Parallel Software for Inductance Extraction*

Hemant Mahawar† and Vivek Sarin
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112

Abstract

The next generation VLSI circuits will be designed with millions of densely packed interconnect segments on a single chip. Inductive effects between these segments begin to dominate signal delay as the clock frequency is increased. Modern parasitic extraction tools to estimate the on-chip inductive effects with high accuracy have had limited impact due to large computational and storage requirements. This paper describes a parallel software package for inductance extraction called ParIS*, which is capable of analyzing interconnect configurations involving several conductors within reasonable time. The main component of the software is a novel preconditioned iterative method that is used to solve a dense complex linear system of equations. The linear system represents the inductive coupling between filaments that are used to discretize the conductors. A variant of the Fast Multipole Method is used to compute dense matrix-vector products with the coefficient matrix.* ParIS *uses a two-tier parallel formulation that allows mixed mode parallelization using both MPI and OpenMP. An MPI process is associated with each conductor. The computation within a conductor is parallelized using OpenMP. The parallel efficiency and scalability of the software is demonstrated through experiments on the IBM p690 and Intel and AMD Linux clusters. These experiments highlight the portability and efficiency of the software on multiprocessors with shared, distributed, and distributed-shared memory architectures.*

Keywords. Inductance extraction, Parallel computing, Iterative methods, Preconditioning, Mixed mode parallelization.

1 Introduction

The design and testing phases in the development of VLSI chips rely on accurate estimation of the signal delay. Signal delay in a VLSI chip is due to the parasitic resistance (R), capacitance (C), and inductance (L) of the interconnect segments. As a result of newer technology, which uses thicker copper wires, the influence of parasitic resistance has decreased. On the other hand, clock rates in the GHz range have increased the effect of the parasitic inductance on signal delay. Fast and accurate inductance extraction techniques are needed for signal delay estimation in the next-generation microprocessors with millions of interconnect segments.

At high frequencies, the physical proximity of interconnect segments leads to strong inductive coupling between neighboring conductors. This coupling arises due to a magnetic field that is created when current flows through a conductor. This magnetic field opposes any change in the current flow within the conductor as well as in the neighboring conductors. Self-inductance is the resistance offered to change in current within a conductor. Mutual inductance refers to the resistance offered to change in current in a neighboring conductor. Inductance extraction refers to the process of estimating self and mutual inductance between interconnect segments of a chip.

To estimate inductance between a set of conductors in a particular configuration, one needs to determine current in each conductor under appropriate equilibrium conditions. The surface of each conductor is discretized by a uniform two-dimensional grid whose edges represent current-carrying filaments. The potential drop across a filament is due to its own resistance and due to the inductive effect of other filaments. Kirchoff's current law is enforced at the grid nodes. This results in a large dense system of equations that is solved by iterative methods such as the generalized minimum residual method (GMRES) [7]. Each iteration requires a matrix-vector product with the coefficient matrix that can be computed without explicitly forming the matrix itself. Matrix-vector products with the dense ma-

*This work has been supported in part by NSF under the grants NSF-CCR 9984400 and NSF-CCR 0113668, and by the Texas Advanced Technology Program grant 000512-0266-2001. The AMD Linux cluster at Texas A&M University was acquired through the MRI grant NSF-DMS 0216275. Access to the IBM p690 and Intel Linux clusters was provided by the NCSA, University of Illinois at Urbana-Champaign.

†Corresponding author: mahawarh@cs.tamu.edu.

trix are computed approximately via hierarchical methods such as the Fast Multipole Method (FMM) [3]. The task of developing preconditioners, which accelerate the rate of convergence of the iterative method, is complicated by the unavailability of the coefficient matrix.

FastHenry [4] is a commonly available software package that uses the above approach for accurate inductance extraction. Matrix-vector products are computed efficiently by using FMM. Preconditioners are obtained by approximating the dense matrix with a sparse matrix that is derived from the FMM hierarchical structure. Since the process of constructing and applying these preconditioners requires large amount of memory and time, the software has found limited use.

