IES^3: A Fast Integral Equation Solver for Efficient 3-Dimensional Extraction

Sharad Kapur David E. Long
Bell Laboratories, Lucent Technologies
Murray Hill, NJ 07974

Abstract

Integral equation techniques are often used to extract models of integrated circuit structures. This extraction involves solving a dense system of linear equations, and using direct solution methods is prohibitive for large problems. In this paper, we present IES^3 (pronounced "ice cube"), a fast Integral Equation Solver for three-dimensional problems with arbitrary kernels. Extraction methods based on IES^3 are substantially more efficient than existing multipole-based approaches.

1 Introduction

Parasitic extraction of IC packages and interconnect is now critical because of increases in operating frequencies and reductions in operating voltages. Extraction procedures based on integral equations [10, 13] are robust and have many advantages over finite-difference or finite-element schemes, including good conditioning, reduction in dimensionality, and the ability to treat arbitrary regions. However, they have one overriding disadvantage: the high cost of working with large dense matrices. In this paper, we present IES^3 ("ice cube"), an Integral Equation Solver for three-dimensional problems involving *arbitrary* kernels, and show that this new solver has significant performance advantages compared to competing approaches.

The fast multipole method (FMM) [3], while originally developed for particle simulation problems, can be combined with iterative techniques to solve the dense integral equation matrices that arise from the Laplace equation. Parameter extraction programs such as FastCap [7] and FastHenry [5] use the FMM for accelerating the dense matrix-vector products required by an iterative solver. Because the FMM is tailored to the $1/||x - x'||$ kernel, dealing with common situations such as layered dielectrics is difficult. For example, in FastCap this requires discretizing the dielectric boundaries and introducing extra unknowns for the polarization charge [8, 10]. This can result in a large increase in problem size. An extractor based on IES^3 has no such difficulty: it simply incorporates the variation in dielectrics into the Green's function.

Surprisingly, even for extraction problems that only involve the Laplace kernel, IES^3 is faster and more memory-efficient than FastCap, as we show in Section 4. This is because IES^3 provides greater compression than the FMM, though at the cost of a more expensive preprocessing phase. In extraction problems, the number of matrix-vector multiplies is typically large; hence the added compression provided by IES^3 more than makes up for the FMM's smaller startup cost. In contrast, in a particle simulation environment where the geometry (and hence the matrix) changes after each multiply, the FMM has a definite advantage [3].

We did not have access to the Precorrected-FFT version of FastCap [9], but based on the relative performance of both approaches to the FMM version of FastCap, we believe that our approach still has performance advantages. Also note that there have been recent advances in the FMM algorithms [2], but they have not yet been incorporated into extraction programs such as FastCap.

IES^3 uses the fact that large parts of the integral operator matrix are numerically *low rank*. The singular value decomposition (SVD) is an extremely effective tool for the compression of rank-deficient matrices [12]. Based on this observation, Kapur and Zhao [6] describe a scheme for recursively partitioning and compressing the matrix. This compressed representation is then used to compute matrix-vector products during an iterative solve. While these products are fast ($O(N \log N)$ time), the method suffers from an $O(N^2)$ preprocessing phase. This justifies its use only in situations where there are a large number of multiplies. The main result of this paper is a new interpolation-based construction procedure that reduces the preprocessing time to $O(N \log N)$. As a result, IES^3 is dramatically faster than the earlier method while still retaining the same degree of compression.

2 Formulation of the problem

We consider integral equations such as

$$\phi(x) = \int_{R'} G(x, x')\sigma(x')\, dR' \qquad (1)$$

where ϕ is the known "right-hand side" (for historical reasons, written on the left), σ is the unknown to be solved for, R is the region of integration and G is the kernel (or Green's function). For example, in the standard problem of capacitance extraction in three dimensions, ϕ is the potential, σ is the surface charge

0-89791-993-9/97 $10.00 © 1997 IEEE

density, R ranges over the surfaces of conductors, and G is the free-space Green's function $1/(4\pi\epsilon_0||x-x'||)$. To solve such a problem numerically, we must first discretize the region and reduce (1) to a matrix equation. In engineering applications, first-order collocation schemes are adequate. The domain R is subdivided into regions $\{R_1, R_2, \ldots, R_N\}$, a collocation point x_j is chosen in each region R_j, and σ is assumed to be piecewise constant over each region. Then (1) reduces to the following set of equations:

$$\phi(x_i) = \sum_{j=1}^{N} \left(\int_{R'_j} G(x_i, x')\, dR'_j \right) \sigma(x_j) \qquad 1 \le i \le N. \tag{2}$$

If we define the dense matrix $A = \{a_{ij}\}$ by

$$a_{ij} = \int_{R'_j} G(x_i, x')\, dR'_j, \qquad 1 \le i, j \le N. \tag{3}$$

then the system (2) is equivalent to the matrix equation $\phi = A\sigma$. While our algorithm does not assume any particular method of discretization and does not depend on the Green's function, we do assume that the problem has been reduced to matrix form. For the remainder of this paper, A will denote the $N \times N$ matrix (3) associated with the integral equation. For notational convenience, we assume that N is a power of 2.

