Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Parallel Computing 29 (2003) 1261–1273

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Multipole-based preconditioners for large sparse linear systems

Sreekanth R. Sambavaram [a,1], Vivek Sarin [a,1], Ahmed Sameh [b], Ananth Grama [b,*,2]

[a] *Department of Computer Science, Texas A&M University, 410 B, H.R. Bright Building, College Station, TX 77843-3112, USA*
[b] *Department of Computer Science, Purdue University, 250 N. University Street, West Lafayette, IN 47907-2066, USA*

Received 20 January 2003; received in revised form 16 June 2003; accepted 1 July 2003

Abstract

Dense operators for preconditioning sparse linear systems have traditionally been considered infeasible due to their excessive computational and memory requirements. With the emergence of techniques such as block low-rank approximations and hierarchical multipole approximations, the cost of computing and storing these preconditioners has reduced dramatically. This paper describes the use of multipole operators as parallel preconditioners for sparse linear systems. Hierarchical multipole approximations of explicit Green's functions are effective preconditioners due to their bounded-error properties. By enumerating nodes in proximity preserving order, one can achieve high parallel efficiency in computing matrix–vector products with these dense preconditioners. The benefits of the approach are illustrated on the Poisson problem and the generalized Stokes problem arising in incompressible fluid flow simulations. Numerical experiments show that the multipole-based techniques are effective preconditioners that can be parallelized efficiently on multiprocessing platforms.

Keywords: Multipole methods; Preconditioning; Iterative methods; Stokes problem

* Corresponding author.
E-mail address: ayg@cs.purdue.edu (A. Grama).

[1] The authors' work has been supported by the grants NSF-CCR 9984400 and NSF-CCR 9972533.
[2] The author's work has been supported by the grant NSF-CCR 9972533. Sameh's work has also been supported by NSF-CCR-9912388.

0167-8191/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-8191(03)00102-9

1. Introduction

Conventional preconditioners for sparse linear systems have focused on representations with limited fill that have sparse non-zero structure. This is critical for restricting the storage and computational complexity of the preconditioning step. In general, it is not necessary to have a sparse non-zero structure to restrict the cost associated with the preconditioner. Examples of such matrices include low-rank matrices, Toeplitz matrices, and, more interestingly, hierarchical operators based on analytical approximations such as multipole representations.

Multipole methods were first proposed in the context of particle dynamics methods and boundary element solvers. The intuition behind hierarchical multipole methods is that the influence of a set of entities coupled by the underlying Green's functions can be approximated by a bounded degree multipole series. For example, the gravitational potential due to a set of objects can be represented by their center of mass, provided the evaluation point is sufficiently far away. The coupling Green's function in this case is determined by the Laplace operator, and is $1/r$ in three dimensions, where r is the distance between source and observation points. With this simple intuition, the entire domain can be hierarchically decomposed and potentials computed at all n source/observation points in $O(n)$ time. This family of hierarchical multipole methods includes the fast multipole method [10] and the method of Barnes and Hut [1].

Hierarchical multipole methods can be adapted to solve integral equations for which a closed-form Green's operator and the associated multipole expansions are known. This has been used to solve scattering problems and problems in electrostatics with considerable success [2–6]. Since the multipole expansions satisfy boundary conditions implicitly, these methods have an added advantage over finite element and finite difference methods for problems where boundary conditions are specified at infinity. One such example is the Sommerfield radiation condition for scattering problems.

In many problems, however, a closed-form Green's function is not available. Examples of this include scattering from a dielectric and scattering from a dielectric embedded in a conductor. Even in these cases, the functions' decaying nature can be exploited to construct a block low-rank approximation to the matrix of coupling Green's functions. This allows computation and storage of the matrix as well as its application to a vector in $O(n)$ or $O(n \log n)$ time (depending on the specific method used). This principle forms the basis for dense solvers such as CDENSE [14].