This paper presents an object-oriented parallel inductance extraction software package called *ParIS*. The software uses a different formulation in which the current is restricted to the subspace satisfying Kirchoff's law through the use of solenoidal basis functions. The reduced system of equations is solved by a preconditioned iterative solver that uses FMM to compute products with the dense coefficient matrix and the preconditioner. Improved formulation and the associated preconditioning is responsible for significant reduction in computational and storage requirements.

This paper describes a parallel formulation of the algorithm and the performance of the parallel code on a variety of multiprocessors. Section 2 presents the inductance extraction problem and outlines the solenoidal basis method. Section 3 describes the parallel formulation of the algorithm and implementation of the software. Section 4 presents a set of experiments that show the parallel efficiency and scalability of the software on a variety of architectures. The experiments have been conducted on IBM p690, 64-bit Intel Linux cluster, and 64-bit AMD Linux cluster. Conclusions are presented in Section 5.

2 Background

For a set of s conductors, we need to determine an $s \times s$ impedance matrix that represents pair-wise mutual inductance among the conductors at a given frequency. The element (l, k) of the matrix equals the potential drop across conductor l when there is zero current in all conductors, except conductor k that carries unit current. The kth column is computed by solving an instance of the inductance extraction problem with the right hand side denoting unit current flow through conductor k. The impedance matrix can be computed by solving s instances of this problem with different right hand sides.

The current density $\mathbf{J}$ at a point r is related to potential ϕ by the following equation [4]

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

where μ is magnetic permeability of the material, ρ is the resistivity, r is position vector, ω is 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 r' is denoted by dV'. Equation (1) can be derived from Maxwell's equations that define the fundamental laws of electrodynamics.

To obtain a numerical solution for (1), each conductor is discretized into a mesh of n filaments $f_1, f_2, \ldots, f_n$. Current is assumed to flow along the filament length. The current density within a filament is assumed to be constant. Filament currents are related to the potential drop across the filaments according to the linear system

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

where $\mathbf{R}$ is an $n \times n$ diagonal matrix of filament resistances, $\mathbf{L}$ is a dense inductance matrix denoting the inductive coupling between current carrying filaments, $\mathbf{I}_f$ is the vector of filament currents, and $\mathbf{V}_f$ is the vector of potential difference between 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 and a_k are the length and cross-sectional area of the filament f_k, respectively. Let $\mathbf{u_k}$ denote the unit vector along the kth filament. The elements of the inductance matrix $\mathbf{L}$ 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 states that the net current flow into a mesh node must be zero. These constraints on current lead to additional equations

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

where $\mathbf{B}^T$ is a sparse $m \times n$ branch index matrix and $\mathbf{I}_s$ is the known branch current vector of length m with non-zero values for source currents only. The branch index matrix defines the connectivity among filaments and nodes. The (k, l) entry of the matrix is -1 if filament l originates at node k, 1 if filament l terminates at node k, and 0 otherwise. Since the unknown filament potential drop $\mathbf{V}_f$ can be represented in terms of node potential $\mathbf{V}_n$ by the relation $\mathbf{B}\mathbf{V}_n = \mathbf{V}_f$, one needs to solve the following system of equations to determine the unknown filament current $\mathbf{I}_f$ and node potential $\mathbf{V}_n$

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

For systems involving a large number of filaments, it is not feasible to compute and store the dense matrix $\mathbf{L}$. These linear systems are typically solved by iterative techniques such as GMRES. The matrix-vector products with

Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)
0190-3918/04 $20.00 © 2004 **IEEE**

