Parallel Performance of Hierarchical Multipole Algorithms for Inductance Extraction*

Hemant Mahawar[1,**], Vivek Sarin[1], and Ananth Grama[2]

[1] Department of Computer Science, Texas A&M University, College Station, TX, U.S.A.
{mahawarh, sarin}@cs.tamu.edu
[2] Department of Computer Science, Purdue University, West Lafayette, IN, U.S.A.
ayg@cs.purdue.edu

Abstract. Parasitic extraction techniques are used to estimate signal delay in VLSI chips. Inductance extraction is a critical component of the parasitic extraction process in which on-chip inductive effects are estimated with high accuracy. In earlier work [1], we described a parallel software package for inductance extraction called *ParIS*, which uses a novel preconditioned iterative method to solve the dense, complex linear system of equations arising in these problems. The most computationally challenging task in *ParIS* involves computing dense matrix-vector products efficiently via hierarchical multipole-based approximation techniques. This paper presents a comparative study of two such techniques: a hierarchical algorithm called Hierarchical Multipole Method (HMM) and the well-known Fast Multipole Method (FMM). We investigate the performance of parallel MPI-based implementations of these algorithms on a Linux cluster. We analyze the impact of various algorithmic parameters and identify regimes where HMM is expected to outperform FMM on uniprocessor as well as multiprocessor platforms.

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. At high frequencies, the physical proximity of interconnect segments leads to strong inductive coupling between neighboring conductors. This coupling arises because a magnetic field 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

* Support for Mahawar and Sarin was provided by NSF-CCR 9984400, NSF-CCR 0113668, and Texas ATP 000512-0266-2001 grants. Grama's research was supported by NSF-EEC 0228390 and NSF-CCF 0325227 grants. Computational resources were acquired through NSF-DMS 0216275 grant.

** Corresponding author.

L. Bougé and V.K. Prasanna (Eds.): HiPC 2004, LNCS 3296, pp. 450–461, 2004.

resistance offered to change in current within the 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 using 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 using iterative methods such as the generalized minimum residual method (GMRES) [2]. Each iteration requires a matrix-vector product with the coefficient matrix, which can be computed without explicitly forming the matrix itself. Matrix-vector products with the dense matrix are computed approximately via multipole algorithms such as the Fast Multipole Method (FMM) [3, 4].

In an earlier paper [1], we described an object-oriented parallel inductance extraction software called *ParIS*. The software uses a formulation in which 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 in which products with the dense coefficient matrix and the preconditioner are computed via FMM. Improved formulation and the associated preconditioning is responsible for significant reduction in computational and storage requirements [5]. *ParIS* achieves high parallel efficiency on a variety of multiprocessors with shared-memory, distributed-memory, and hybrid architectures.

In this paper, we present a comparative study of multipole-based methods for computing dense matrix-vector products. We consider the well-known FMM algorithm and a hierarchical algorithm, called Hierarchical Multipole Method (HMM), which can be considered as a variant of the FMM based on particle-cluster multipole evaluations only (related to a Barnes-Hut type approach [6]). We present parallel formulations of these methods and discuss their performance on a Beowulf cluster. We analyze the impact of parameters such as the multipole degree (d), the multipole acceptance criterion threshold (α), and the maximum number of particles allowed in a leaf box (s) on these methods. Since these parameters influence both accuracy and cost, it is important to develop a framework to select the optimal method for a given set of parameters. The experimental results presented in this paper can be used to identify the optimal method for difference parameter subspaces.

The paper is organized as follows – Section 2 outlines the inductance extraction problem, the solenoidal basis method, and the software design of *ParIS*; Section 3 describes HMM and FMM algorithms and outlines their parallel formulations; Section 4 presents a set of experiments on an AMD cluster to illustrate the performance of these methods for a range of parameters; and Section 5 presents concluding remarks.

2 Background

2.1 Inductance Extraction Problem

For a set of t conductors, we need to determine an $t \times t$ impedance matrix that represents pairwise 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 the 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 t 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 [7]

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

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'.