The underlying premise of this paper is the simple observation that a multipole operator is the ideal preconditioner for the corresponding differential operator. For instance, a multipole operator with the Green's function $1/r$ may be used as a preconditioner for the Laplace operator to improve the rate of convergence of the iterative solver. The preconditioning step in each iteration requires $O(n)$ operations only since the cost of applying the multipole operator is $O(n)$. Indeed, this method competes against such effective preconditioners as multigrid and multilevel methods. However, since multipole operators are analytic in nature, they can be considered extensions of multigrid methods in which restriction and interpolation operators have

been analytically computed to yield bounded-error results (where a closed-form Green's function is available).

In this paper, we describe the application of a multipole-based preconditioner to the solution of the Poisson problem on a three-dimensional domain. To further illustrate the use of the preconditioner for more challenging problems, we consider the solution of the generalized Stokes problem which arises in the simulation of incompressible fluid flows. By using discrete divergence-free basis for velocity, the system is transformed into one that can be preconditioned effectively by a multipole-based preconditioner for the Laplace operator. The experiments presented in the paper indicate that (i) multipole-based preconditioners are very effective in improving the rate of convergence of iterative solvers for a variety of problems, (ii) the parallel implementation is very efficient and scalable, and (iii) the implementation of the solver–preconditioner is extremely simple since the preconditioner is geometric, but meshless. The code has demonstrated high efficiency on several platforms including the SGI Origin, IBM p690, and x86-based SMPs.

The rest of the paper is organized as follows: Section 2 presents a brief overview of multipole methods; Section 3 describes the multipole-based preconditioners and their application to the Poisson problem and the generalized Stokes problem; Section 4 discusses parallel formulations of the preconditioned solver and presents the experimental results; conclusions are drawn in Section 5.

2. Hierarchical multipole-based methods

We start with a brief overview of hierarchical multipole methods and their use in computing dense matrix–vector products in $O(n)$ time (or $O(n \log n)$ time) for an $n \times n$ matrix. Consider the problem of determining the charge on the surface of a conductor under given boundary conditions. The charge distribution is related to the potential by the following integral equation:

$$\psi(x) = \int_{\partial\Omega} G(x,x')\sigma(x')\,da', \tag{1}$$

where $\partial\Omega$ is the boundary of the domain Ω, $G(x,x')$ is the Green's function, $\psi(x)$ is the given potential at point x on the surface, $\sigma(x)$ is the unknown surface charge density, and da' represents the surface element. The Green's function $G(x,x')$ in three dimensions is given by $1/r$, where r is Euclidean distance between x and x'.

The surface charge density can be approximated as a weighted sum of n known expansion functions

$$\sigma(x) = \sum_{i=1}^{n} q_i r_i(x). \tag{2}$$

The coefficients of the expansion function can be determined by using a collocation condition of the form

$$p = Cq, \tag{3}$$

where p is a vector of the known potentials at n panels, C is an $n \times n$ coupling matrix of Green's functions, and q is the vector of unknown coefficients. Since a large number of panels are required for acceptable accuracy, iterative techniques must be used to solve the system (3). At each iteration, the most time-consuming component is the matrix–vector product of the form $p' = Cq'$, which requires computing the potential at a set of n collocation points due to the charge density of q', as indicated in (2).

There are several ways of computing the required potentials. The first approximation is that the charge densities are uniform over a panel and that the charge is represented as a point charge at the centeroid of the panel. This results in a discrete potential estimation problem similar to the n-body problem. This approximation is valid only when the source and observation panels are far apart; i.e., $\|x - x'\|$ is much larger than size of the element over which the integration is performed. This is not the case when we are dealing with spatially proximate elements. In these cases, Gaussian quadratures with higher number of Gauss points must be used to compute the potential.

The required matrix–vector product can be computed using approximate hierarchical methods. The matrix C is never computed explicitly. Instead panels are aggregated and their impact on other panels is expressed in terms of multipole expansions. We first introduce approximate hierarchical methods for particle dynamics and then illustrate how they can be used to compute matrix–vector products.

