Parallel Algorithms for Inductance Extraction of VLSI Circuits[*]

Hemant Mahawar[1†] and Vivek Sarin[1]

[1]Texas A&M University
Department of Computer Science
College Station, TX 77843-3112 USA
{mahawarh,sarin}@cs.tamu.edu

Abstract

Inductance extraction involves estimating the mutual inductance in a VLSI circuit. Due to increasing clock speed and diminishing feature sizes of modern VLSI circuits, the effects of inductance are increasingly felt during the testing and verification stages. Hence, there is a need for fast and accurate inductance extraction software. A generalized approach for inductance extraction requires the solution of a dense complex symmetric linear system that models mutual inductive effects among circuit elements. Iterative methods are used to solve the system without explicit computation of the matrix itself. Fast hierarchical techniques are used to compute approximate matrix-vector products with the dense system matrix. This work presents an overview of a new parallel software package for inductance extraction of large VLSI circuits. The technique uses a combination of the solenoidal basis method and effective preconditioning schemes to solve the linear system. Fast Multipole Method (FMM) is used to compute approximate matrix-vector products with the inductance matrix. By formulating the preconditioner as a dense matrix similar to the coefficient matrix, we are able to use FMM for the preconditioning step as well. A two-tier parallelization scheme allows an efficient parallel implementation using both OpenMP and MPI directives simultaneously. The experiments conducted on various multiprocessor machines demonstrate the portability and parallel performance of the software.

1. Introduction

With more than million of interconnects tightly packed in the next-generation VLSI circuits, signal delays play an important role in the testing and verification stage of VLSI circuit design. These delays are due to the presence of parasitic resistance, capacitance and inductance. With clock frequency reaching giga-hertz (GHz) range and shrinking feature sizes, these delays will be dominated by inductance. Hence, there is a significant need for fast and accurate inductance extraction software.

Inductance refers to the property of an electrical circuit to oppose any change in its current. When current flows through a conductor, it creates a magnetic field around the conductor. This induced magnetic field affects the current flow in the conductor and its neighbors. The effect on the neighboring conductors is measured through mutual inductance, whereas the change in self current is measured through self inductance. In an electrical circuit with hundreds of thousands of interconnect segments, the inductive effects among the segments define the characteristics of the circuit. Inductance extraction process refers to the estimation of the inductance of the entire circuit.

A general approach for inductance extraction involves discretization of the conductor surface using a two dimensional mesh of filaments [5]. Kirchoff's laws governing current and voltage in such a mesh give rise to a set of linear equations that are solved using iterative methods. The linear system consists of both sparse and dense sub-matrices. Kirchoff's current law at the mesh nodes constitutes the sparse sub-matrix, whereas the potential drop constraints across the mesh filaments give rise to the dense sub-matrix. To avoid the memory and computational penalties of exact matrix-vector product, fast hierarchical schemes are often used to compute approximate matrix-vector products with the system matrix. These approaches have a trade-off between accuracy and speed. To accelerate the convergence of iterative methods, preconditioning schemes are used. The

[*]This work has been supported in part by NSF under the grants NSF-CCR 9984400 and NSF-CCR 0113668. The AMD Linux cluster at Texas A&M University was acquired through the MRI grant NSF-DMS 0216275. Access to the IBM p690 was provided by the NCSA, University of Illinois at Urbana-Champaign.

[†]Corresponding Author: mahawarh@cs.tamu.edu

preconditioning step transforms the original system into an "easier" one for the iterative methods.

This work presents a parallel inductance extraction software for fast and accurate estimation of circuit inductance. We use solenoidal basis functions [9] to represent the filament currents in terms of circular cell currents. This approach converts the linear system into a reduced system with fewer unknowns to solve for. Further, we employ a preconditioning scheme that reduces the number of iterations required for the solution. We rely on the characteristics of the system matrix to devise the preconditioner. These steps reduce both memory and time requirements as compared to existing software [9]. To further reduce the solution time and to handle large problem instances, we have developed a parallel implementation [6, 7]. A two-tier parallelization scheme enables mixed mode parallelization, which uses both OpenMP and MPI directives. Mixed mode parallelization allows the software to run on shared, distributed and distributed-shared memory machines. To the best of our knowledge, this is the first parallel software for inductance extraction that can run on a variety of multiprocessors.

The next section presents the mathematical formulation of the problem. Section 3 describes the solenoidal basis functions and the transformation steps to convert the system matrix into a reduced system. Section 4 outlines the preconditioning scheme and presents experimental results to show the effectiveness of the preconditioner. Parallel implementation details are presented in Section 5. Section 6 ends the paper with concluding remarks.