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

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

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

$$\mathbf{B}^T \mathbf{I}_f = \mathbf{I}_s, \tag{3}$$

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 corresponding to the source currents. 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 $\mathbf{V}_n$

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

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 using iterative techniques such as GMRES. The matrix-vector products with $\mathbf{L}$ 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.

2.2 The Solenoidal Basis Method

We present a brief overview of the solenoidal basis method for solving (4) (see, e.g., [5] for details). Consider the discretization of a ground plane shown in Fig. 1. Current flowing through the filaments must satisfy Kirchoff's law at each node in the mesh. The bold line indicates a path for current that satisfies boundary conditions. Current is made up of 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 converts the system in (4) into the following system with a different right hand side

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

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. Figure 1 shows how to construct unit circular flows on mesh cells that automatically satisfy Kirchoff's law at the grid nodes. The unknown filament currents can be expressed in the solenoidal basis: $\mathbf{I} = \mathbf{P}x$, where x is 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 flow in the filaments of the cell.

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

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

which is solved by a preconditioned iterative method. The preconditioning step involves product with a dense matrix that represents the inductive coupling among filaments placed at the cell centers. This preconditioning scheme can be implemented using FMM as well, and leads to rapid convergence of the iterative

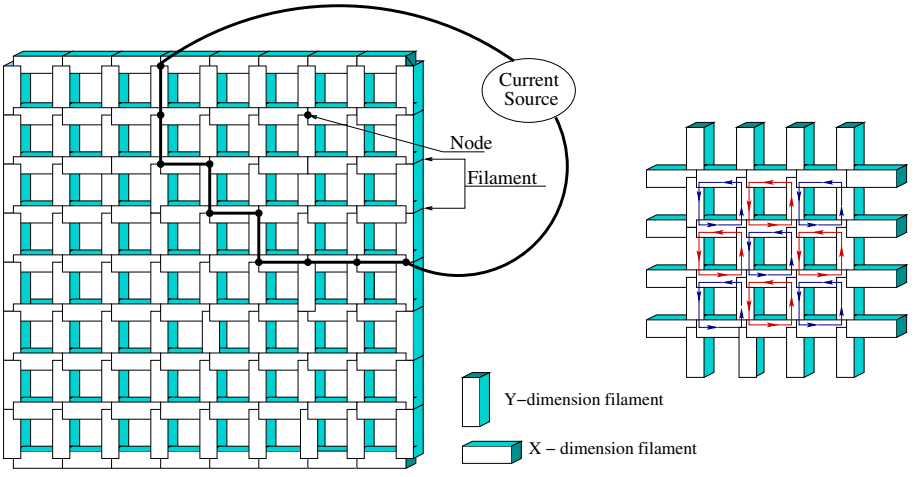

Fig. 1. Discretization of a ground plane with a mesh of filaments (left) and solenoidal current flows in each mesh cell (right) (Reproduced from [5])

method. On a set of benchmark problems, a serial implementation of this software is up to 5 times faster than FastHenry [7], a commonly available induction extraction software, with only one-fifth of memory requirements [5].

2.3 *ParIS*: Parallel Inductance Extraction Software

We have developed an object-oriented parallel implementation of the solenoidal basis algorithm for inductance extraction [1]. This software combines the advantages of the solenoidal basis method, fast hierarchical methods for dense matrix-vector products, and a 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 [8] on multiprocessors.

The building blocks of *ParIS* are conductor elements. Each conductor is uniformly discretized with a mesh of filaments. Kirchoff's law constraints on the filament currents of a conductor contribute a block in the system matrix. The most time-consuming step in the solution of the reduced system involves matrix-vector products with the impedance matrix, which is the sum of a diagonal matrix $\mathbf{R}$ and a dense inductance matrix $\mathbf{L}$. Since the preconditioning step involves matrix-vector product with a dense matrix, which is similar to $\mathbf{L}$, it is worthwhile to reduce the cost of the matrix-vector product with $\mathbf{L}$.

