A Parallel Precorrected FFT Based Capacitance Extraction Program for Signal Integrity Analysis

N. R. Aluru, V. B. Nadkarni and J. White
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139

Abstract—In order to optimize interconnect to avoid signal integrity problems, very fast and accurate 3-D capacitance extraction is essential. Fast algorithms, such as the multipole or precorrected Fast Fourier Transform (FFT) accelerated methods in programs like FASTCAP, must be combined with techniques to exploit the emerging cluster-of-workstation based parallel computers like the IBM SP2. In this paper, we examine parallelizing the precorrected FFT algorithm for 3-D capacitance extraction and present several algorithms for balancing workload and reducing communication time. Results from a prototype implementation on an eight processor IBM SP2 are presented for several test examples, and the largest of these examples achieves nearly linear parallel speed-up.

I. Introduction

Finding signal integrity problems in high performance integrated circuits and integrated circuit packaging is extremely difficult, primarily because these problems are typically created by the detailed interactions between hundreds of conductors. Although 3-D simulation tools can help designers find signal integrity problems, even the fastest of these tools running on a scientific workstation are too slow to allow a designer to quickly investigate a variety of conductor layouts. Therefore, reducing analysis turn-around time is critical, and can result in 3-D simulation being used as part of design optimization rather than just a-posteriori verification.

One approach to reducing simulation turn-around time is to exploit the recently developed parallel computers based on clusters of scientific workstations, like the IBM SP2. Achieving efficiency using the distributed parallelism available from clusters of workstations is more challenging than using a specialized parallel supercomputer. Even with an enhanced communication networks, it will not be possible to achieve good parallel efficiency with workstation clusters by relying on huge volumes of short messages.

In this paper we demonstrate that turn-around time for fast 3-D capacitance extraction can be substantially reduced using a cluster-of-workstations based parallel computer. In the next section we present background on 3-D capacitance extraction, and describe one of the recently developed fast methods for computing capacitances, the precorrected-FFT accelerated method [1]. In Section 3 we describe the algorithms used to parallelize the precorrected-FFT based capacitance extraction program and in Section 4 computational results are presented. Conclusions are given in Section 5.

This research was supported by ARPA under ONR contract DABT63-94-C-0053 and FBI contract J-FBI-92-196, by SRC under contract SJ-558, and by grants from IBM and Digital Equipment Corporation.

II. Background

A. Problem Formulation

The capacitance of an m-conductor geometry is summarized by an $m \times m$ symmetric matrix C. The j-th column of the capacitance matrix is determined by finding the surface charges on each conductor by raising conductor j to one volt and grounding the rest. The charge on each conductor can be determined by solving the integral equation [2]

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

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

A standard approach to numerically solving (1) for σ is to use a piece-wise constant collocation scheme. The conductor surfaces are broken into n small panels, and it is assumed that on each panel i, a charge, q_i, is uniformly distributed. For each panel, an equation is written which relates the known potential at the center of that i-th panel, denoted $\overline{p}_i$, to the sum of the contributions to that potential from the n charge distributions on all n panels. The result is a dense linear system,

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

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

$$P_{ij} = \frac{1}{a_j} \int_{panel_j} \frac{1}{4\pi\epsilon_0 \|x_i - x'\|} da', \quad (3)$$

where x_i is the center of the i-th panel and a_j is the area of the j-th panel.

B. The Precorrected FFT Method

When an iterative algorithm like GMRES [3] is used to solve (2), the major cost of the algorithm is the order n^2 operations required to form the dense matrix P and the order n^2 operations to compute the dense matrix-vector products for each GMRES iteration. Sparsification techniques, such as fast multipole algorithms [4], [5] or precorrected-FFT methods [1], avoid forming P and can be used to compute dense matrix-vector products in order n or $nlogn$ operations. Empirical studies on typical 3-D capacitance extraction problems indicate that precorrected-FFT methods generally use the least memory and CPU time.