3 Rapid matrix solution

Direct solution of the linear system $A\sigma = \phi$ via Gaussian Elimination requires $O(N^2)$ storage and $O(N^3)$ time and is impractical for large problems. Fortunately, typical systems arising from integral equations almost always have an asymptotically bounded condition number and can be solved by Krylov-subspace iterative schemes such as Conjugate Gradient [12] or GMRES [11]. Iterative solvers require application of the matrix A to a sequence of recursively generated vectors. The dominant costs become the $O(N^2)$ time and space required for constructing and storing the matrix and the $O(N^2)$ time required for each matrix-vector product. In this section, we describe IES^3, a kernel-independent algorithm that reduces each of these costs to $O(N \log N)$ for typical problems.

3.1 A simple example

The idea behind our algorithm (and other fast algorithms such as the FMM) is to exploit the fact that typical Green's functions vary smoothly with distance. Consider the situation depicted in Figure 1. Suppose that $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ are two sets of points in R^3 separated by a distance d. Let $H(x, y)$ be the function giving the interaction between the points. Then the influence of Y on X is can be computed by multiplication of the $n \times n$ matrix $B = \{b_{ij}\} = \{H(x_i, y_j)\}$ with a vector. Direct evaluation of this product would require n^2 operations. However, if d is relatively large, and the function H is smooth, then the numerical rank of the matrix B is small compared to n. Intuitively the influence due to a point y_j is very similar to the influence of its neighbors. As a result, the columns of B are nearly linearly dependent.

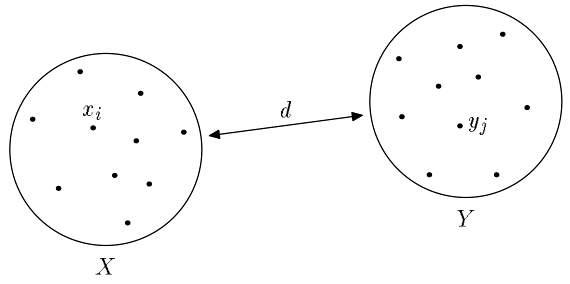


Figure 1: Well-separated regions

We take advantage of this fact using the Singular Value Decomposition (SVD) [12]. The SVD of B is given by

$$B = USW$$

where

$$UU^* = W^*W = I,$$

and $S = \text{diag}(s_1, \ldots, s_n)$ with $s_1 \ge s_2 \ge \cdots \ge s_n \ge 0$. The numerical rank of a matrix to a precision ϵ is the integer r such that $s_r/s_1 < \epsilon$. Clearly, if B is of rank r then it can be approximated as[1]

$$\tilde{B} = U(:, 1{:}r)S(1{:}r, 1{:}r)W(1{:}r, :)$$

with

$$||B - \tilde{B}|| < \epsilon.$$

Given the SVD of the matrix B of rank r, a matrix-vector product can be computed in $(2n+1)r$ operations by multiplying the vector in turn by $W(1{:}r, :)$, $S(1{:}r, 1{:}r)$, and $U(:, 1{:}r)$. When $r \ll n$ it is much more efficient to multiply with B in this manner.

Use of the SVD to represent B requires no prior knowledge of the function H, in contrast to the multipole approach which is based on a Taylor series approximation of H. This added flexibility does not reduce the compression that is achieved. In fact, as our numerical experiments in Section 4 show, we obtain *substantially greater compression* than the multipole schemes.

Even though we use the SVD for the theoretical development of our algorithm, it is sufficient to use the Gram-Schmidt process [12]. Given an $n \times n$ matrix B and a user specified tolerance ϵ, the Gram-Schmidt process computes the numerical rank r of B and an $n \times r$ orthogonal matrix U which spans the column space of B. B can be approximated as UV where $V = U^*B$. Decomposing B in this way is faster than computing the SVD of B.

[1]We use Matlab notation for submatrices. $U(:, 1{:}r)$ denotes the submatrix consisting of columns 1 through r of U. Similarly, $W(1{:}r, :)$ denotes the submatrix consisting of the first r rows of W.