3 Hierarchical Multipole-Based Algorithms

The computational complexity of a matrix-vector product with a dense $n \times n$ matrix is $O(n^2)$. This can be reduced significantly through the use of hierarchi-

cal approximation techniques. These algorithms exploit the decaying nature of the $\frac{1}{r}$ kernel for the matrix entries to compute approximations with acceptable error. Higher accuracy can be achieved at the expense of more computation. Well-known techniques such as the Barnes-Hut [6] method compute particle-cluster interactions to achieve $O(n \log n)$ complexity, whereas the Fast Multipole Method (FMM) [4] computes cluster-cluster interactions in addition to particle-cluster interactions to achieve $O(n)$ complexity.

3.1 Hierarchical Multipole Method

The hierarchical multipole method (HMM) can be viewed either as an augmented version of the Barnes-Hut method or as a variant of FMM that uses only particle-cluster multipole evaluations. The method works in two phases: the tree construction phase and the potential computation phase. In the tree construction phase, a spatial tree representation of the domain is derived. At each step in this phase, if the domain contains more than s particles, where s is a preset constant, it is recursively divided into eight equal parts. This process continues until each part has at most s elements. The resulting tree is an unstructured oct-tree. Each internal node in the tree computes and stores an approximate multipole series representation of the particles contained in its subtree. The multipole series of a node is computed from the series of its children through an up-traversal of the nodes from the leaves to the root. Once the tree has been constructed, the potential at each particle can be computed as follows: a *multipole acceptance criterion* is applied to the root of the tree to determine if an interaction can be computed; if not, the node is expanded and the process is repeated for each of the eight children. The multipole acceptance criterion computes the ratio of the distance of the point from the center of the box to the dimension of the box. If this ratio is greater than α, where α is a constant greater than $\sqrt{3}/2$, an interaction can be computed.

3.2 Fast Multipole Method

ParIS uses a variant of FMM to compute approximate matrix-vector products with dense matrices. FMM is used to compute the potential at each filament due to the current flow in all filaments. The algorithm divides the domain into eight equal non-overlapping subdomains, and continues the process recursively until each subdomain has at most s filaments, where s 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. The multipole computation does 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.

3.3 Parallel Formulation

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 oct-tree. With the exception of matrix-vector products with the inductance matrix, all other computations are local to each conductor.

The matrix-vector product with the inductance matrix involves two types of filament interactions. Interactions among 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. Since matrix-vector products with the dense inductance matrix and the preconditioner are computed at each iteration, *ParIS* identifies those nodes in a conductor's tree that are required by other conductors during a pre-processing step. The cost of this step is amortized over the number of iterations of the solver. While computing the dense matrix-vector product, communication is needed for the translation of the multipole coefficients of these nodes to nodes on other processors. Communication is also needed between adjacent nodes that belong to different subtrees when computing direct interactions. 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 oct-tree, we are able to partition the computation for subdomains among processes. Fewer processes can be assigned to the top part of the oct-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 variety of parallel implementations are discussed in [9–12].

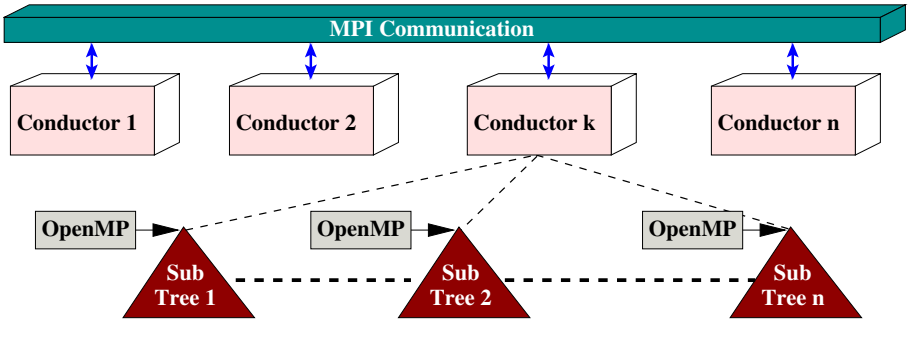

Fig. 2. Two-tier parallelization scheme implemented in *ParIS*

A two-tier parallelization approach shown in Fig. 2 simplifies the implementation 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 seamlessly [1].