In the precorrected-FFT method, once three-dimensional conductor geometry is discretized into panels, the parallepiped containing the entire problem is subdivided into an $k \times l \times m$ array of small cubes so that each small cube contains only a few panels. Several sparsification techniques for P are based on the idea of directly computing only those portions of Pq associated with interactions between panels in neighboring cubes. The distant interactions can be computed by exploiting the fact that evaluation points distant from a cube can be

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

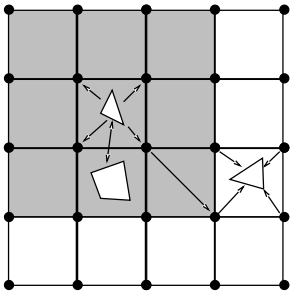


Fig. 1. 2-D Pictorial representation of the precorrected FFT algorithm. Interactions with nearby panels (grey area) are computed directly, interactions between distant panels are computed using the grid.

accurately computed by representing the given cube's charge distribution using a small number of weighted point charges. If the point charges all lie on a uniform grid, then FFT can be used to compute the potential at these grid points due to the grid charges.

Specifically, Pq may be approximated in order $n \log n$ operations in four steps: (1) directly compute nearby interactions, (2) project the panel charges onto a uniform grid of point charges, (3) compute the grid potentials due to grid charges using an FFT, (4) interpolate the grid potentials onto the panels. This four-step process is summarized in Figure 1. The calculations using the FFT on the grid duplicate the calculation of the nearby interactions and this is precorrected by modifying the computation of direct interactions.

As an example in step (2) above, charges in a given cube, a, are projected onto a $3 \times 3 \times 3$ array of grid points. Next, test points are selected outside cube a's surface and the potentials due to grid charges are forced to match the potential due to the cube's actual charge distribution at the test points. Since such collocation equations are linear in the charge distribution, this projection operation which generates a subset of the grid charges, denoted q_a^g, can be represented as a matrix, W_a, operating on a vector representing the panel charges in cube a, q_a i.e. $q_a^g = W_a q_a$

Once the charge has been projected to a grid, computing the potentials at the grid points due to the grid charges is a three-dimensional convolution. This is denoted as

$$\psi_g(i,j,k) = \sum_{i',j',k'} h(i-i', j-j', k-k') q_g(i',j',k'). \qquad (4)$$

where i, j, k and i', j', k' are triplets specifying the grid points, ψ_g is the vector of grid potentials, q_g is the vector of grid charges, and $h(i-i', j-j', k-k')$ is the inverse distance between grid points i, j, k and i', j', k'. The above convolution can be computed in $O(N \log N)$ time, where N is the number of grid charges, by using the FFT.

Once the grid potentials have been computed, they can be interpolated to the panels in each cube using the transpose of W_a. Therefore, projection, followed by convolution, followed by interpolation, can be represented as

$$\psi_{fft} = W^t H W q, \qquad (5)$$

where q is the vector of panel charges, ψ_{fft} is an approximation to the panel potentials, W is the concatenation of the W_a's for each cube, and H is the matrix representing the convolution in (4).

In ψ_{fft} of (5), the portions of Pq associated with neighboring cube interactions have already been computed, though this close interaction has been poorly approximated in the projection/interpolation. Denoting $P_{a,b}$ as the portion of P associated with the interaction between neighboring cubes a and b, $H_{a,b}$ as the potential at grid points in cube

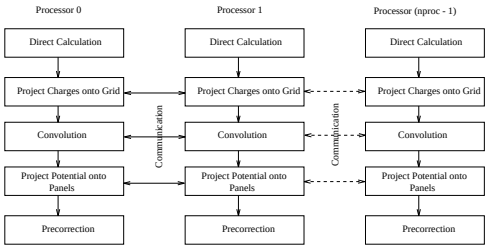


Fig. 2. Parallel computation of the matrix-vector product in a precorrected FFT approach

a due to grid charges in cube b, ψ_a and q_b as the panel potentials and charges in cubes a and b respectively, a better approximation to ψ_a is

$$\psi_a = \psi_{a_{fft}} + \left(P_{a,b} - W_a^t H_{a,b} W_b\right) q_b \qquad (6)$$

where $P_{a,b}^{cor} \equiv P_{a,b} - W_a^t H_{a,b} W_b$ is the precorrected direct interaction operator. When used in conjunction with the grid charge representation $P_{a,b}^{cor}$ results in exact calculation of the interactions of nearby panels.

III. Parallel Approach

The most computationally expensive step, and the one which must be efficiently parallelized, is the matrix-vector product using the precorrected-FFT method. It is also true that in order to avoid a serial bottleneck, it is important to parallelize some of the inner products used in the GMRES algorithm, but this is straight-forward. In the subsections below, we describe how the problem is decomposed onto different processors and then how each of the steps of the precorrected-FFT method are parallelized. An overview of the approach to parallelization is shown schematically in Figure 2.