3.2 Ordering

To compress the integral operator matrix A in (3), which represents both *near and far* interactions, we need to identify large submatrices that correspond to interactions between well-separated regions. (As we saw in the previous subsection, such submatrices can be compactly represented as low-rank UV products.) In some situations, this is easily achieved. For example, in a one-dimensional problem, the natural ordering of points by their position in space is a good one, yielding a matrix whose *rank map* is shown in Figure 2. (This example is drawn from Section 3-2 of [4]. The problem is to solve for the surface current induced on a conducting cylinder by an impressed electric field, and the Green's function G is the Helmholtz kernel $H_0^2(k\|x - x'\|)$.) The rank map shows the partitioning of the matrix A into submatrices and the rank of each submatrix. The structure of this rank map is typical in problems where the storage requirements drop from $O(N^2)$ to $O(N \log N)$.

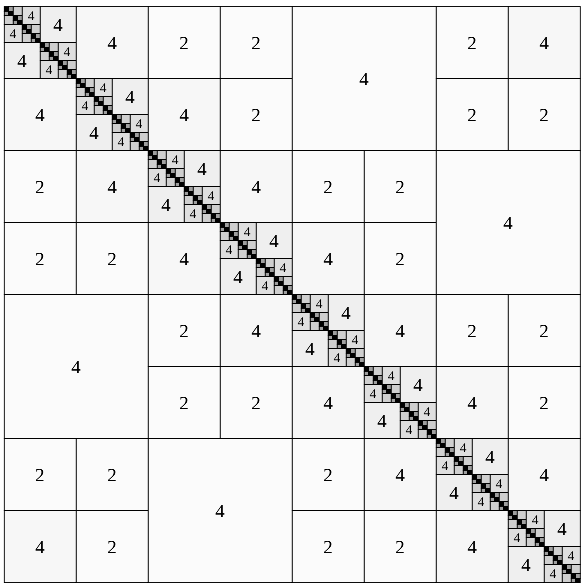


Figure 2: Rank map for a one-dimensional Helmholtz kernel ($N = 2048$)

In general, our goal is to find a permutation matrix P such that $\tilde{A} = P^{-1}AP$ is highly compressible. Ideally, all strong interactions should be close to the diagonal in $\tilde{A}$, as in Figure 2. For two- and three-dimensional problems there is generally no permutation P that can completely accomplish this. Nevertheless, the following heuristic based on recursive subdivision yields excellent results and requires only $O(N \log N)$ time. We assume that there is a spatial coordinate associated with each element in the discretization. Under this assumption, the problem reduces to one of ordering a set of points such that points that are close in space are also close in the ordering. For notational simplicity, we present the algorithm for the two dimensional case.

1. If the current set of points S is the singleton $\{(x, y)\}$ then return the sequence $\langle(x, y)\rangle$.

2. Otherwise compute the bounding box for the points in S. We assume without loss of generality that the longest dimension corresponds to the x-axis. Equally divide S into two disjoint sets S_1 and S_2 such that $x_1 \leq x_2$ for all points $(x_1, y_1) \in S_1$ and $(x_2, y_2) \in S_2$. Recursively order S_1 and S_2 and concatenate the results to obtain the ordering for S.

The rank map for a three-dimensional problem after ordering with this technique is shown in Figure 4. This problem corresponds to capacitance extraction for structure of Figure 3 [8]. Although there are some strong interactions off the diagonal, the rank map is similar to the one in Figure 2. For the remainder of the paper, we will assume that all matrices have been ordered in this fashion.

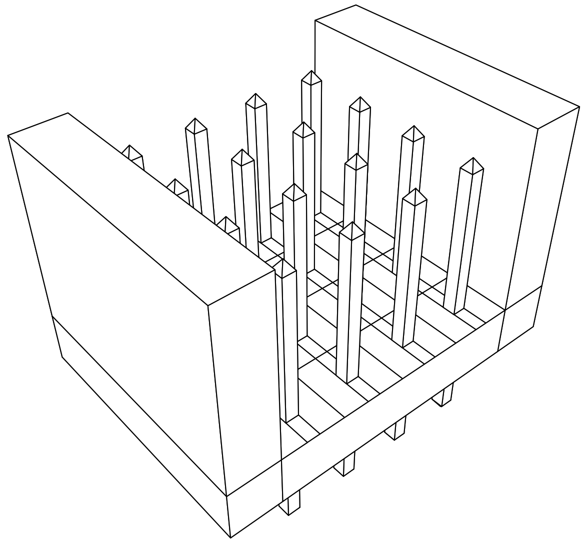


Figure 3: The backplane connector example from [8]

3.3 Compression in $O(N^2)$ time

In this subsection, we briefly describe the algorithm of [6] for the recursive UV decomposition of A. The main difference between the algorithm of [6] and IES^3 is in the construction of the compressed representation of the matrix. However, the basic structure of the two algorithms is the same: both partition the matrix by a recursive subdivision scheme. In [6], the construction proceeds as follows.

1. If the dimension n of the current submatrix B is less than b, where b is a small fixed number, then:

   (a) Use the Gram-Schmidt procedure to obtain an orthonormal column basis to a user specified precision ϵ. If the size of the basis r is greater than $n/2$ then B is simply stored as a dense matrix.