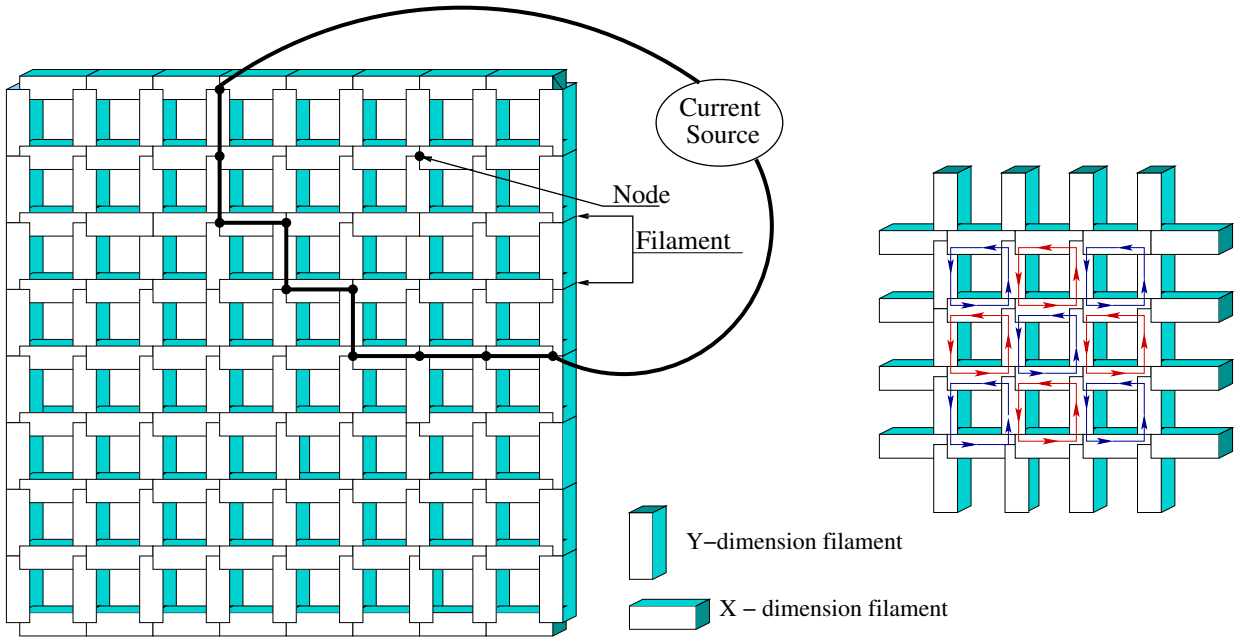

Figure 1. Discretization of a ground plane with a mesh of filaments (left) and solenoidal current flows in each mesh cell (right). (Reproduced from [6].)

$\mathbf{L}$ are computed using fast hierarchical methods such as the FMM. The main hurdle in this matrix-free approach is the construction of effective preconditioners for the coefficient matrix.

2.1 The Solenoidal Basis Method

We present a brief overview of the solenoidal basis method for solving (4) (see, e.g., [6] for details). Consider the discretization of a ground plane in Fig. 1. Current flowing through the filaments must satisfy Kirchoff's law at each node in the mesh. Current can be decomposed into two components: constant current along the bold line shown on the left and a linear combination of mesh currents as shown in the partial mesh on the right. This decomposition converts the system in (4) into the following system:

$$\begin{bmatrix} \mathbf{R}+j\omega\mathbf{L} & -\mathbf{B} \\ \mathbf{B}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ \mathbf{V}_n \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}. \qquad (5)$$

The main difference between matrix representations in (4) and (5) is that the former uses current boundary conditions and the later uses potential boundary conditions.

Solenoidal functions are a set of basis functions that satisfy conservation laws automatically. As shown in the partial mesh in Fig. 1, unit circular flows defined on mesh cells automatically satisfy Kirchoff's law at the grid nodes. The unknown filament currents can be expressed in the solenoidal basis: $\mathbf{I}_f = \mathbf{P}x$, where x is a vector of unknown mesh currents and $\mathbf{P}$ is a sparse matrix whose columns denote filament current in each mesh. A column of $\mathbf{P}$ consists of four non-zero entries that have the value 1 or -1 depending on the direction of current in the filaments of the cell.

The system (5) is converted to a *reduced* system

$$\mathbf{P}^T \left[\mathbf{R}+j\omega\mathbf{L}\right] \mathbf{P}x = \mathbf{P}^T\mathbf{F}, \qquad (6)$$

which is solved by a preconditioned iterative method. The preconditioning step involves product with a dense matrix that represents inductive coupling between filaments placed at the cell centers. This scheme can be implemented using FMM as well, and leads to rapid convergence of the iterative method. On a set of benchmark problems, serial implementation of this software is up to 5 times faster than FastHenry [4], with only one-fifth of memory requirements [6].