A number of hierarchical approximation techniques have been proposed for particle dynamics. Of these, the methods of Barnes and Hut [1], and the fast multipole method [10] have gained widespread popularity. In this paper, we use an augmented version of the Barnes–Hut method for computing the mat–vecs. The method works in two phases: the tree construction phase and the force 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 (for some preset constant s), 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 that sub-tree. Once the tree has been constructed, the force or 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 for the Barnes–Hut method computes the ratio of the distance of the point from the center of mass of the box to the dimension of the box. If this ratio is greater than $1/\alpha$, where α is a constant less than unity, an interaction can be computed. The Barnes–Hut method is illustrated in Fig. 1.

For computing matrix–vector products, particles correspond to Gauss points within elements and the force model between them corresponds to numerical integration. Computing a matrix–vector product in this manner involves the following steps:

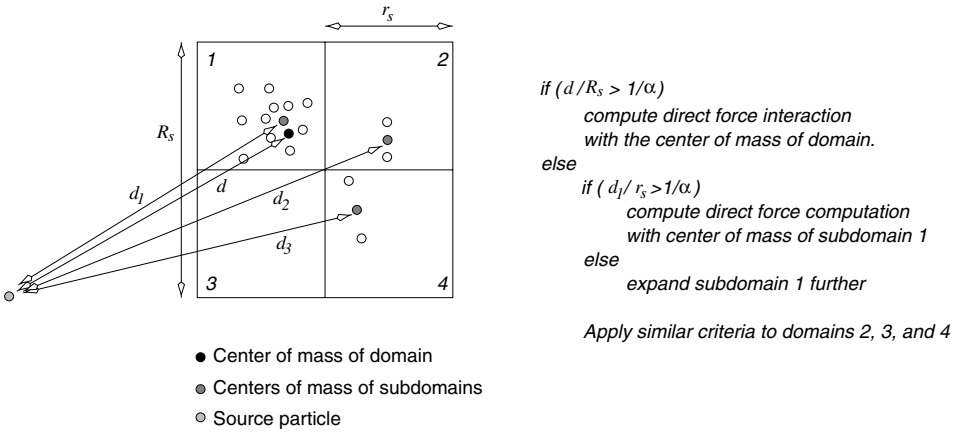

Fig. 1. Illustration of the Barnes–Hut method.

(1) Construct a hierarchical representation of the domain: given a domain discretization into elements, element centers correspond to particle coordinates. An octtree is constructed based on these element centers. Each node in the tree stores the extremities along x, y, and z dimensions of the subdomain corresponding to the node.
(2) The number of particles in the tree is equal to the product of the number of elements and the number of Gauss points in the far field. In the case of a single Gauss point in the far field, the multipole expansions are computed with the center of the element as the particle coordinate.
(3) For computing the matrix–vector product, we need to compute the potential at each of the n observation points. This is done using a variant of the Barnes–Hut method. The hierarchical tree is traversed for each of the elements. If an element falls within the near field of the observation element, integration is performed using direct Gaussian quadrature. The far-field contributions are computed using the multipole expansions. The α criterion of the Barnes–Hut method is slightly modified. The size of the subdomain is now defined by the extremities of all elements corresponding to the node in the tree. This is unlike the original Barnes–Hut method which uses the size of the box for computing the α criterion.

3. Multipole-based preconditioning techniques

Multipole-based approximation schemes described in the preceding section can be directly used as effective solvers/preconditioners for the Laplace operator.

Consider the following Laplace problem:

$$\Delta u = 0 \text{ in } \Omega, \quad u = g \text{ on } \partial\Omega. \tag{4}$$

Physically, this problem is identical to one of estimating the potential in the interior of a perfectly conducting body, given the boundary potential. Since Δu corresponds to

the charge and all charge resides on the boundary of a conductor, the correspondence is natural. The sparse linear system associated with this problem, $Ax = 0$, discretizes the Laplacian operator in the interior of the domain. To solve this sparse system, we can place m charges on the boundary of the conductor. We must now estimate these charges, subject to the constraint that the potential induced by these charges satisfies the boundary conditions. This reduces the problem to an equivalent dense linear system of the form $p = Cq$, discussed before. Here, C is the dense matrix of coupling Green's functions ($1/r$ in three dimensions). Iterative methods for solving this system require a matrix–vector product with matrix C, which can be computed using multipole methods. Once boundary charges have been estimated, the potential at interior points can be estimated easily using multipole methods as well. If a higher level of accuracy is desired, this multipole-based solver can be used as a preconditioner for an outer sparse solver.