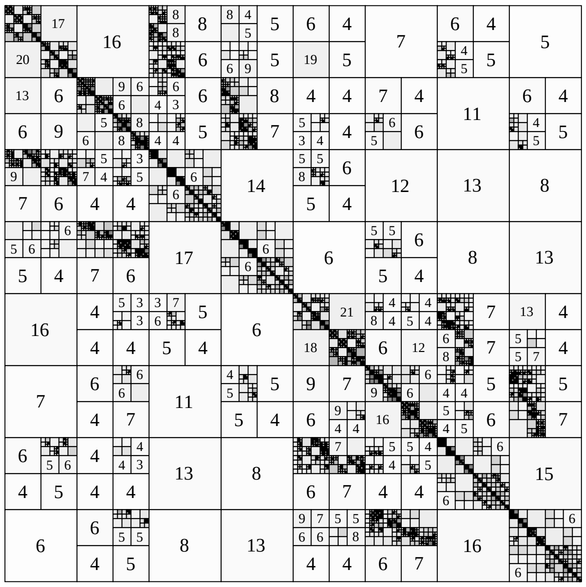


Figure 4: The rank map for the backplane connector example (discretized into $N = 10672$ panels)

(b) Otherwise let U be the $n \times r$ matrix whose columns are the basis. Compute $V = U^* B$.

2. If n is at least b, then construct the representation of the four submatrices $B(1{:}n/2, 1{:}n/2)$, $B(1{:}n/2, n/2+1{:}n)$, $B(n/2+1{:}n, 1{:}n/2)$, and $B(n/2+1{:}n, n/2+1{:}n)$. If these child submatrices are all low rank, i.e., are represented as UV decompositions, then

   (a) Construct a decomposition UV for B by merging (as described below).

   (b) If the size of this decomposition is smaller than the sum of the sizes of the child submatrix representations then represent B by U and V. Otherwise represent B by the child submatrices.

3. If some submatrix is not low rank, then represent B by its four children.

The merging process constructs a decomposition $U_{12}V_{12}$ for the matrix

$$\begin{bmatrix} U_1 V_1 \\ U_2 V_2 \end{bmatrix}$$

where U_1, V_1, U_2, and V_2 have dimensions $n \times r_1$, $r_1 \times n$, $n \times r_2$, and $r_2 \times n$ respectively, and where U_1 and U_2 are both orthogonal. We run the Gram-Schmidt procedure on the rows of V_1 and V_2 to obtain a row basis of size r. V_{12} is the $r \times n$ matrix whose rows are this basis. The $2n \times r$ matrix U_{12} is given by

$$\begin{bmatrix} U_1 V_1 V^* \\ U_2 V_2 V^* \end{bmatrix}.$$

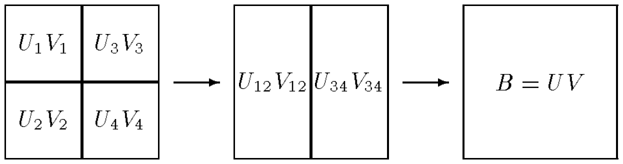


Figure 5: The merging process.

Note that the matrix V_{12} generated by this procedure is orthogonal. Three merges (two vertical and one horizontal) yield the UV representation of B in Step 2a above. As a result of the final horizontal merge, U is orthogonal. The process is illustrated in Figure 5. Note that directly using the Gram-Schmidt process to construct the UV decomposition of B would require $O(n^2 r)$ operations. In contrast, merging requires only $O(nr^2)$ steps. For rank deficient matrices ($r \ll n$) merging is much more efficient.

Computing a matrix-vector product with the final compressed representation of is done with the following recursive algorithm.

1. Let B of size $n \times n$ be the current submatrix, x be the current input vector, and y be the current output vector. If B is represented densely, then compute $y = Bx$ directly.

2. If B is represented as UV, then compute $y = U(Vx)$.

3. If B is represented as a combination of four submatrices
$$\begin{bmatrix} B_1 & B_3 \\ B_2 & B_4 \end{bmatrix},$$
then $y(1{:}n/2) = B_1 x(1{:}n/2) + B_3 x(n/2+1{:}n)$ and $y(n/2+1{:}n) = B_2 x(1{:}n/2) + B_4 x(n/2+1{:}n)$.

This summarizes the algorithm described in [6] for compression of A and the computation of matrix-vector products. Empirically, the compression obtained is much higher than that achieved by the multipole algorithms. This leads to a faster matrix-vector multiply. However, the construction of the UV decompositions requires at least $O(N^2)$ operations while constructing the multipole representation requires only $O(N \log N)$ steps. In situations where the construction cost can be amortized over a large number of solves (as is the case in [6]), the UV representation is more efficient. But when there are few solves, the multipole approach is faster overall. In the following subsection we present an improved construction procedure that requires subquadratic time. This makes our method substantially faster than the multipole algorithms in most cases.