4 Experiments

To investigate the performance of hierarchical multipole algorithms used in *ParIS* we considered the *cross-over* benchmark problem. Figure 3 shows two layered cross-over of interconnect segments called buses. The problem consists of determining the impedance matrix of these buses. Each bus is assumed to be 2cm long and 2mm wide. Buses within a layer are separated by 300μm while the layers are separated by 3mm. This problem leads to a non-uniform point distribution for the dense matrix-vector multiplication algorithm.

The main goal of this study is to analyze the performance of HMM and FMM codes within *ParIS*. Instead of solving the full inductance extraction problem, we observed the performance of the codes for a fixed number of GMRES iterations. Each iteration involved dense matrix-vector products with the coefficient matrix as well as the preconditioner. 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 (see, e.g., [5]).

A generalized notion of efficiency is used to provide a uniform basis to compare different experiments. We compute *scaled efficiency* as shown below:

$$E_s = \frac{BOPS}{p}, \tag{7}$$

where p is the number of processors and $BOPS$ is the average number of *base operations* executed *per second*. A base operation equals the cost of computing a direct interaction between a pair of filaments. In principle, $BOPS$ should remain unchanged when the number of conductors and filaments per conductor are varied. With this definition of efficiency, it possible to compare the performance of the code on a variety of benchmarks that require different number of interactions. The experiments were conducted on the *Tensor* cluster at Texas A&M University. The cluster consists of 1.4GHz 64-bit AMD Opteron processors running LAM/MPI on SuSE-Linux, connected via Giga-bit ethernet. GNU compilers were used on *Tensor* for compiling the code.

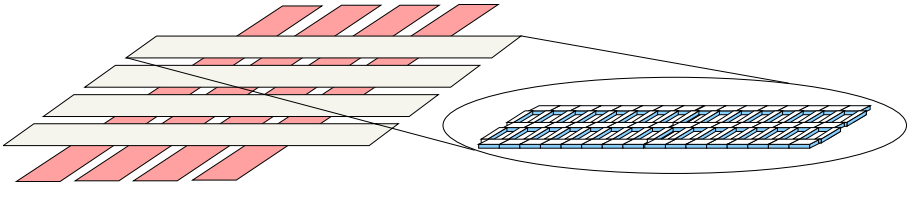

Fig. 3. The cross-over benchmark

4.1 Impact of Parameters

The performance of the hierarchical multipole algorithms depends on the choice of multipole degree (d), the multipole acceptance criterion determined by α, and the maximum number of particles allowed per leaf box (s). Since d and α parameters influence accuracy of the approximate dense matrix-vector product, a fair comparison is possible only when the impedance error is bounded. In these experiments, the impedance error was always within 1% of a reference value that was calculated by FMM with $d = 8$.

The dominant computation in FMM consists of multipole-to-local translations (M2Ls) with computational cost proportional to $(d + 1)^4$. The dominant computation in HMM consists of multipole evaluations at particles (M2Ps) with computational cost proportional to $(d + 1)^2$. Table 1 shows that with increase in d, the FMM time increases proportional to $(d + 1)^4$, while the HMM time increases proportional to $(d+1)^2$. For HMM experiments, α was chosen to be 1.