3 *ParIS*: Parallel Inductance Extraction Software

We have developed an object oriented parallel implementation of the solenoidal basis algorithm for inductance extraction. This software combines the advantages of the solenoidal basis method, fast hierarchical methods for dense matrix-vector products, and highly effective preconditioning scheme to provide a powerful package for inductance extraction. In addition, the software includes an efficient parallel implementation to reduce overall computation time [5] on parallel architectures.

Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)
0190-3918/04 $20.00 © 2004 **IEEE**

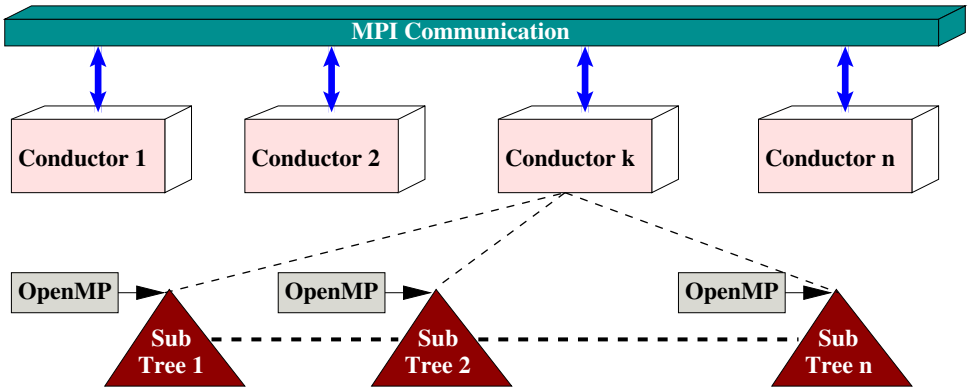

Figure 2. Two-tier parallelization scheme implemented in *ParIS*.

The building blocks of *ParIS* are conductor elements. Each conductor is uniformly discretized with a mesh of filaments. The most time-consuming step in the solution of the reduced system involves matrix-vector products with the impedance matrix. This matrix is the sum of a diagonal matrix $\mathbf{R}$ and a dense inductance matrix $\mathbf{L}$. Since the preconditioning step also involves matrix-vector products with a dense matrix that is similar to $\mathbf{L}$, it is worthwhile to reduce the cost of the matrix-vector product with $\mathbf{L}$.

3.1 Parallel Dense Matrix Vector Product

The computational complexity of a matrix-vector product with a dense $n \times n$ matrix is $O(n^2)$. This complexity can be reduced significantly through the use of hierarchical approximation techniques. These algorithms exploit the decaying nature of the $\frac{1}{r}$ kernel for the matrix entries to compute approximate matrix-vector products with bounded error. Higher accuracy can be achieved at the expense of more computation. For instance, the Barnes-Hut [1] method computes particle-cluster interactions to achieve $O(n \log n)$ complexity, while the Fast Multipole Method (FMM) [3] computes cluster-cluster interactions to achieve $O(n)$ complexity. *ParIS* uses a variant of FMM to compute approximate matrix-vector products.

FMM is used to compute the potential at each filament due to current flowing in all the filaments. The algorithm divides the domain into eight non-overlapping subdomains, and continues the process recursively until each subdomain has at most k filaments, where k is a parameter that is chosen to maximize computational efficiency. A subdomain is represented by a subtree whose leaf nodes contain the filaments in the subdomain. These subdomains are distributed across processors. The potential evaluation phase consists of two traversals of the tree. During the up-traversal, multipole coefficients are computed at each node. These coefficients can be used to compute potential due to all the filaments within the node's subdomain at a *far away* point. These multipole computations do not require any communication between processors. During the down-traversal, local coefficients are computed at each node from the multipole coefficients. The local coefficients can be used to compute potential due to *far away* filaments at a point within the node's subdomain. Potential due to *near by* filaments is computed directly.