2. Mathematical Background

For a z conductor geometry, inductance extraction refers to computing a $z \times z$ impedance matrix that represents the inductive effect among the conductors. The kth column of the matrix represents the inductive effect on all the conductors when a unit current flows through the kth conductor. Under such conditions, the inductive effect on any conductor is equal to the potential drop across it. Solution of z such instances yields the entire impedance matrix.

The current density $\mathbf{J}$, at any point $\mathbf{r}$, is related to potential ϕ by the following integral equation [5]

$$\rho\mathbf{J}(\mathbf{r}) + j\omega \int_V \frac{\mu}{4\pi} \frac{\mathbf{J}(\mathbf{r}')}{\|\mathbf{r}-\mathbf{r}'\|} dV' = -\nabla\phi(\mathbf{r}), \qquad (1)$$

where μ is the magnetic permeability of the material, ρ is the resistivity, ω is the angular frequency, $\|\mathbf{r}-\mathbf{r}'\|$ is the Euclidean distance between $\mathbf{r}$ and $\mathbf{r}'$, and $j = \sqrt{-1}$. The volume of the conductor is denoted by V and incremental volume with respect to $\mathbf{r}'$ is denoted by dV'. Eq. (1) arises from Maxwell's equations that defines the basic laws of electrodynamics.

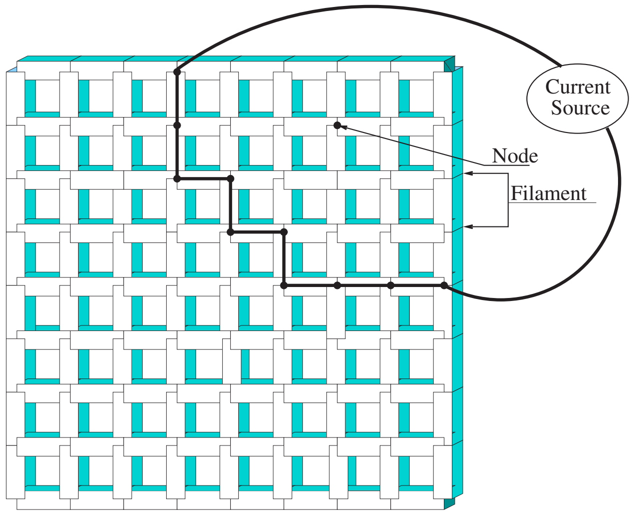

Figure 1. Discretization of a ground plane with a mesh of filaments. Current flow along an arbitrary path in the mesh can be used to satisfy the constraints imposed by the external current source. (Reproduced from [9].)

To obtain the numerical solution of (1), each conductor is discretized into n filaments, $f_1, f_2 \ldots, f_n$ (see, e.g., Fig 1). Assuming current flows only along the length of the filaments, and has constant current density, the following linear system is obtained for the discretized conductor

$$[\mathbf{R} + j\omega\mathbf{L}]\mathbf{I}_f = \mathbf{V}_f, \qquad (2)$$

where $\mathbf{R}$ is an $n \times n$ diagonal matrix of filament resistances, $\mathbf{L}$ is an $n \times n$ dense inductance matrix that relates the inductive effects among the filaments, $\mathbf{I}_f$ is the vector of filament currents, and $\mathbf{V}_f$ is the vector of potential difference across the ends of each filament. The kth diagonal element of $\mathbf{R}$ is given by $\mathbf{R}_{kk} = \rho l_k / a_k$, where l_k is the length of the kth filament f_k, and a_k is the cross-sectional area. Let $\mathbf{u_k}$ denote the unit vector along the kth filament f_k. The elements of $\mathbf{L}$ are given by

$$\mathbf{L}_{kl} = \frac{\mu}{4\pi} \frac{1}{a_k a_l} \int_{r_k \in f_k} \int_{r_l \in f_l} \frac{\mathbf{u}_k \cdot \mathbf{u}_l}{\|\mathbf{r}_k - \mathbf{r}_l\|} dV_k dV_l.$$

Kirchoff's current law at the nodes give rise to the following equation

$$\mathbf{B}^T \mathbf{I}_f = \mathbf{I}_s, \qquad (3)$$

where $\mathbf{B}^T$ is $m \times n$ branch index matrix of nodes and filaments, and $\mathbf{I}_s$ is the known branch current vector of length m with non-zero values corresponding to the source currents. The (k, l) entry of $\mathbf{B}^T$ has the value -1 if the lth filament originates at node k, 1 if it terminates at k, and zero otherwise.