Table 1. Effect of the multipole degree (d) on the execution time, in secs, for different choices of maximum particles per leaf box (s)

	FMM code				HMM code			
d	s=2	s=8	s=32	s=128	s=2	s=8	s=32	s=128
1	49.5	18.3	12.7	29.9	25.7	21.5	21.3	34.8
2	225.8	62.5	25.3	32.8	46.8	36.5	31.3	41.9
4	1513.3	398.2	110.8	50.7	110.8	84.5	63.0	61.9

The execution time for both methods decreases when s is increased due to a decrease in the number of M2Ls and M2Ps. The cost of direct interactions is proportional to s^2 and is negligible for small values of s. Direct interactions begin to dominate the overall cost for large values of s, resulting in higher execution time. Table 1 shows that when s is increased, the FMM execution time reduces rapidly due to reduction in M2Ls, until the direct interactions begin to dominate the computational cost. Similarly, the HMM execution time decreases due to reduction in M2Ps, until the direct interactions begin to dominate. The decrease in the HMM case is not as rapid due to the lower complexity of M2Ps compared to M2Ls. For a given problem, one can identify (d, s) pair that minimizes the execution time for each method.

HMM has an additional parameter for the multipole acceptance criteria. A large value of α improves the accuracy of the approximate dense matrix-vector product at additional computational cost. Larger values of α increase the number of direct interactions as well as the number of M2Ps by ensuring that multipole evaluations at particles are computed for smaller boxes. This behavior is clear in Tables 2 and 3. The increase in time with α can be estimated from the increase in the number of direct interactions. A choice of $s = 8$ is used in Table 2 and $d = 2$ is used in Table 3.

Table 2. Effect of the multipole acceptance criterion threshold (α) on the execution time, in secs, of the HMM code for different choices of multipole degree (d)

α	d=1	d=2	d=4
1	21.5	36.5	84.5
1.5	40.1	70.6	158.2

Table 3. Effect of the multipole acceptance criterion threshold (α) on the execution time, in secs, of the HMM code for different choices of maximum particles per leaf box (s)

α	s=2	s=8	s=32
1	46.8	36.5	31.3
1.5	89.3	70.5	59.5

4.2 Parallel Performance

The parallel performance of FMM and HMM codes is primarily determined by the ratio of computation to communication. To compute M2L between a pair of oct-tree nodes residing on different processors, multipole coefficients must be exchanged. This requires communication of $(d+1)^2$ data units followed by M2L computation, which is proportional to $(d+1)^4$. Thus, the computation-to-communication ratio grows rapidly with increase in d. On the other hand, computing M2P between a node and a particle requires communication of $(d+1)^2$ data units followed by M2P computation, which is proportional to $(d+1)^2$. In this case, there will be limited effect of d on the parallel performance as long as the multipole coefficients received by a processor q are stored and reused by the particles on q.

The use of scaled efficiency E_s defined in (7) allows us to scale the problem linearly with processors. A cross-over problem with p conductors was chosen for experiments that used p processors. This benchmark is characterized by proximity between pairs of conductors on different layers. Thus, the number of M2Ls and M2Ps requiring communication grows linearly with the number of conductors p. Similarly, the number of direct interactions that require communication between processors also grows linearly with p. This is observed in Table 4 for the HMM code with $\alpha = 1$.

The computation in FMM is varied, with M2Ls forming the dominant component. Table 5 shows the parallel execution time for the FMM code for $s = 8$ and $s = 32$. The execution time grows much faster with p for the case when $s = 32$ because of reduced M2Ls and increased direct interactions. This behavior is consistent with the observation that the FMM code achieves higher parallel efficiency for larger d.

The scaled efficiency allows us to compare the performance if the two methods. Table 6 shows the efficiency of the HMM and FMM codes on the cross-over

problem with $s = 8$ and $\alpha = 1$. The codes maintain high efficiency as p increases. The efficiency also increases when d is increased, and the effect is more pronounced in the FMM code.