3.1. Application to the Poisson problem

Consider, now, the problem of computing a function u on a domain Ω such that

$$\Delta u = f \text{ in } \Omega, \quad u = g \text{ on } \partial\Omega, \tag{5}$$

where f and g are known functions. The associated linear system of equations has the form

$$Ax = b, \tag{6}$$

in which b and x can again be interpreted as vectors of charges and potentials, respectively, at n points in the interior of the domain. In contrast to the previous case, the potential at the boundary points is a result of interior as well as boundary charges. To reduce the problem to one defined on the boundary nodes, we must first cancel the effect on boundary potential of the internal nodes. To do this, we first locate m points on the boundary where we estimate the potential due to internal nodes. We subtract this potential from the specified boundary conditions f to define a dense problem over the m boundary nodes. The charges at these boundary nodes are therefore determined by solving an $m \times m$ linear system of the form (3) with $p = g - g'$ where g' is the potential on the boundary due to interior charges. Once the boundary charges are known, x can again be computed as the potential due to the interior and boundary charges.

The linear system for the boundary charges is solved using an iterative method. The multipole-based approximations that are used to compute matrix–vector products with the dense coefficient matrix introduce errors in the solution. Nevertheless, one can use this approach as a preconditioning step for an outer iterative method which is used to solve (6). Since the linear system for the preconditioning step is defined only on the boundary, the inner iterative solve is not expensive. This is evident from Table 1, which shows the number of iterations and total time taken by the preconditioned CG method for solving the Poisson problem on a uniform three-dimensional grid. The experiments were conducted on a 2.4 GHz Pentium 4 workstation with 1GB memory. The outer CG method was terminated when the relative

Table 1
Performance of the multipole-based preconditioner for the Poisson problem

Mesh size	Nodes	Unpreconditioned		Preconditioned	
		Iterations	Time (s)	Iterations	Time (s)
25×25×25	15 625	1301	3.86	3	4.06
30×30×30	27 000	1895	14.06	4	11.74
40×40×40	64 000	3451	61.97	3	17.97
50×50×50	125 000	5465	184.68	3	39.67
60×60×60	216 000	7936	413.02	3	67.90

residual norm was reduced below 10^{-5}. The inner GMRES method used a Krylov subspace of dimension 3 and a tolerance of 0.05 on the relative residual norm.

3.2. Application to incompressible flows

The generalized Stokes problem arises in the solution of time-dependent Navier–Stokes equations for incompressible fluid flows. It consists of solving the following linear system:

$$\begin{bmatrix} \frac{1}{\Delta t}M + \nu L & B \\ B^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}, \tag{7}$$

where u is the fluid velocity, p is the pressure, Δt is the time step, ν is the viscosity, and M, L, and B^{T} are the mass, Laplace, and divergence matrices, respectively. The linear system is large, sparse, and indefinite due to the incompressibility constraint $B^{\mathrm{T}}u = 0$, which forces the velocity to lie in a discrete divergence-free subspace. While the primary challenge in developing robust preconditioners for this system is due to the indefiniteness of the matrix, parameters such as viscosity and time step also influence the effectiveness of the preconditioner.

The linear system in (7) can be transformed to the following *reduced system* by restricting velocity to the discrete divergence-free subspace:

$$P^{\mathrm{T}}\left[\frac{1}{\Delta t}M + \nu L\right]Px = P^{\mathrm{T}}f, \quad u = Px. \tag{8}$$

Here, P is a basis for the discrete divergence-free subspace which is identical to the null space of B^{T}. While the symmetric positive definite character of the reduced system allows use of the CG method, it introduces additional complexity in constructing preconditioners.