A. Problem Decomposition

Finding an efficient task decomposition for the precorrected FFT method is difficult, as the task decomposition must minimize interprocessor communication while simultaneously balancing the operations required to compute the direct interactions, the grid projection/interpolation, and the grid convolution.

A single problem decomposition can not insure all sets of operations in the precorrected-FFT method are simultaneously balanced. Balancing the direct computation implies balancing the number of nearby interactions, balancing the projection/interpolation implies associating the same number of panels with each processor, and balancing the grid convolution implies associating the same number of grid points to each processor. One approach to resolving this difficulty is to consider separate decomposition algorithms for each part of the precorrrected-FFT algorithm, but the advantage of the better load balancing might be lost due to additional communication costs associated with realigning the problem decomposition. In this paper, we take the approach of picking a single decomposition which best fits the convolution algorithm, that of balancing the number of grid points per processor, as the convolution is the most computationally expensive part. As we note in the results section, it is more important to balance the grid convolution and load imbalances in the number of direct interactions and charges may not be significant as they take only a small fraction of the total cpu time.

The decomposition algorithm simply allocates equal number of grid planes to each processor. The partitioning is performed along the z-direction or the third FFT dimension and the number of planes

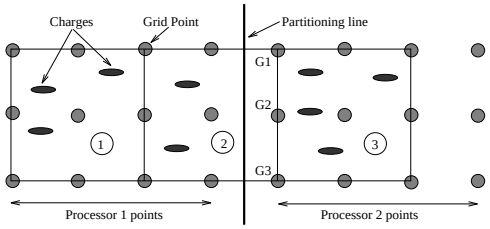


Fig. 3. 2-D Pictorial representation of the decomposition algorithm

allocated to each processor is computed as

$$\#planes = \frac{\#z - direction\ grid\ points}{nproc} \tag{7}$$

where $nproc$ is the number of processors and the number of z-direction grid points is a power of two. Figure 3 illustrates a two-dimensional example with 3 squares and 8 lines of grid points for convolution. Each square contains 9 grid points and a two-way partitioning of the problem puts 4 lines of points per processor, splitting the second square (or cube in 3D) in the middle. The charges in cube 1 and cube 2 are associated with processor 1 and the charges in cube 3 are associated with processor 2. The first line (plane in 3D) of points in each processor (except the first processor) is shared by cubes belonging to processors i and $i+1$. The communication associated with this sharing is described in the section below on projection.

B. Precorrected Direct Interactions

The precorrected direct interaction between panels in a cube and panels in the cube's neighbor is computed directly. If all the neighbors of a cube lie in the same processor, then no inter-processor communication is needed to compute precorrected direct interactions. However, if a cube's neighbor lies in a remote processor, information about the panels in the neighbor must be communicated to the cube's processor. This can be quite expensive as direct interactions are recomputed every time a matrix-vector product is needed. A faster approach is to eliminate most of the inter-processor communication by storing copies of panel charges for remote-processor neighbors, though this second approach requires somewhat more memory. For the experiments in this paper, the direct interactions of a cube are computed by using the faster approach based on storing copies.

C. Projection of Panel Charges onto a Grid

The charges in a cube can be projected onto local representations of the grid, but some inter-processor communication is required to complete the global grid representation because of the decomposition into z planes. To illustrate the problem, consider again Figure 3 where the problem is decomposed between two processors. The line of grid points (planes in 3-D), identified as $G1, G2$ and $G3$, and denoted as interface points, are shared by cubes 2 and 3 where cubes 2 and 3 belong to processors 1 and 2 respectively. Grid points $G1, G2$ and $G3$ are assigned to processor 2 to balance the FFT computation and these grid points are not known to processor 1. However, processor 1 stores an extra line (or plane) to maintain information about the interface points. Denoting q^1_{G1} and q^2_{G1} to be the projected charges at grid point $G1$ in processors 1 and 2, respectively, the global value for the charge at grid point $G1$ is $q_{G1} = q^1_{G1} + q^2_{G1}$ To obtain the global values, each processor i (except the last processor) sends the extra plane of data it stores for the interface points to processor $i+1$. Processor $i+1$ receives and adds the data to its local data to obtain the global values for the interface plane. Processor i, at this stage, does not need the global values for the extra plane as the interface points are not involved in the convolution operation in processor i.