Using Eq. (2) and (3), and the fact that the unknown filament potential drop $\mathbf{V}_f$ can be represented in terms of the node potential $\mathbf{V}_n$ by $\mathbf{B}\mathbf{V}_n = \mathbf{V}_f$, the following linear system is obtained

$$\begin{bmatrix} \mathbf{R} + j\omega\mathbf{L} & -\mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{I}_f \\ \mathbf{V}_n \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{I}_s \end{bmatrix}. \tag{4}$$

This system must be solved to determine the unknown filament current $\mathbf{I}_f$ and node potential $\mathbf{V}_n$. The solution of large linear systems like (4) is not feasible through direct methods. These systems are often solved using iterative methods, such as the generalized minimal residual (GMRES) method [10], to avoid the penalties of the direct method. Further, the use of fast hierarchical methods to compute approximate matrix-vector products reduces the solution time.

3 Solenoidal Basis Method

Solenoidal functions are divergence-free basis functions that automatically satisfies the conservation laws, such as Kirchoff's current law in electrical circuits and mass conservation law in fluid mechanics. These functions have been applied to a variety of engineering applications such as CFD [11]. For the inductance extraction problem with uniform discretization it is easy to construct such a solenoidal basis. Figure 2 shows several instances of discrete solenoidal current flows that can be used to construct a solenoidal basis. Each such flow consists of constant current flowing anticlockwise through the four filaments of a cell in the mesh. Since the net flow of current into any node is zero, these flows automatically satisfy Kirchoff's current law.

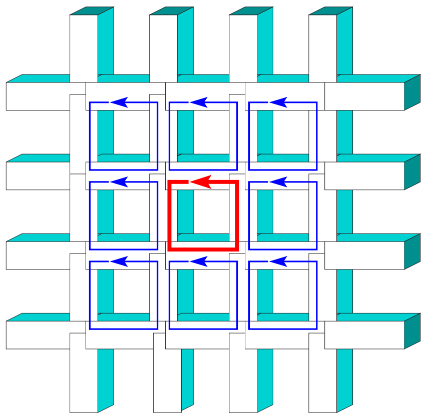

Figure 2. Examples of solenoidal current flows in a section of a uniform two-dimensional mesh. (Reproduced from [9].)

The solenoidal basis matrix $\mathbf{P}$ is an $n \times s$ matrix that is derived from the current flows in the mesh cells. The columns of $\mathbf{P}$ correspond to the cells in the mesh. Each column of consists of four non-zero entries that denote the current flow in a cell: 1 indicates a unit current flow along the edge, and -1 indicates a unit current flow opposite to the direction of edge. Construction of the solenoidal basis matrix in this manner ensures that the following condition is satisfied

$$\mathbf{B}^T\mathbf{P} = 0. \tag{5}$$

Before one can use the solenoidal basis method, the linear system (4) needs to be transformed. We determine a particular current vector $\mathbf{I}_p$ that satisfies the constraints imposed by the external current source. The vector $\mathbf{I}_p$ represents current flow along an arbitrary path between the nodes where the external source is connected (see, e.g., Fig. 1). By splitting the filament current $\mathbf{I}_f$ into a particular current $\mathbf{I}_p$ and an unknown current $\mathbf{I}$, the linear system (4) can be transformed to an equivalent system with a different right hand side

$$\begin{bmatrix} \mathbf{R} + j\omega\mathbf{L} & -\mathbf{B} \\ \mathbf{B}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ \mathbf{V}_n \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}, \tag{6}$$

where

$$\mathbf{I} = \mathbf{I}_f - \mathbf{I}_p, \qquad \mathbf{F} = -\left[\mathbf{R} + j\omega\mathbf{L}\right]\mathbf{I}_p.$$

The difference between (4) and (6) is that the first system satisfies current boundary conditions whereas the second system satisfies voltage boundary conditions.

Using the circular cell currents, we can express the unknown current $\mathbf{I}$ as: $\mathbf{I} = \mathbf{P}x$. Here x is a vector of unknown cell currents, and $\mathbf{P}$ is the solenoidal basis matrix that represents filament currents in terms of circular cell currents. From (5), it follows that $\mathbf{I}$ satisfies the constraints imposed by the second block of equations in (6). The unknown vector $\mathbf{V}_n$ can be eliminated by multiplying the system with $\mathbf{P}^T$ from the left. The linear system (6) can be transformed to the following *reduced* linear system of order s that must be solved to determine x

$$\mathbf{P}^T\left[\mathbf{R} + j\omega\mathbf{L}\right]\mathbf{P}x = \mathbf{P}^T\mathbf{F}. \tag{7}$$

By transforming the linear system (6) to the reduced system (7), we can reduce the number of unknowns considerably. Table 1 shows the number of unknowns in a ground plane problem that involves computing the self impedance of a square conductor.