One can also use other hierarchical multipole-based approximation techniques instead of FMM to compute the dense matrix-vector products. Parallel formulations of multipole-based techniques including the FMM have been developed by several groups. The reader is referred to [2, 8, 9, 10] for a representative set of approaches.

3.2 Conductor Level Parallelism

To exploit parallelism at the conductor level, each conductor is assigned to a different processor. The data structures native to a conductor are local to its processor. This includes the filaments in a conductor and the associated FMM tree. With the exception of matrix-vector products, all other computations are local to each conductor.

The matrix-vector product with the inductance matrix involves two types of filament interactions. Interactions between the filaments of the same conductor are computed locally by the associated processor. To get the effect of filaments in other conductors, a processor needs to exchange multipole coefficients with other processors. During a preprocessing step, *ParIS* identifies the nodes in a conductor's tree that are required by other conductors. The cost of this step is amortized over the iterations of the solver. While computing the dense matrix-vector product, communication is needed to translate the multipole coefficients of these nodes to the nodes on other processors. Communication is also needed when computing direct interactions between adjacent nodes that belong to different subtrees. This type of communication is proportional to the number of filaments on the subdomain boundary.

Additional parallelism is available within each conductor. By assigning different processes or threads to all the nodes at a specific level in the FMM tree, one can partition the computation for subdomains between processes. 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.

A two-tier approach, as shown in Fig. 2, allows the algorithm to be implemented in hybrid or mixed mode using both MPI and OpenMP. The software can be executed on a variety of platforms ranging from shared-memory multiprocessors to workstation clusters without any change.

4 Experimental Results

The software design of *ParIS* has the dual advantage of portability and performance across a variety of platforms. This is achieved through a two-tier parallelization approach that uses MPI processes for conductor level parallelism and OpenMP directives to exploit parallelism within a conductor. This section presents experiments to demonstrate the parallel performance of *ParIS* on multiprocessors with shared, distributed, and distributed-shared memory architectures. Distributed memory platforms such as the 64-bit Intel and AMD Linux clusters allow parallelism to be exploited via MPI processes only. Distributed-shared memory platforms such as the IBM p690 allow mixed mode parallelization with both MPI and OpenMP.

The performance of the code is measured by the efficiency on a set of benchmark problems. In these experiments, a generalized notion of efficiency is used to provide a uniform basis to compare different experiments. Efficiency is defined as follows:

$$E = \frac{BOPS}{p},$$

where p is the number of processors and $BOPS$ is the average number of *base operations* executed *per second*. A base operation involves computing an interaction between a pair of filaments. In general, $BOPS$ should remain unchanged when the number of conductors and the filaments per conductor are varied. With this definition of efficiency, it is possible to compare the performance of the code on different benchmarks that require unequal number of mutual inductance interactions.

Since the multipole degree, d, influences the accuracy of the approximate dense matrix-vector product, a fair comparison is possible only when the impedance error is bounded. It was seen that for $d = 4$, the impedance error was always within 1% of a reference value that was calculated using FMM with $d = 8$. The parallel performance of the algorithm is reported for a fixed number of GMRES iterations. The results are identical to the case when the full inductance extraction problem is solved because the dense matrix-vector products account for over 98% of the execution time. It may be noted that the use of higher multipole degree increases the fraction of time spent in the dense matrix-vector product, which in turn improves the parallel effficiency (see, e.g., [5] for details).

The symbols used for the experiments are summarized below.

T	Execution time (seconds)
E	Parallel efficiency (%)
p	Number of processors
P_{OMP}	Number of OpenMP processes
P_{MPI}	Number of MPI processes

4.1 Shared Memory Parallelization

The benchmark problem presented in this section illustrates 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. The *ground plane problem* shown in Fig. 1 models the ground plane that is used to provide a uniform ground potential to all the components of a VLSI circuit.