3.4 Compression in subquadratic time

The key idea behind IES^3 is to exploit smoothness in the entries of the interaction matrix. By using interpolation, we can build the UV decomposition without examining all n^2 entries.

Consider constructing a UV decomposition of the interaction matrix B for the situation shown in Figure 6. Let S_X and S_Y be sets of p sampling points in X and Y respectively. The idea for reconstructing B is to use the influence of S_Y on X to construct a column basis for B. Then the influence of Y on S_X is used to express each column of B in terms of this basis. The columns of B that have not been completely sampled (which correspond to points in $Y - S_Y$) are interpolated based on the sampling S_X of rows. More specifically, the influence due to any point in Y on X is approximated by a linear combination of the influences of the sampling points S_Y. Hence if C is the $n \times p$ matrix of columns of B corresponding to points in S_Y, then a column basis for C should also span the columns of B (the validity of this is discussed below). Define U to be the $n \times r$ matrix obtained by running the Gram-Schmidt process on the columns of C. To get the coefficients of the columns of B in the basis represented by U, we use the sampling points S_X. Let R be the $p \times n$ matrix of rows of B corresponding to points in S_X and let $\tilde{U}$ be the $p \times r$ matrix of rows of U corresponding to these same points. Finally, define V to be the solution (in the least squares sense) to $\tilde{U}V = R$. Then B is approximated by UV.

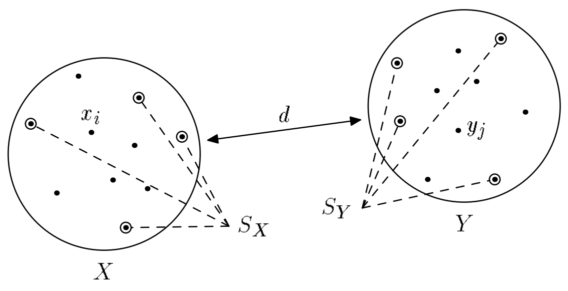


Figure 6: Well-separated regions with sampling points

In the old algorithm, U was the result of running the Gram-Schmidt process on all the columns of B. Clearly, obtaining U from C instead of B is only justified when there are sufficient sampling points to reconstruct the interaction. The issue is akin to interpolation of an unknown function. When the function is smooth (in the sense that the derivatives over the interval have a known bound), the accuracy of the interpolation can be rigorously controlled by the number and location of sampling points [12]. However, when designing a general-purpose algorithm, requiring bounds on the derivatives of the function is usually very conservative. Empirically, given the assumptions that the function is smooth, the sets are well separated, and the sampling is reasonably uniform, the interpolation can be made arbitrarily accurate. Hence, instead of derivative information, our algorithm uses a statically-determined map which indicates the low-rank regions of the matrix A and information about appropriate sampling points within each such region. We provide default methods that compute this information based purely on the spatial coordinates of elements. For typical kernels these default routines suffice. As an example, a static map of the interactions for the backplane connector mentioned in Subsection 3.2 is shown in Figure 7. Figure 4 shows the rank map for the same example; note the close correspondence between the two. Also note that the static map is conservative, in that many of the small submatrices shown on the static map have merged into larger low-rank matrices in Figure 4.

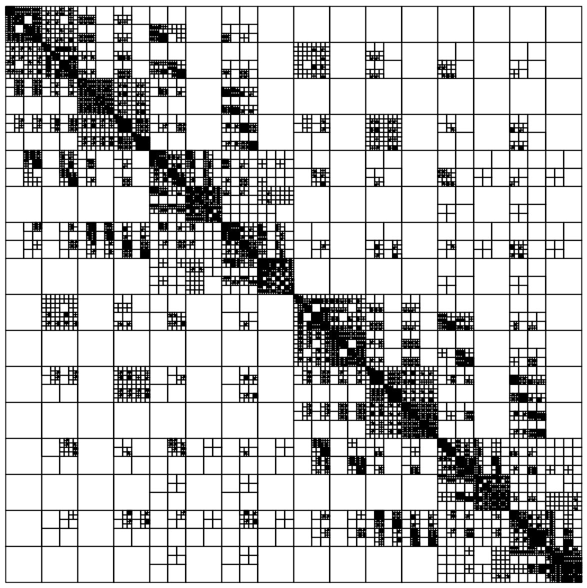


Figure 7: The static sampling map for the backplane connector example

The new algorithm is exactly the same as the one in Subsection 3.3, except for the addition of a new base case:

> If the statically determined map indicates that B should be low rank, then sample B and construct the UV decomposition as discussed at the start of this subsection.