The use of a local solenoidal basis results in a sparse matrix $\mathbf{P}$ that is amenable to efficient matrix-vector product computations. Matrix-free implementations are also possible since explicit construction of $\mathbf{P}$ is not necessary. In addition, the sparsity of $\mathbf{P}$ can improve the efficiency of parallel implementations.

Table 1. Comparison of the sizes of the original and reduced systems for ground plane problem.

Mesh Size	Nodes (m)	Filaments (n)	Unknowns $(n+m)$	Solenoidal Functions (s)	Size Ratio $(s/(n+m))$
32×32	1089	2112	3201	1024	32.0%
64×64	4225	8320	12545	4096	32.6%
128×128	16641	33024	49665	16384	33.0%
256×256	66049	131584	197633	65536	33.1%

4. Preconditioning Technique

Even for the reduced system (7) the use of direct methods to compute the unknown cell current becomes prohibitively expensive for modest sized problems. Direct methods suffer from high computational costs and large memory requirements. To overcome these hurdles, iterative methods are used. The most computationally expensive step of iterative methods involve computing matrix-vector products with the coefficient matrix. This cost can be reduced by using *matrix free* hierarchical methods, such as FMM.

The convergence of iterative methods is related to the spectral properties of the system matrix. For instance, a large separation between the smallest and largest eigenvalues of a matrix often results in a large number of iterations required for convergence. Preconditioning is a process of transforming a linear system into one that has more favorable spectral properties. The linear system $\mathbf{A}x = b$ may be preconditioned by a matrix $\mathcal{M}$ as shown below

$$\mathbf{A}\mathcal{M}^{-1}y = b, \qquad x = \mathcal{M}^{-1}y.$$

To solve the transformed system iteratively, each iteration incurs an additional cost of matrix-vector product with $\mathcal{M}^{-1}$. A preconditioning approach is advantageous only if the overall time to compute the solution is reduced. The preconditioner must be easy to compute, the preconditioning step must be relatively inexpensive, and the matrix $\mathcal{M}$ should be an effective preconditioner that reduces the number of iterations considerably. A preconditioning approach is considered optimal if the number of iterations required are independent of the discretization parameters.

For inductance extraction problem, we derive a preconditioner by analyzing the reduced system (7). The matrix $\mathbf{P}$ allows transformation of circular mesh currents to filament currents, and is equivalent to a discrete curl operator. Furthermore, the matrix $\mathbf{P}^T\mathbf{P}$ is equivalent to a discrete Laplace operator. Using these characteristics, we propose the following preconditioner for the reduced system (7) [9]

$$\mathcal{M}^{-1} = \tilde{\mathbf{L}}\left[\tilde{\mathbf{R}} + j\omega\tilde{\mathbf{L}}\right]^{-1}\tilde{\mathbf{L}}, \qquad (8)$$

where $\tilde{\mathbf{R}}$ is a diagonal matrix of resistance to mesh currents. The $\tilde{\mathbf{L}}_{kl}$ entry gives the mutual inductance between parallel filaments placed at the centers of loop k and l, and is defined as follows

$$\tilde{\mathbf{L}}_{kl} = \frac{\mu}{4\pi}\frac{1}{a_k a_l}\int_{r_k \in V_k}\int_{r_l \in V_l}\frac{1}{\|\mathbf{r}_k - \mathbf{r}_l\|}dV_k dV_l.$$

At low and high frequencies, one can use the following approximations to the preconditioner without any significant change in the rate of convergence

$$\mathcal{M}^{-1}_{\text{low}} = \tilde{\mathbf{L}}\tilde{\mathbf{R}}^{-1}\tilde{\mathbf{L}}, \qquad \mathcal{M}^{-1}_{\text{high}} = -j\omega^{-1}\tilde{\mathbf{L}}.$$

In each case, the preconditioning step is relatively cheap since it does not involve an inner solve. For intermediate frequencies, however, one must use the preconditioner in (8).

As shown in Table 2, the number of iterations required by the preconditioned GMRES algorithm to solve the linear system (7) is almost constant when either the mesh width h or the angular frequency ω is changed. Table 2 shows the effectiveness of the preconditioner for the *ground plane* problem (see Fig. 1). The ground plane is used to provide a uniform ground potential to all the components of a VLSI circuit. The problem requires computing the self-impedance of a 1cm $\times$ 1cm ground plane. A uniform two-dimensional mesh has been used to discretize the ground plane. A tolerance $\tau = 10^{-3}$ has been used as a stopping criteria for the iterative method.

