai, and W. Hong, "Fast parameters extraction of general 3-D interconnects using geometry independent measured equation of invariantce," *Proc. 33rd DAC*, 1996.

[16] D. R. Wilton, et al., "Potential integration for uniform and linear source distributions on polygonal and polyhedral domains," *IEEE Trans. Antennas and Propagation*, Vol. AP-32, No. 3, March 1984, 276–281.