For a (statically-identified low-rank) submatrix B of size $n \times n$ and rank r, the algorithm takes only $O(r^2 n)$ operations to construct U and V. As a result, this simple change dramatically reduces the construction time for the compressed representation of A. For general problems it is difficult to give a bound for the complexity, but for the simple case of a one-dimensional problem (where we assume that the rank of each of the submatrices is r), the time is $O(r^2 N \log N)$.

4 Experimental results

In this section, we present experimental results for IES^3. All tests were run on an SGI machine (200 MHz R10000 CPU).

4.1 Comparison to FastCap

We begin by comparing IES^3 to the FMM-based FastCap program (version 2) [7, 8]. To make a fair comparison, we modified FastCap to use our algorithm within FastCap's iterative solver. The discretization, matrix formulation, and preconditioning were unchanged. The relative tolerance was set to give the same accuracy as FastCap's default two-term multipole expansions. Results for a number of the standard examples supplied with FastCap are presented in Table 1. For FastCap we computed the memory requirements by subtracting the preconditioner memory and the "miscellaneous" memory from the total memory use reported. Error is determined by running the same examples at a much higher accuracy and computing $||C - \tilde{C}||/||C||$ where C and $\tilde{C}$ are the high- and low-accuracy capacitance matrices, respectively. For the RAM cell example, the default tolerances only yielded one digit of accuracy for both methods. The example was run with tighter tolerances for both.

The first two examples in the table involve conductors in free space, while the last two also include dielectric interfaces. For examples with dielectrics, FastCap uses a finite-difference approximation to compute normal components of the electric field at the dielectric interfaces. This computation is done after the potential is evaluated, since incorporating it directly would require substantial alterations to the multipole algorithm (as discussed in the appendix of [8]). Because IES^3 is not restricted to any specific kernel, we chose to incorporate the same finite-difference computation directly into the matrix. We could as easily have analytically differentiated the potential to avoid the inaccuracies inherent in finite-differencing.

The memory required by IES^3 can be as little as one-fifth that required by FastCap. As a consequence, IES^3 is much faster than FastCap for larger problems and those involving many conductors. Our startup overhead is still substantial for small examples such as the via structure. For medium-sized examples such as the backplane connector, IES^3 is almost two and one-half times faster.

4.2 Simulation of an RF test socket

As an example of a problem involving a non-classical kernel, we consider capacitance extraction for an RF IC test socket. Figure 8 shows a cross-sectional view of one side of the socket. The socket is mounted on a single-layer board with a ground plane. In the figure, $\epsilon_1 = 3.08$ is the effective dielectric for the socket and $\epsilon_2 = 9.8$ is the dielectric constant of the circuit board. Figure 9 is a three-dimensional mesh of the socket, computed using a surface mesh generator[2]. The picture also shows the solution for the charge distribution on the socket when a potential difference of one volt is applied to the first pin. To decrease the size of the problem, we use an adaptive mesh, with finer discretization at corners and edges. This reflects the expected rapid variation in charge density in these regions. Due to the large separation between the two sides of the socket, it was sufficient to model only one side.

[2]Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator, by Jonathan Shewchuk

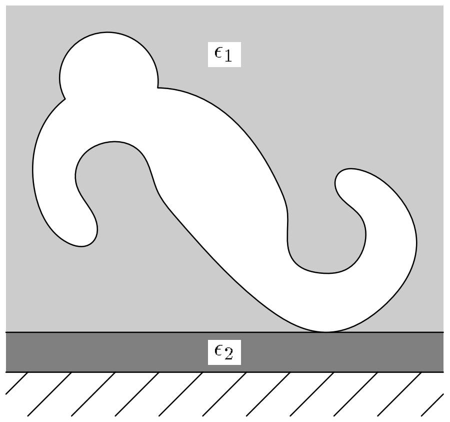


Figure 8: Cross section of one side an RF IC socket, showing dielectric layers and ground plane

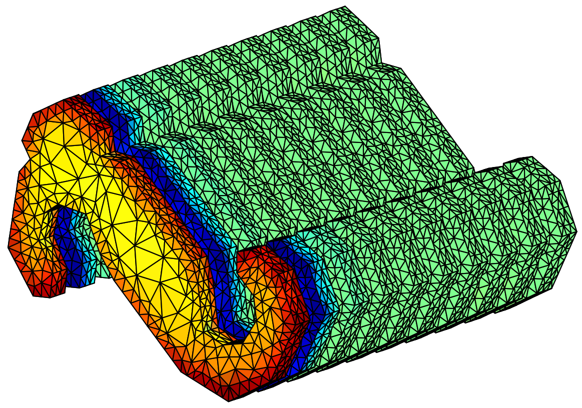


Figure 9: Discretization and charge distribution of the socket with one volt applied to pin one

A standard first-order collocation scheme of the type discussed in Section 2 was used together with a modified Green's function incorporating the layered dielectrics and ground plane [13]. This Green's function takes the form of an infinite series. Fortunately, for dielectric constants that are relatively close, the series is rapidly convergent and can be numerically approximated with ten to fifteen terms.