In order to compute a discrete divergence-free basis P with modest computational and storage requirements, we observe that circulating flows or vortices can be used as a divergence-free basis in the continuum space. The discrete counterpart uses the edges of a mesh to define circulating flows that are divergence-free in the discretized domain. One can obtain these flows by computing the null space of submatrices of B^{T} defined over local regions of the mesh. Each such flow is represented as a vector, and the set of these vectors forms the columns of P (see, e.g., [11] for details). The

matrix P is sparse due to the local support of the flows. In addition, matrix–vector products with P can be computed directly from the local flow vectors without assembling the matrix itself. Due to the unavailability of P, several commonly used preconditioning techniques such as those based on incomplete factorizations are no longer viable.

The similarity of this approach with vorticity–velocity function formulation of the Navier–Stokes equations for incompressible flows can be exploited to develop a preconditioner for the reduced system. We observe that the matrix–vector products Px and $P^{\mathrm{T}}y$ compute the discrete curl of the functions represented by x and y, respectively. Furthermore, the matrix–vector product $P^{\mathrm{T}}Px$ represents the application of the Laplace operator, say L_s, defined on the discrete divergence-free space. The reduced system may be approximated as shown below:

$$\frac{1}{\Delta t}P^{\mathrm{T}}MP + vP^{\mathrm{T}}LP \approx \frac{1}{\Delta t}M_sL_s + vL_s^2,$$

where M_s is the equivalent mass matrix for the flow vectors, and preconditioned by the following matrix:

$$G = \left[\frac{1}{\Delta t}M_s + vL_s\right]L_s. \tag{9}$$

Since the preconditioned system is spectrally equivalent to a symmetric positive definite matrix, one can use preconditioned CG to solve the reduced system (8). Experiments reported in [11] show that the preconditioner is very effective over a large range of values for Δt and v.

To illustrate the effectiveness of the preconditioner, we consider the driven cavity problem on a three-dimensional cube. We use the Marker-and-Cell (MAC) scheme in which the domain is discretized by an $n \times n \times n$ uniform mesh. Pressure unknowns are defined at the nodes and velocity unknowns are defined at the mid-point of the edges. The x-component of velocity is defined on the edges along x-axis. Similarly, y- and z-components of the velocity are defined on the edges along y-axis and z-axis, respectively. This gives a linear system with n^3 pressure unknowns and $3n^2(n-1)$ velocity unknowns.

The mesh is made up of an $(n-1) \times (n-1) \times (n-1)$ array of cubes. The local divergence-free flows are defined on the faces of these cubes. The normals to the faces are used to divide the flows into x-, y-, and z-components. The preconditioning step requires computing $z = [\Delta t^{-1}I + vL_s]^{-1}L_s^{-1}r$, where L_s is a Laplace matrix with a block diagonal structure: $L_s = \mathrm{diag}[L_x, L_y, L_z]$. The matrix L_x is the Laplace operator for x-component of the divergence-free flows, and is defined on a mesh with nodes at the mid-points of the faces supporting these flows. The matrices L_y and L_z are defined in a similar manner. Dirichlet boundary conditions are assumed in each case.

The influence of the mesh width h, viscosity v and time step Δt is indicated by the following condition number estimate:

$$\kappa\left(\frac{1}{\Delta t}M + vL\right) = \frac{\tau + 12}{\tau + h^2}, \quad \tau = \frac{h^2}{v\Delta t}.$$

Table 2
Effectiveness of the preconditioner for the generalized Stokes problem ($\tau = 10^{-3}$)

Mesh size	Unknowns	Iterations	
		Unpreconditioned	Preconditioned
$8\times8\times8$	1856	66	8
$16\times16\times16$	15 616	208	12
$32\times32\times32$	128 000	772	17

Table 3
The number of iterations required by the preconditioned CG for various instances of the generalized Stokes problem

Mesh size	Unknowns	$\tau = 10^{-3}$	$\tau = 10^{-1}$	$\tau = 10^{0}$	$\tau = 10^{1}$	$\tau = 10^{3}$
$8\times8\times8$	1856	8	8	6	5	5
$16\times16\times16$	15 616	12	10	8	6	6
$32\times32\times32$	128 000	17	13	10	7	7