A comparison of the parallel execution time of the two methods for different values of d is also instructive. Table 7 shows the ratio of parallel execution times

Table 4. Impact of multipole degree (d) on the execution time, in secs, of the HMM code on p processors for two different choices of maximum particles per leaf box (s)

	$s = 8$				$s = 32$			
d	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
1	21.5	26.5	50.9	105.8	21.3	24.4	48.8	94.1
2	36.5	46.5	96.5	184.3	31.3	38.3	77.9	157.5
4	84.5	101.9	220.9	436.8	63.0	78.2	169.6	347.9

Table 5. Impact of multipole degree (d) on the execution time, in secs, of the FMM code on p processors for two different choices of maximum particles per leaf box (s)

	$s = 8$				$s = 32$			
d	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
1	18.3	25.7	34.5	59.2	12.7	13.9	40.4	94.4
2	62.5	72.5	87.5	131.3	25.3	26.6	58.0	126.3
4	398.2	431.4	470.9	683.3	110.8	113.4	165.7	277.8

Table 6. Efficiency of the extraction codes on p processors for different choices of multipole degree (d)

	HMM code				FMM Code			
d	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
1	0.99	0.93	0.94	0.86	0.98	0.74	0.87	0.87
2	1.00	0.92	0.90	0.92	0.99	0.86	0.97	0.98
4	1.00	0.98	0.93	0.94	1.00	0.93	1.04	0.98

Table 7. Ratio of the execution time of FMM and HMM codes on p processors for different choices of multipole degree (d)

	$s = 8$				$s = 32$			
d	p=1	p=2	p=4	p=8	p=1	p=2	p=4	p=8
1	0.9	1.0	0.7	0.6	0.6	0.6	0.8	1.0
2	1.7	1.6	0.9	0.7	0.8	0.7	0.7	0.8
4	4.7	4.2	2.1	1.6	1.8	1.4	1.0	0.8

of FMM and HMM codes for $d = 1, 2, 4$ and $s = 8, 32$. It is clear that HMM is superior to FMM when a larger value of d is used. The comparative advantage of HMM is diminished for $s = 32$ due to improved performance of FMM.

5 Conclusions

This paper presents a comparison of multipole-based methods for computing dense matrix-vector products in inductance extraction problems. The Fast Multipole Method is compared with a hierarchical multipole method on a set of benchmark problems. Numerical experiments are conducted on an AMD cluster for range of parameters such as the multipole degree (d), the multipole acceptance criterion threshold (α), and the maximum number of particles allowed in a leaf box (s). The results provide insight into the relative merits of these methods and suggest ways to determine the optimal method for a given set of parameters.

References

1. Mahawar, H., Sarin, V.: Parallel software for inductance extraction. In: Proceedings of the International Conference on Parallel Processing, Montreal, Canada (2004)
2. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Company, Boston (1996)
3. Greengard, L.: The Rapid Evaluation of Potential Fields in Particle Systems. The MIT Press, Cambridge, Massachusetts (1988)
4. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. Journal of Computational Physics **73** (1987) 325–348
5. Mahawar, H., Sarin, V., Shi, W.: A solenoidal basis method for efficient inductance extraction. In: Proceedings of the IEEE Design Automation Conference, New Orleans, Louisiana (2002) 751–756
6. Barnes, J., Hut, P.: A hierarchical O(n *log* n) force calculation algorithm. Nature **324** (1986) 446–449
7. Kamon, M., Tsuk, M.J., White, J.: FASTHENRY: A multipole-accelerated 3D inductance extraction program. IEEE Transaction on Microwave Theory and Techniques **42** (1994) 1750–1758
8. Mahawar, H., Sarin, V.: Parallel iterative methods for dense linear systems in inductance extraction. Parallel Computing **29** (2003) 1219–1235
9. Grama, A., Kumar, V., Sameh, A.: Parallel hierarchical solvers and preconditioners for boundary element methods. SIAM Journal on Scientific Computing **20** (1998) 337–358
10. Sevilgen, F., Aluru, S., Futamura, N.: A provably optimal, distribution-independent, parallel fast multipole method. In: Proceedings of the International Parallel and Distributed Processing Symposium, Cancun, Mexico (2000) 77–84
11. Singh, J.P., Holt, C., Totsuka, T., Gupta, A., Hennessy, J.L.: Load balancing and data locality in hierarchical n-body methods. Journal of Parallel and Distributed Computing **27** (1995) 118–141
12. Teng, S.H.: Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. SIAM Journal of Scientific Computing **19** (1998) 635–656