D. Convolution

The three-dimensional convolution to compute grid potentials involves a forward 3-D FFT computation of the kernel and the grid point charges, pointwise multiplication of the kernel and the grid point charges in the Fourier domain, and 3-D inverse Fourier transform of the pointwise multiplied data. The kernel is a fixed set of data and is fourier transformed and stored. The grid point charges, however, change during each iteration of the GMRES algorithm and must be Fourier transformed. The three-dimensional grid charge and kernel data is distributed across the processors along one of the dimensions. The FFT computations along the two dimensions which are local to a processor do not require inter-processor communication. For an FFT along the third dimension data must be moved across the processors. A high level description of the convolution algorithm is given below:

```
for every processor iproc, 0 ≤ iproc < nproc do
    for k = 1 to nzp/2 do /*half the size because of zero pad*/
        for [j = 1:ny/2] fft1d(nx); /*FFT in the first dimension*/
        for [j = 1:nx] fft1d(ny); /*FFT in the second dimension*/
    end for
    transpose();
    for i = 1 to ny do
        for j = 1 to nxp do
            fft1d(nz); /*FFT in the third dimension*/
            multiplykernel(nz); /*pointwise multiply with kernel*/
            ifft1d(nz); /*Inverse FFT in the third dimension*/
        end for
    end for
    transpose();
    for k = 1 to nzp/2 do
        for [j = 1:nx] ifft1d(ny); /*Inverse FFT in the 2nd dim*/
        for [j = 1:ny/2] ifft1d(nx); /*Inverse FFT in the 1st dim*/
    end for
end for
```

In the above description, nx, ny, nz are the zero-padded sizes along the first, second and third dimensions, respectively (see Figure 4); $nxp = \frac{nx}{nproc}$, $nyp = \frac{ny}{nproc}$ and $nzp = \frac{nz}{nproc}$; fft1d() and ifft1d() are the 1-D forward FFT and inverse FFT respectively [6], [7]; multiplykernel() performs a pointwise multiplication of the grid charges and the kernel in the Fourier domain and transpose() is a global operation which requires inter-processor communication. To understand the transpose operation, consider (see Figure 4) a four-way partitioning of the data. (i,i) denotes the data on processor i which belongs to processor i and (i,j) denotes the data on processor i which should be transferred to processor j to compute the FFT in the third dimension. The data is transposed back after a forward FFT in the third dimension, pointwise multiplication of the kernel and the charge data, and inverse FFT in the third dimension are done.

E. Projection of Grid Potentials onto Panels

Once the convolution is completed, the potential at the grid points is available. Then, the first plane of grid points in each processor i (except processor 0) is sent to processor $(i-1)$. Processor $(i-1)$ overwrites the extra plane it stores with the received potential data. The extra plane in processor $(i-1)$ stores the data for the first plane of grid points in processor i. The panel potentials in each processor are then computed locally with no inter-processor communication.

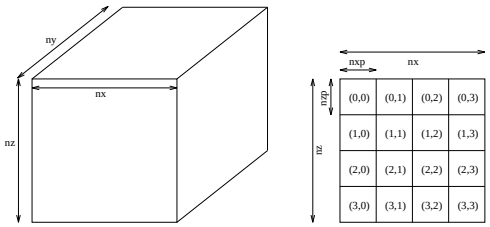
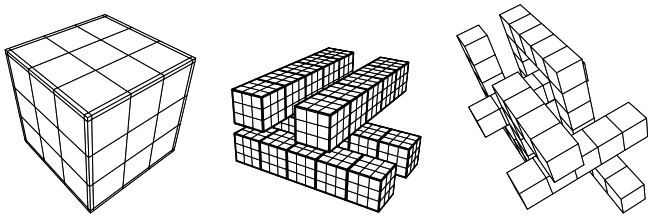


Fig. 4. Definition of the cube dimensions and the transpose operation

IV. Results

Parallel performance results are presented on an IBM SP2 for the test structures shown in Figure 5. The IBM SP2 is a scalable parallel system containing upto 12 RISC/6000 Model 590 nodes. Each node can be configured upto 128 MB memory and upto 2 GB hard disk. The nodes on the SP2 are interconnected through a high performance switch. The unidirectional communication bandwidth available at a node is approximately 40 MB/second.