Table 2 shows that the number of iterations required by the CG method to solve the reduced system in (8) is dramatically reduced by the use of the preconditioner in (9). The preconditioning step uses the multipole-based preconditioner for the Poisson problem and an inner CG method for the Helmholtz problem. The diagonal dominance of the Helmholtz problem assures rapid convergence without additional preconditioning. The outer CG iterations were terminated when the relative residual norm reduced below 10^{-4} whereas the inner iterations used a tolerance of 10^{-2}. This variation in the preconditioner over successive outer iterations can be accommodated by using a flexible CG as the outer solver.

Table 3 shows the number of iterations required by the preconditioned CG method for several instances of the generalized Stokes problem. Depending upon the value of τ, the condition number of the reduced system ranges from $O(h^{-2})$ to $O(h^{-4})$. The preconditioner ensures a stable convergence rate which is nearly independent of problem parameters.

4. Parallel formulation

The multipole-based preconditioning step is the most expensive component of the iterative solver. The execution time in the experiments reported in Section 3 is dominated by the hierarchical multipole-based matrix–vector products with the dense preconditioner (over 90% of total time is spent in the preconditioning step). The focus of this section is therefore to develop efficient parallel formulations of the multipole-based preconditioning step. The remaining components of the iterative solver include sparse matrix–vector products and vector operations, which can be parallelized easily once the mesh has been partitioned across processors (using conventional mesh partitioners such as METIS [8] and CHACO [7]).

As described in Section 3.1, the multipole-based preconditioner is posed as a boundary-enforced solve in which unknown boundary charges are computed to satisfy the boundary condition on potential. The interior and boundary charges are then used to evaluate potential inside the domain. The kernel operation in each case is a dense matrix–vector product, which is computed using a multipole-based hierarchical method. A single instance of this method requires a tree construction and a tree traversal. Since the tree construction phase is relatively inexpensive (requiring less than 2% of total time in our experiments) we focus on efficient parallelization of the tree traversal phase. It has been observed by us and others in the past [9,12,13] that an effective parallelization strategy can be derived from the observation that two spatially proximate particles are likely to interact with largely the same nodes in the tree. This leads to a partitioning strategy in which spatially proximate particles are aggregated into a single concurrent computational unit.

In our parallel formulation, we first sort the nodes in the mesh in a proximity preserving order such as a Peano–Hilbert ordering, and group a set of m nodes into a single thread. The parameter m should be chosen in such a way that the number of threads is greater than the number of processors p by a factor $\log p$. This is generally adequate for ensuring load balance in the tree traversal phase as well as amortizing the cost of remote communication. This parallel formulation of multipole methods on SMP multiprocessors such as the SGI Origin and IBM p690 is observed to yield high performance and excellent scalability.

4.1. Performance of multipole-based methods

In this section, we profile the parallel performance of the multipole operator in isolation, followed by the preconditioned solve. The multipole operator is implemented in POSIX threads and is portable across a range of serial and parallel platforms. The performance of the operator is a function of a number of parameters—the degree of multipole series, the α parameter of the Barnes–Hut method, and the underlying architecture, in addition to the problem size.

Increasing the degree d of the multipole series increases the volume of data (as $O(d^2)$) that must be communicated from remote nodes. While the corresponding computation also increases (as $O(d^2)$), the associated per-word communication time exceeds per-degree computation time. Therefore the efficiency can be expected to decrease with increasing degree. (Note that this does not hold for translation of multipole operators since translation complexity varies as $O(d^4)$.) Decreasing the α parameter of the Barnes–Hut method increases the range of interaction, and therefore results in a lower efficiency for the corresponding parallel formulation.

In Table 4, we present the parallel performance of the multipole operator for a range of problem sizes and multipole degrees. These results correspond to a single dense matrix–vector product, which is encapsulated in an inner GMRES preconditioning solve. It is evident from the table that the parallel formulation of the multipole operator yields excellent performance across a range of problem sizes.