The proposed preconditioning approach has several advantages over incomplete factorization based preconditioners. Our preconditioning step requires a matrix-vector product that is relatively inexpensive compared to incomplete factorization based preconditioners. The latter involve incomplete factorizations of a partially computed coefficient matrix and triangular solves that are expensive, especially on parallel platforms. In addition, experimental evidence suggests that unlike incomplete factorization, our preconditioner is robust and very effective over a wide range of frequencies.

The second benchmark problem is the *cross-over* problem shown in Fig. 3. This setup represents a cross-over of interconnect segments. The problem consists of determining the impedance matrix for these segments. The segments are 2cm long and 2mm wide, and are separated by 300μm in the horizontal direction and by 3mm in vertical direc-

Table 2. Number of iterations required for convergence of preconditioned GMRES method to compute the self-impedance of a ground plane.

Mesh Size	Filament Length(cm)	Frequency			
		100 MHz	1GHz	10 GHz	100 GHz
32×32	1/32	5	5	5	5
64×64	1/64	7	5	5	5
128×128	1/128	9	6	6	6
256×256	1/256	12	6	6	6
512×512	1/512	14	7	6	6

tion. The discretization is similar to the ground plane problem. These simulations were conducted for a frequency of 10GHz. Table 3 reports the number of iterations required by the preconditioned GMRES method to compute the complete impedance matrix($\tau = 10^{-3}$). The growth in number of iterations is minimal as the number of conductors in the configuration is increased. Similarly, the growth in iterations is slow as the mesh is refined. These results illustrate the effectiveness of the preconditioning scheme for typical extraction problems.

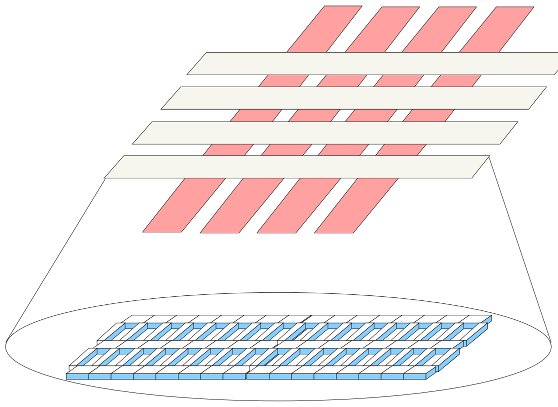
Figure 3. Cross-over problem with a view of a discretized conductor.

Table 3. Number of iterations required for convergence of preconditioned GMRES method for the cross-over problem.

Mesh Size	Filament Length (cm)	Conductor Layout		
		1+1	2+2	4+4
16×160	1/80	6	7	8
32×320	1/160	7	8	9
64×640	1/320	8	9-10	11
128×1280	1/640	8	9-10	11

The cost of the orthogonalization step in GMRES is proportional to k^2, where k is the number of iterations. Hence, the parallel performance of GMRES degrades as k increases due to the increased communication overhead of orthogonalization step. By using an effective preconditioner that requires very few iterations, we reduce the computational cost as well as the storage requirement of GMRES. Furthermore, the parallel implementation does not suffer from the effects of orthogonalization step.

5. Parallel Matrix Vector Product

Both the preconditioner matrix $\tilde{\mathbf{L}}$ and the inductance matrix $\mathbf{L}$ are dense matrices with similar structures. Since each step of the iterative method requires matrix-vector products with $\mathbf{L}$ and $\tilde{\mathbf{L}}$, it is worthwhile to reduce the computational cost of these operations. For a dense $n \times n$ matrix, the computational cost of a straightforward matrix-vector product is $O(n^2)$. The entries of $\mathbf{L}$ and $\tilde{\mathbf{L}}$ have a $1/r$ decaying kernel, which makes them suitable candidates for the fast hierarchical methods. A number of such techniques have been developed, including the well known Appel's algorithm [1], the Barnes-Hut [2] method, and Fast Multipole Method (FMM) [4]. Parallel formulations of multipole-based techniques have been developed by several groups [3, 12, 13, 14].

These methods compute approximate matrix-vector products in $O(n \log n)$ or $O(n)$ steps. The reduction in computational complexity is at the expense of accuracy. These methods provide a *matrix-free* way to compute the matrix-vector product in which the system matrix is never stored. This approach reduces the memory requirements of the solver. The Barnes-Hut method relies on particle-cluster interaction to achieve $O(n \log n)$ computational complexity, whereas FMM additionally uses cluster-cluster interactions to obtain a complexity of $O(n)$. The particles are clustered into groups that form the nodes of an oct-tree. In Barnes-Hut method, one calculates the "*center of mass*" at the internal nodes in a bottom-up fashion. The center of mass at each node approximates the effect of all the particles in its subtree. To compute the effect due to the node's particles at an observation point, a top-down traversal is done to identify nodes that satisfy the multipole acceptance criteria. The acceptance criteria requires that the ratio of the size of the node to the distance of the observation point from the node be below a threshold value. To get the effect of the particles in a node's subtree, one has to use the center of mass of the node. More details on it can be found in [2].