For the construction of the matrix A, we use analytic integration for the singular integrals when the collocation point lies in the triangular panel and low-order Gaussian integration when the point does not.

Experimental results for the capacitance extraction are given in Table 2 and Figure 10. For lower levels of discretization, we compare three methods: standard Gaussian Elimination (G.E. in the table), GMRES using a direct matrix multiply, and our algorithm. For direct GMRES and IES^3, a tolerance of 10^{-2} was used. The memory and time required to construct the matrix (for both Gaussian Elimination and GMRES) is shown in the second and third columns. Memory is in MB and time is in CPU seconds. The "Error"

		FastCap			IES^3		
Example	Panels	Time (sec)	Mem. (MB)	Error	Time	Mem.	Error
5x5 bus crossing	7360	185	66	3e-3	105	26	6e-3
via structure	6126	45	37	1e-3	60	22	4e-3
backplane connector	10672	675	165	4e-3	280	45	3e-3
RAM cell	6129	480	135	1e-2	195	26	1e-2

Table 1: Comparison of IES^3 to the FMM-based scheme in FastCap

	Direct Methods					IES^3					
			G.E.	GMRES							
Panels	Mem.	Cons.	Solve	Solve	Error	Mem.	Cons.	Solve	Error	C_{11}	C_{12}
1320	13	40	10	20	1e-3	2	20	10	3e-3	905	-474
2500	48	140	70	90	1e-3	5	65	25	4e-3	906	-484
4440	150	435	385	290	1e-3	11	150	50	5e-3	898	-477
8500	(550)	(1650)	(2700)	(1050)	—	24	375	90	—	892	-469
15690	(1870)	(5425)	(17000)	(3600)	—	54	865	230	—	891	-467

Table 2: Comparison of IES^3 to direct methods

columns give relative errors in the capacitance matrices, as compared to Gaussian Elimination. The "Solve" columns show CPU times for doing ten solves (one per pin). The construction time for IES^3 includes the time to compress the matrix. The capacitance matrix entries C_{11} and C_{12} are in fF. The numbers in parenthesis are extrapolated values.

The capacitance matrix entries have converged to two digits of accuracy at a discretization of $N = 15690$. At this level of discretization, memory requirements made the direct methods impractical. Even if memory were available, IES^3 would be twenty times faster than Gaussian Elimination and eight times faster than GMRES. Note that both the time and memory required by IES^3 are growing almost linearly with problem size. The total time required for the largest problem is less than twenty minutes.

The extracted capacitance values were used for building an equivalent circuit model for the socket. The coupling between pins was measured using a network analyzer. Figure 11 shows predicted crosstalk with our model, as compared to measurements. The two agree to about 3 dB.

As an aside, it is interesting that a carefully-engineered and machine-optimized Gaussian Elimination code [1] is extremely competitive with iterative techniques for medium-sized problems. This is because modern machines tend to be memory bound rather than CPU bound. In such an environment, we feel that the primary benefit of codes such as IES^3 is the dramatic memory reduction that they offer.

5 Conclusions

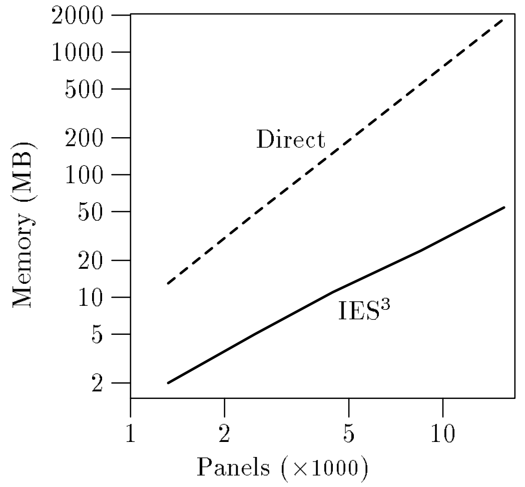
In this paper we presented IES^3, a fast general-purpose integral equation solver. The algorithm uses a combination of SVD-based compression and an iterative solver. The compressed representation of the integral operator matrix is constructed in $O(N \log N)$ time by combining a recursive partitioning strategy with interpolation of the function that defines the matrix entries. Unlike the kernel-specific multipole algorithms, IES^3 does not depend on the particular form of the Green's function. Further, our scheme gives higher compression even for those kernels for which the multipole methods were developed. We demonstrated IES^3 on three-dimensional capacitance extraction problems. On large problems with Laplace kernels, our method uses half the time of FastCap and only about one-fifth the memory. Because IES^3 is not kernel-specific, we solve examples involving multiple dielectric layers using a modified Green's function, thereby restricting the discretization only to conductor surfaces. On one such example, an RF test socket, IES^3 required an order of magnitude less time and thirty five times less memory than direct GMRES. While we have discussed IES^3 only in the context of extraction problems, it should also be useful for other purposes, such as full wave problems involving Helmholtz kernels.