Fig. 2 shows the speedup achieved by a single multipole-based potential estimation on a 32-processor SGI Origin. It is evident that for reasonable problem sizes

Table 4
Parallel execution time (in seconds) and speedups (in parenthesis) on a 32-processor SGI Origin of a single multipole-based dense matrix–vector product

Nodes	Multipole degree		
	$d = 3$	$d = 5$	$d = 7$
4096	0.14 (20.38)	0.40 (19.31)	0.73 (19.20)
32 768	1.63 (26.04)	4.70 (25.10)	9.26 (24.27)
65 536	4.04 (28.08)	11.88 (26.53)	23.11 (25.91)

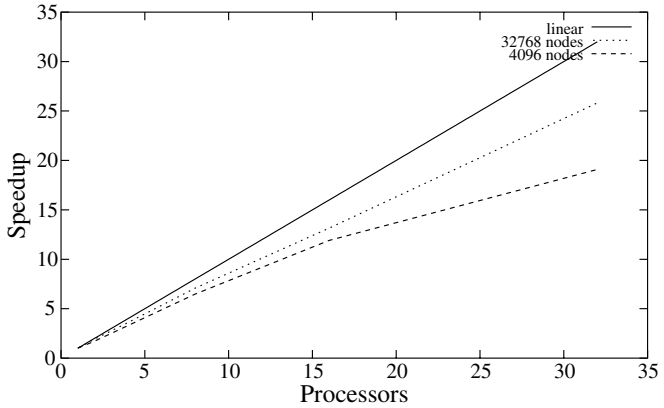

Fig. 2. Speedup on a 32-processor SGI Origin of the preconditioned solve for two problem instances with 32768 and 4096 nodes, respectively.

(with at least 32K nodes), the solver yields over 80% parallel efficiency on up to 32 processors. This efficiency can be expected to further improve as the number of unknowns is increased. This validates our claims of multipole operators as highly parallelizable and effective preconditioners.

4.2. Performance of multipole-based preconditioners

The preconditioning step in the CG method consists of three different types of multipole-based dense matrix–vector products: (i) computation of boundary potential from interior charges, (ii) computation of boundary charges via an inner GMRES method that requires boundary potential computation from boundary charges at each iteration, and (iii) computation of interior potential from all the charges. Since the last step of the computation involves the greatest number of potential evaluations, it dominates the cost of the preconditioning step. The objective of this set of experiments is to demonstrate that it is possible to achieve high efficiency in an algorithm that requires each of these three steps (with associated data redistribution).

Table 5
Parallel execution time (in seconds) and efficiency of the preconditioned CG method on the IBM p690 and SGI Origin shared-memory multiprocessors

Mesh size	IBM p690			SGI origin		
	$p = 1$	$p = 8$	Efficiency	$p = 1$	$p = 32$	Efficiency
$25\times25\times25$	7.72	1.21	0.80	12.06	1.17	0.32
$30\times30\times30$	19.96	3.25	0.77	30.78	2.44	0.39
$40\times40\times40$	31.18	4.80	0.81	48.29	3.40	0.44
$50\times50\times50$	68.88	9.69	0.81	96.28	5.73	0.52
$60\times60\times60$	108.17	15.47	0.87	163.63	7.41	0.69

Table 6
Parallel execution time (in seconds) and efficiency of the preconditioned CG method on x86 Solaris shared-memory multiprocessors

Mesh size	4-processor SMP (550 MHz P3)			8-processor SMP (750 MHz Xeon)		
	$p = 1$	$p = 4$	Efficiency	$p = 1$	$p = 8$	Efficiency
$25\times25\times25$	16.51	5.73	0.72	11.44	2.73	0.52
$30\times30\times30$	41.39	13.19	0.78	28.78	5.97	0.60
$40\times40\times40$	67.37	20.76	0.81	47.23	7.19	0.82
$50\times50\times50$	129.87	39.49	0.82	115.29	16.70	0.86

Table 5 shows that execution time and parallel efficiency of the preconditioned CG solver on the IBM p690 and SGI Origin. On 8 processors of the IBM p690, the code exhibits parallel efficiency in excess of 80% in most cases. On the 32-processor SGI Origin, the efficiency grows with the problem size, attaining a maximum value of 69%. It must be noted that the corresponding problem size per processor for the Origin is small—contributing to the moderately high efficiency.