In FMM, multipole coefficients are calculated in a bottom-up fashion, whereas local coefficients are computed in a top-down fashion. The multipole coefficients give the effect of the particles belonging to the node's subtree at an observation point far-away from the node. On the other hand, local coefficients give the effect of far-away particle clusters at an observation point inside the nodes. The effect of "near-by" particles is computed directly in both Barnes-Hut and FMM. We have also developed a hierarchical multipole method (HMM) [8]. HMM can be treated as an augmented Barnes-Hut or modified FMM. It has the implementation ease of Barnes-Hut and accuracy of FMM. HMM relies only on particle-cluster interactions to achieve an $O(n \log n)$ computational bound. In this work, we use a variant of FMM algorithm to compute approximate matrix-vector products with both $\mathbf{L}$ and $\tilde{\mathbf{L}}$. Filament mid-points form the set of particles for FMM.

We have developed an object oriented inductance extraction software package. This software combines the advantages of the solenoidal basis method, fast hierarchical methods for dense matrix-vector products, and highly effective preconditioning schemes to provide a powerful package for inductance extraction. In addition, the software includes an efficient parallel implementation that reduces the overall computation time on a variety of parallel architectures [6].

We employ a two-tier parallelization scheme, as shown in Fig. 4, which provides software portability. Each conductor is assigned to a different processor to exploit conductor level parallelism. All data structures that are native to a conductor are local to its processor. This includes filaments in a conductor and the associated FMM tree. Only matrix-vector products incur communication cost as they involve interactions among different conductors that are distributed across processors.

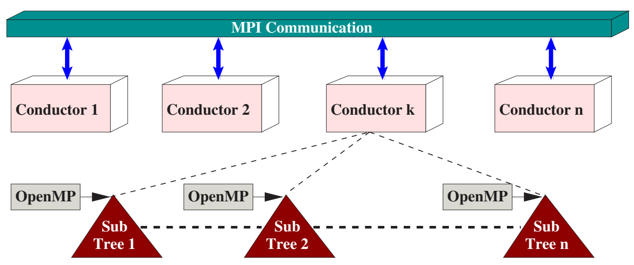

Figure 4. Two-tier parallelization scheme.

The matrix-vector products with $\mathbf{L}$ and $\tilde{\mathbf{L}}$ involve interactions among filaments of the same conductor as well as between the filaments of different conductors. To calculate the interactions between filaments of the same conductor, only local computation is required. Communication is required to get the effect of filaments in other conductors. Communication is also needed to compute direct interaction among "near-by" nodes that reside on different processors. During a pre-processing step, the software identifies those nodes in a conductor's tree that are required by other conductors. This cost is amortized over the iterations of the solver. This type of communication is proportional to the number of filaments on the subdomain boundary.

To exploit parallelism within each conductor, threads are assigned to nodes at a specific level in the FMM tree (see Fig. 4). Fewer processes can be assigned to the top part of the FMM tree to further improve parallel efficiency. With different sized conductors, one can have more processes associated with larger conductors. This scheme allows load balancing to a certain extent. Further details on two-tier parallelism can be found in [7].

5.1. Parallel Performance

The two-tier parallelization approach provides the dual advantage of portability and performance across a variety of platforms. MPI processes are used for conductor level parallelism, while OpenMP directives are used to exploit parallelism within a conductor. We present experiments to demonstrate the parallel performance of the software on multiprocessors with shared, distributed, and distributed-shared memory architectures. We consider distributed memory platforms such as the 64-bit AMD Linux cluster where parallelism can be exploited via MPI processes only. Distributed-shared memory platforms such as the IBM p690 that allow mixed mode parallelization with both MPI and OpenMP are also considered.

The performance of the software is measured by comparing the processor utilization of the code on a set of benchmark problems. We define *computational rate* (CR) on a processor as the number of *base operations* (BOP) executed per second. A base operation involves computing an interaction between a pair of distinct filaments. The computational rate of the code on a multiprocessor depends on the number of processors (p), the total execution time (T), and the number of base operations.

$$CR(p, BOP, T) = \frac{BOP}{p \times T}$$

In experiments involving variable number of MPI processes (P_{MPI}) and OpenMP processes (P_{OMP}), a generalized notion of parallel efficiency is used to provide a uniform basis to evaluate the code's performance. We define

Table 4. Parallel performance for the ground plane problem on IBM p690 using OpenMP (d=4, s=32).