Table 1. Parallel performance for the ground plane problem on shared memory machine.

p	Problem Size					
	128 × 128		256 × 256		512 × 512	
	T	E	T	E	T	E
1	770	100	3727	100	17164	100
2	385	100	1865	100	8610	99
4	193	99	938	99	4315	99
8	103	93	483	96	2203	97
16	55	87	250	93	1193	89

The problem requires computing the self-impedance of a 1cm × 1cm ground plane. A uniform two-dimensional mesh is used to discretize the ground plane. A tolerance of 10^{-3} was specified for the relative residual norm of the preconditioned GMRES method. Table 1 shows the execution time and efficiency for linear systems of order 32K, 128K and 512K unknowns. For a fixed size problem, a modest decrease in parallel efficiency with increase in the number of processors indicates an efficient parallel implementation. Figure 3 illustrates the scalability of the algorithm. It can be seen that by increasing the problem size, parallel efficiency is maintained when the number of processors are increased.

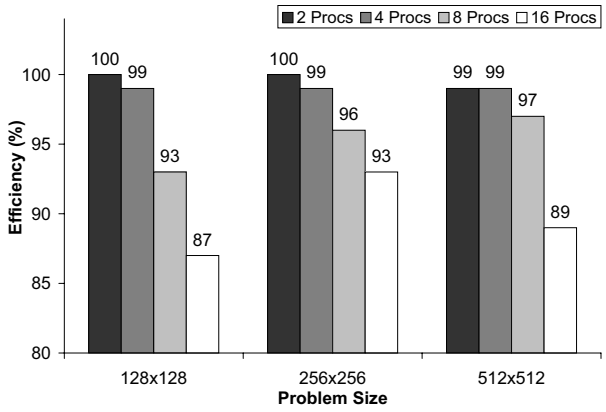

Figure 3. Shared memory parallel efficiency for the ground plane problem.

4.2 Mixed Mode Parallelization

The *cross-over* problem shown in Fig. 4 is a standard benchmark problem for inductance extraction. This setup represents a cross-over of interconnect segments. The problem consists of determining the impedance matrix for these segments. The segments are 1cm long and 2mm wide, and are separated by 500μm in the horizontal direction. This problem leads to a non-uniform point distribution for the dense matrix-vector multiplication algorithm.

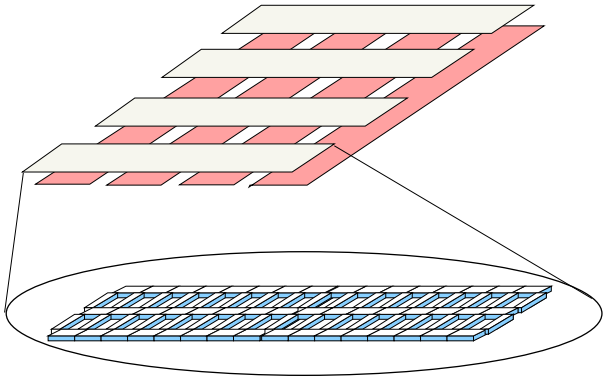

Figure 4. Cross-over problem with a view of a discretized conductor.

Mixed mode experiments were conducted on 16 processors of an IBM p690 at NCSA, Illinois. No more than 16 processors were available due to site restrictions. Various combinations of OpenMP and MPI processes were used to demonstrate the mixed mode parallel efficiency of the software. Each MPI process was responsible for one conductor and OpenMP directives were used to parallelize computation within the conductor.

Table 2. Parallel performance for the cross-over problem with 128 $\times$ 640 filaments per conductor.

P_{OMP}	P_{MPI}							
	1		2		4		8	
	T	E	T	E	T	E	T	E
1	578	100	582	99	597	99	618	98
2	286	100	292	99	296	100	316	96
4	150	96	150	96	158	94		
8	81	88	82	88				
16	49	74						

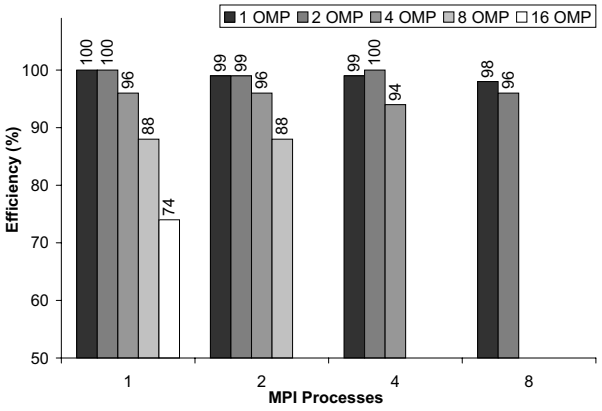

Figure 5. Mixed mode parallel efficiency for the cross-over problem with 128 $\times$ 640 filaments per conductor.