Table 6 shows that the parallel efficiency of the code is retained on x86 Solaris SMPs with up to 8 Pentium processors. These results indicate that for multipole-based preconditioners, these SMPs can be an inexpensive alternative to high-end multiprocessors.

4.3. Implementation details

The development of our multipole-based preconditioned solver is an interesting study in code development and robustness of parallel programming paradigms. The CG solver is written in OpenMP, which is a directive based programming model relying on `parallel for` directives for partitioning loops across threads (and consequently, processors). The multipole preconditioner is written using POSIX threads (pthreads), which directly partition the computation across threads. The only consideration with respect to parallelism, while integrating these codes, was to ensure that the parallel regions of OpenMP were closed before POSIX threads were created. With this simple guideline, the two codes were seamlessly integrated. As we have shown in our experimental results, the parallel performance of the integrated code is excellent.

5. Conclusions

In this paper, we have described a strategy for preconditioning sparse linear systems with multipole operators. The approach has been used to solve a three-dimensional Poisson problem. The technique has also been applied to the generalized Stokes problem in which the reduced linear system obtained by restricting velocity to divergence-free subspace is preconditioned by the multipole operator. Numerical experiments presented in the paper illustrate the benefits of using multipole-based preconditioners to improve the rate of convergence of the iterative solver. Additional experiments indicate that parallel implementations of these techniques can achieve high efficiency on a variety of shared-memory machines including SGI Origin, IBM p690, and x86 SMPs.

References

[1] J. Barnes, P. Hut, A hierarchical O($nlogn$) force calculation algorithm, Nature (1986) 324.
[2] R. Coifman, V. Rokhlin, S. Wandzura, The fast multipole algorithm for the wave equation: a pedestrian prescription, IEEE Antennas Propag. Mag. 35 (3) (1993).
[3] E. Darve, The fast multipole method (I): error analysis and asymptotic complexity, SIAM J. Numer. Anal. 38 (1) (2000) 98–128.
[4] E. Darve, The fast multipole method: numerical implementation, J. Comp. Phys. 160 (1) (2000) 195–240.
[5] B. Dembart, E. Yip, A 3D fast multipole method for electromagnetics with multiple levels, in: Conference Proceedings. 11th Annual Review of Progress in Applied Computational Electromagnetics, Monterey, CA, 1995.
[6] N. Engheta, W. Murphy, V. Rokhlin, M. Vassiliou, The fast multipole method for electromagnetic scattering problems, IEEE Trans. Antennas Propag. 40 (6) (1992).
[7] B. Hendrickson, R. Leland, The CHACO user's guide version 2.0, Technical Report sand94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.
[8] G. Karypis, V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs, SIAM Rev. 41 (2) (1999) 278–300.
[9] A. Grama, V. Kumar, A. Sameh, Parallel hierarchical solvers and preconditioners for boundary element methods, SIAM J. Sci. Comput. 20 (1) (1998) 337–358.
[10] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comp. Phys. 73 (1987) 325–348.
[11] S.R. Sambavaram, High Performance Parallel Algorithms for Incompressible Flows, M.S. Thesis, Texas A&M University, 2002.
[12] J.P. Singh, C. Holt, J.L. Hennessy, A. Gupta. A parallel adaptive fast multipole method, in: Proceedings of the Supercomputing '93 Conference, 1993.
[13] M.S. Warren, J.K. Salmon, A parallel hashed oct-tree N-body algorithm, in: Supercomputing '93 Proceedings, Washington, DC, IEEE Comp. Soc. Press, 1993, pp. 12–21.
[14] S. Goreinov, E. Tyrtyshnikov, A. Yeremin, Matrix-fee iterative solution strategies for large dense linear systems, Technical Report EM-RR 11/93, Elegant Mathematics, Inc., 1993.