No. of processors	Mesh Size					
	128×128		256×256		512×512	
	Time (s)	%Eff.	Time (s)	%Eff.	Time (s)	%Eff.
1	60.4	100	259.7	100	1062.6	100
2	31.3	97	132.5	98	546.0	97
4	15.6	97	66.7	97	275.7	96
8	8.9	85	36.7	88	147.6	90
16	5.5	68	22.3	73	92.6	72

Table 5. Parallel performance for the cross-over problem on IBM p690 using MPI and OpenMP (conductor mesh size=128×1280, d=4, s=32).

	$P_{MPI}=1$ 4195 MBOP[1]		$P_{MPI}=2$ 16798 MBOP[1]		$P_{MPI}=4$ 68316 MBOP[1]		$P_{MPI}=8$ 276077 MBOP[1]	
P_{OMP}	Time (s)	RCR	Time (s)	RCR	Time (s)	RCR	Time (s)	RCR
1	457	1.0	923	0.99	1903	0.98	3868	0.97
2	238	0.96	477	0.96	970	0.96	1957	0.96
4	118	0.96	249	0.92	498	0.93		
8	63	0.90	140	0.82				

relative computational rate (RCR) as the computational rate achieved by the code relative to its computational rate on a single processor.

$$RCR = \frac{CR(p, BOP, T)}{CR(1, BOP, T)} \tag{9}$$

Ideally, RCR should remain unchanged when the number of conductors and the filaments per conductor are varied. With RCR as the metric, it is possible to compare the performance of the code on different benchmarks that require different number of mutual inductance interactions.

We report the parallel performance of the algorithm for a fixed number of GMRES iterations. This is indicative of the actual performance since the dense matrix-vector products account for over 98% of the execution time. The performance of the software depends on various parameters for FMM, such as the number of particles in leaf nodes (s), the multipole degree (d), etc. One should note that the higher multipole degree significantly increases the compputional cost compared to the communication cost, which in turn improves the parallel effficiency (see [6] for details).

5.2. Shared Memory Parallelization

We present the ground plane problem shown in Fig. 1 to illustrate the parallel performance of the code on a shared memory multiprocessor. These experiments were conducted on a 32-processor IBM p690 multiprocessor with 1.3GHz processor speed and AIX5.1 operating system.

Table 4 shows the execution time and parallel efficiency of the software for linear systems of order 32K, 128K and 512K unknowns. It can be seen that by increasing the problem size, parallel efficiency is maintained when the number of processors are increased. For a fixed size problem, a modest decrease in parallel efficiency with increase in the number of processors indicates an efficient parallel implementation.

5.3. Mixed Mode Parallelization

The benchmark problem presented in this section utilizes the two-tier parallel implementation of the software. The *cross-over* problem shown in Fig. 3 consists of computing the impedance matrix for the overlapping segments. The experiments were conducted on 16 processors of an IBM p690 at NCSA, Illinois. Due to site restrictions the product of the number of OpenMP and MPI processes could not exceed 16. Each MPI process was responsible for one conductor, and OpenMP directives were used to parallelize computation within the conductor. The *cross-over* problem leads to non-uniform point distribution for the dense matrix-vector multiplication algorithm.

Table 5 shows the parallel performance of the software on the *cross-over* problem where each conductor has been discretized by a mesh of size 128×1280. The experiments were setup to compute the full impedance matrix. For the given problem size, the linear system includes 320K un-

[1]MBOP is equal to 10^6 *base operations* (BOP)

Table 6. Parallel performance for the cross-over problem using MPI on IBM p690 and AMD-64 Linux cluster (conductor mesh size=128$\times$1280, d=4, s=32).

	Number	IBM p690			AMD-64 Linux		
P_{MPI}	of RHS	MBOP[1]	Time (s)	RCR	MBOP[1]	Time (s)	RCR
1	1	4323	463	1.0	4365	980	1.0
2	2	17303	938	0.99	17471	2039	0.96
4	4	70050	1921	0.98	70718	4152	0.96
8	8	282294	3868	0.98	284947	8447	0.95
16	16	1135510	7883	0.96	1146072	17120	0.94

Table 7. Parallel performance for the cross-over problem using MPI on AMD-64 Linux cluster (d=4, s=32).