(a) (b) (c)

Fig. 5. Test structures (a) Cubic capacitor (b) 2×2 Bus crossing problem (c) 2×2 Woven bus problem

TABLE I
TOTAL CPU(SEC) FOR CUBIC CAPACITOR ON 1, 2, 4, 8 PROCESSORS

# panels	2400	5400	9600	15000	21600
1 proc	16.69	63.91	91.88	127.19	176.03
2 procs	10.27	36.73	52.46	71.93	98.01
4 procs	7.27	22.20	32.86	46.14	63.80
8 procs	7.25	15.53	23.82	34.27	50.98

TABLE II
TOTAL CPU(SEC) FOR BUS CROSSING ON 1, 2, 4, 8 PROCESSORS

# panels	2200	4312	35200	79200
1 proc	24.12	85.29	*	*
2 procs	19.29	54.35	*	*
4 procs	12.48	30.77	330.98	*
8 procs	9.19	20.54	178.44	380.63

* indicates insufficient memory

Tables I, II and III summarize the results for the cubic capacitor, bus crossing and woven bus structures, respectively, employing upto eight processors. It is observed that on one processor the grid convolution algorithm takes between 50-70% of the total CPU time for larger problems. When the computation of the convolution is significant, good speedups and parallel efficiencies are obtained as expected. Typical results include a speedup of about 5 on 8 processors and a parallel efficiency of about 60%. The speedup on two processors is about 1.8 and on four processors it is about 3. Larger problems(e.g. the 35200 panel bus crossing problem) achieve a nearly linear speedup of eight. However, when the convolution algorithm is only about 30% of the total CPU time the parallel efficiencies are about 40% as is to be expected. The parallel efficiencies can be further improved by load balancing the direct interactions and the number of charges.

TABLE III
TOTAL CPU(SEC) FOR WOVEN BUS ON 1, 2, 4, 8 PROCESSORS

# panels	4400	17600	39600
1 proc	98.13	216.95	*
2 procs	63.59	165.93	*
4 procs	35.49	91.57	365.43
8 procs	22.87	56.81	202.17

The bus crossing problem with 35200 panels could not be fit on one and two processors as the program could allocate only about 100 MB of memory per processor. When the program is compiled to access memory from hard disk, the 35200 panel example took 2676.8 and 619.2 seconds on 1 and 2 processors, respectively. These results are not reported in Table II as they would suggest superlinear speedups, which are unrealistic and an artifact of the large latencies associated with accessing data from hard disk.

V. Conclusion and Acknowledgements

In this paper we have presented a prototype implementation of the FastCap program on an eight processor IBM SP2. The results indicate that signal integrity analysis can be performed quickly and efficiently on network-of-workstation parallel computers. Analysis of complicated and large 3-D interconnect structures, which cannot be performed on present day serial computers, can now be performed with Parallel FastCap. Parallel FastCap is portable to most parallel computers as it is implemented using MPI standard. The approach presented in this paper involved minimal changes to serial code and we have reported good efficiencies. For higher parallel efficiencies we are presently studying more efficient task decomposition strategies.

The authors would like to thank J. R. Phillips and K. Nabors for providing the Fastcap and precorrected FFT software on which these experiments were based.

References

[1] J. R. Phillips and J. White, "A Precorrected-FFT method for capacitance extraction of complicated 3-D structures," *Proc. of the Int. Conf. on CAD*, Santa Clara, CA, Nov., 1994.

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

[3] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems," *SIAM J. on Scient. and Stat. Comp.*, vol. 7, pp. 856-869, 1986.

[4] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*. Cambridge, Massachusetts: M.I.T. Press, 1988.

[5] K. Nabors, S. Kim, and J. White, "Fast capacitance extraction of general three-dimensional structure," *IEEE Trans. on Microwave Theory and Tech.*, vol. 40, pp. 1496-1507, 1992.

[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C*. Cambridge University Press, 1992.

[7] R. C. Agarwal, F. G. Gustavson and M. Zubair, "An efficient parallel algorithm for the 3-D FFT NAS parallel benchmark," *Proc. of Scalable High Perf. Comp. Conf.*, pp. 129-133, 1994.