Acknowledgements

Prashant Singh provided the description and measurements for the socket example. We thank Jacob White and his group at MIT for making the FastCap program and the associated examples available for benchmarking purposes.

References

[1] E. Anderson et al. *LAPACK User's Guide, Release 2.0*. SIAM Publications, Philadelphia, 2nd edition, 1995.



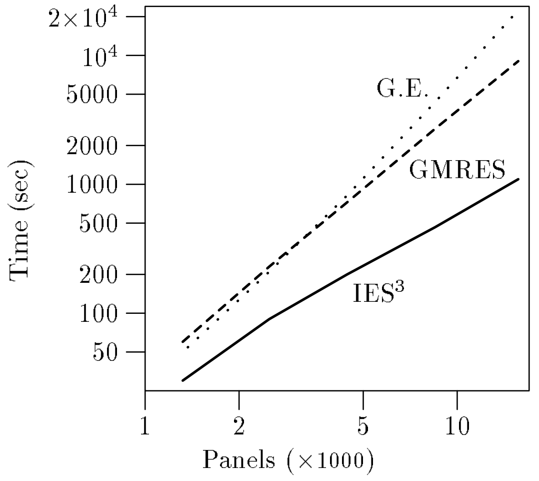


Figure 10: Time and memory requirements for the RF test socket example at various levels of discretization

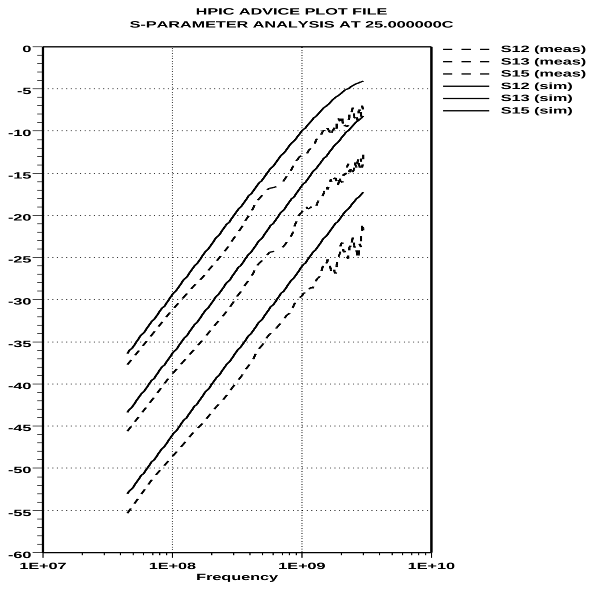


Figure 11: Comparison of simulated S parameters versus measurements

[2] L. Greengard and V. Rohklin. A new version of the fast multipole method for the Laplace equation in three dimensions. Technical Report YALEU/DCS/RR-1115, Yale University Department of Computer Science, September 1996.

[3] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, December 1987.

[4] R. F. Harrington. *Field Computation by Moment Methods*. IEEE Press, New York, 1991.

[5] M. Kamon, M. J. Tsuk, and J.White. FASTHENRY: A multipole-accelerated 3-d inductance extraction program. *IEEE Transactions on Microwave Theory and Techniques*, 42(9):1750–1758, September 1994.

[6] S. Kapur and J. Zhao. A fast method of moments solver for efficient parameter extraction of MCMs. In *34th Design Automation Conference*, pages 141–146, June 1997.

[7] K. Nabors and J. White. Fastcap: A multipole-accelerated 3-d capacitance extraction program. *IEEE Transactions on Computer-Aided Design*, 10(10):1447–1459, November 1991.

[8] K. Nabors and J. White. Multipole-accelerated capacitance extraction for 3-d structures with multiple dielectrics. *IEEE Transactions on Circuits and Systems*, 39(11):946–954, November 1992.

[9] J. R. Philips and J. White. A precorrected-FFT method for capacitance extraction of complicated 3-d structures. In *International Conference on Computer-Aided Design*, November 1994.

[10] S. M. Rao, T. K. Sarkar, and R. F. Harrington. The electrostatic field of conducting bodies in multiple dielectric media. *IEEE Transactions on Microwave Theory and Techniques*, 32(11):1441–1448, November 1984.

[11] Y. Saad and M. H. Shultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[12] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1979.

[13] W. T. Weeks. Calculation of coefficients of capacitance of multiconductor transmission lines in the presence of a dielectric interface. *IEEE Transactions on Microwave Theory and Techniques*, 18(1):35–43, January 1970.