		Conductor Mesh Size					
	Number of	32×320			64×640		
P_{MPI}	RHS	MBOP[1]	Time (s)	RCR	MBOP[1]	Time (s)	RCR
4	4	4827	291	0.94	18072	1066	0.96
8	8	20044	607	0.93	73036	2181	0.94
16	16	83944	1346	0.88	295956	4441	0.94
32	32	364100	2982	0.86	1222530	9268	0.93
64	64	1498175	6525	0.81	4952507	19006	0.92

knowns per conductor. The number of right hand sides required to compute the complete inductance matrix is equal to the number of conductors. We use RCR as defined in (9) to compare the performance of the software for problems of varying sizes. The experiments show that parallelism within a conductor is exploited very effectively via the OpenMP directives.

5.4. Distributed Memory Parallelization

We report the parallel performance of the software on distributed memory multiprocessors. The experiments were conducted on IBM p690 at NCSA and a 64-bit AMD Opteron-240 Tensor cluster at Texas A&M University. The Tensor cluster consists of 1.4GHz 64-bit AMD Opteron processors with SuSE-Linux operating system. PGI compilers were used on the Tensor cluster for compiling the code.

Table 6 shows the execution time and parallel performance of the software for the *cross-over* problem. The parallel implementation uses MPI directives only. Note that the number of conductors is identical to the number of MPI processors. In these cases, the total base operations and the execution time increases with increasing conductors. This is accompanied by a growth in the communication required among the processors. However, the parallel performance defined as the average base operations per second is maintained across problem instances. This indicates that the code utilizes each processor efficiently when the load is distributed uniformly across processes. Table 7 shows the parallel performance of the software on up to 64 processors of Tensor cluster. For the two problem instance with variable conductor discretization, the linear system includes 20K and 80K unknowns per conductor. Experimental results demonstrate that the software is able to maintain high parallel performance, indicating an efficient parallel implementation.

6. Conclusion

This paper presents a high performance parallel software package for inductance extraction of VLSI circuits. The software utilizes solenoidal basis method and an effective preconditioning scheme to deliver a fast and accurate inductance extraction algorithm. The solenoidal basis method transforms the system matrix into a reduced system and the preconditioning scheme reduces the number of iterations of the iterative method. Fast hierarchical methods are used for the computationally intensive matrix-vector products with the dense coefficient and preconditioner matrices. An efficient parallel implementation of the algorithm reduces the overall computation time considerably on multiprocessors. A two-tier parallelization approach involving MPI and OpenMP directives is used to achieve portability and parallel performance. Experimental results demonstrate high parallel efficiency on shared-memory, distributed-memory, and distributed-shared memory multiprocessors.

References

[1] A. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6:85–103, 1985.

[2] J. Barnes and P. Hut. A hierarchical O($n\ log\ n$) force calculation algorithm. *Nature*, 324:446–449, 1986.

[3] A. Grama, V. Kumar, and A. Sameh. Parallel hierarchical solvers and preconditioners for boundary element methods. *SIAM Journal of Scientific Computing*, 20:337–358, 1998.

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

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

[6] H. Mahawar and V. Sarin. Parallel iterative methods for dense linear systems in inductance extraction. *Parallel Computing*, 29:1219–1235, September 2003.

[7] H. Mahawar and V. Sarin. Parallel software for inductance extraction. In *Proceedings of the International Conference on Parallel Processing*, pages 380–386, Quebec, Canada, August 2004.

[8] H. Mahawar, V. Sarin, and A. Grama. Parallel performance of hierarchical multipole algorithms for inductance extraction. In *Proceedings of the* 11^{th} *International Conference on High Performance Computing*, pages 450–461, Banglore, India, December 2004.

[9] H. Mahawar, V. Sarin, and W. Shi. A solenoidal basis method for efficient inductance extraction. In *Proceedings of the* 39^{th} *Conference on Design Automation*, pages 751–756, New Orleans, Louisiana, June 2002.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, 1996.

[11] S. R. Sambavaram and V. Sarin. A parallel solenoidal basis method for incompressible fluid flow problems. In *Proceedings of the Parallel Computational Fluid Dynamics '01*, pages 309–314, North-Holland, Amsterdam, May 2002.

[12] F. Sevilgen, S. Aluru, and N. Futamura. A provably optimal, distribution-independent, parallel fast multipole method. In *Proceedings of the* 14^{th} *IEEE International Parallel and Distributed Processing Symposium*, pages 77–84, Cancun, Mexico, May 2000.

[13] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. L. Hennessy. Load balancing and data locality in hierarchical n-body methods. *Journal of Parallel and Distributed Computing*, 27:118–141, 1995.

[14] S. H. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive n-body simulation. *SIAM Journal of Scientific Computing*, 19:635–656, 1998.