Table 2 shows the parallel performance of the software on the *cross-over* problem where each conductor has been discretized by a mesh of size 128 $\times$ 640. Figure 5 shows that the parallelism within a conductor is exploited very effectively via the OpenMP directives.

4.3 Distributed Memory Parallelization

The parallel performance of the software was studied on the IBM pSeries 690 as well as the Intel and AMD Linux clusters. The experiments were conducted on a 64-bit Intel Itanium cluster at NCSA, Illinois and on a 64-bit AMD Opteron-240 Tensor cluster at Texas A&M University. The Intel cluster consists of 800MHz 64-bit Itanium processors with Redhat-Linux operating system. Intel compilers were used to compile the code. The Tensor cluster consists of 1.4GHz 64-bit AMD Opteron processors with SuSE-Linux operating system. GNU compilers were used on the Tensor cluster for compiling the code.

Table 3 shows the execution time and parallel efficiency

Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)
0190-3918/04 $20.00 © 2004 **IEEE**

Table 3. Parallel performance for the cross-over problem on workstation clusters. Each conductor has been discretized by a 128 $\times$ 640 filament mesh.

	IBM p690		Intel-64 Linux		AMD-64 Linux	
p	T	E	T	E	T	E
1	578	100	738	100	300	100
2	585	99	746	99	301	99
4	604	98	765	99	310	99
8	629	97	790	99	320	99

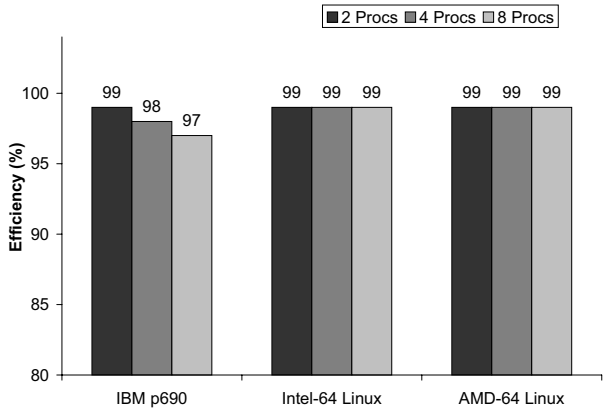

Figure 6. Parallel efficiency of the software for cross-over problem on different clusters. Each conductor was discretized by a 64 $\times$ 320 filament mesh.

of the software for the *cross-over* problem. 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 increase with increasing conductors. This is accompanied by a growth in the communication required among the processors. However, the parallel efficiency defined as the average base operations per second per processor is maintained across problem instances. This indicates that the code utilizes each processor efficiently when the load is distributed uniformly across processes.

5 Conclusion

This paper presents a high performance parallel software package called *ParIS* for inductance extraction of VLSI circuits. The software is based on a parallel formulation of the solenoidal basis method for inductance extraction. The computational complexity of the algorithm depends on the cost of computing matrix-vector products with dense coefficient and preconditioner matrices. *ParIS* uses an efficient parallel formulation of a hierarchical approximation scheme that is similar to the FMM algorithm. The software employs a two-tier approach in the mixed mode parallel code that is both portable and efficient on a variety of multiprocessors. Experimental results demonstrate that *ParIS* achieves very high parallel efficiency on shared-memory, distributed-memory, and distributed-shared memory multiprocessors.

References

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

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

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

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

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

[6] 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.

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

[8] 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.

[9] 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*, Vol. 27:118–141, 1995.

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