ELSEVIER

Parallel Computing 28 (2002) 285–299

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Parallel algorithms for indefinite linear systems

Ahmed H. Sameh [a,1], Vivek Sarin [b,*,2]

[a] Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA

[b] Department of Computer Science, Texas A&M University, College Station, TX 77843, USA

Abstract

Saddle-point problems give rise to indefinite linear systems that are challenging to solve via iterative methods. This paper surveys two recent techniques for solving such problems arising in computational fluid dynamics. The systems are indefinite due to linear constraints imposed on the fluid velocity. The first approach, known as the multilevel algorithm, employs a hierarchical technique to compute the constrained linear space for the unknowns, followed by the iterative solution of a positive definite reduced problem. The second approach exploits the banded structure of sparse matrices to obtain a different reduced system which is determined by the unknowns common to adjacent block rows. Although the reduced system in this approach may still be indefinite, the algorithm converges to the solution at an accelerated rate. These methods have two desirable characteristics, namely, robust numerical convergence and efficient parallelizability. The paper presents the performance of these methods for incompressible particulate flow problems on a shared-memory parallel architecture. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Parallel computing; Saddle-point problems; Preconditioning; Iterative methods; Computational fluid dynamics

[*] Corresponding author.

E-mail addresses: sameh@cs.purdue.edu (A.H. Sameh), sarin@cs.tamu.edu (V. Sarin).

[1] This work has been supported in part by NSF under the grants NSF-CTS 9873236 and NSF-CCR 9972533.

[2] This work has been supported in part by NSF under the grants NSF-CCR 9984400, NSF-CCR 9972533, and NSF-CTS 9873236.

0167-8191/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-8191(01)00140-5

1. Introduction

Efficient solution of linear systems of equations is a critical component of most scientific simulations. Linear systems that arise from the discretization of partial differential equations are typically large and sparse. The most effective solution strategy involves an iterative algorithm with a preconditioning step that ensures rapid convergence to the solution. Preconditioning may be viewed as a way to provide a cheap approximation to the solution to assist in convergence of the iterations. It has been observed that the performance of iterative methods and preconditioners is highly sensitive to the eigenvalues of the linear system. For instance, well-conditioned systems such as diagonally dominant systems can be solved quickly by many iterative methods. Indefinite linear systems are among the most challenging systems of equations. These systems are characterized by eigenvalues that lie on both sides of the imaginary axis. Well-known iterative methods can have arbitrarily slow convergence, and commonly used preconditioners may not be useful either. The reader is referred to [1,9] for a detailed description of preconditioned iterative methods.

In this paper, we focus on indefinite linear systems of equations that arise from incompressible fluid simulations. The Navier–Stokes equations consist of the momentum equation which conserves fluid momentum and the continuity equation which enforces incompressibility. The incompressibility condition is satisfied via a set of linear constraints, ensuring that the fluid velocity is divergence-free. The resulting linear system is a saddle-point system which is indefinite due to the constraints. Exploiting the block structure of these systems, one can devise effective iterative algorithms that circumvent the problem of indefiniteness by solving the positive definite Schur-complement of the saddle-point system. These techniques are known as Uzawa methods. Several researchers have proposed variants of this basic iterative scheme [2,8] and preconditioners that exploit the block structure in a similar way [13,14].

Our approach is based on the construction of an explicit basis that satisfies the incompressibility constraints. A divergence-free velocity is computed by multiplying an arbitrary vector with the matrix that represents the basis. This yields a reduced system defined on the subspace of divergence-free velocity. The reduced system is solved via an iterative method such as the method of conjugate gradients (CGs) or generalized minimum residual method (GMRES). Appropriate choice of the divergence-free basis accelerates the convergence of these methods by acting as a preconditioner. We outline a hierarchical technique to compute this basis that allows computation and application of the matrix to a vector in time proportional to the size of the saddle-point linear system. We also describe the parallelization of the computation of the hierarchical basis and matrix–vector products with it. Details of the algorithm are available in [11,12].

The use of divergence-free basis has been proposed by several researchers (see, e.g. [4,15]). There are two features that distinguish our approach. First, the basis is computed using linear algebraic techniques which allows our scheme to be applied to a variety of discretization schemes including finite difference and finite element methods. Secondly, the hierarchical technique is designed to produce well-conditioned basis that accelerate the convergence of the iterative methods in particular instances of the problem.

The hierarchical divergence-free basis approach has been applied to particulate flows [5] in which one needs to solve a coupled system of equations consisting of Navier–Stokes equations for the incompressible fluid and Newton's equation of motion for the rigid particles suspended in the fluid. We also discuss an alternate approach that solves the Schur-complement system defined either on the pressure unknowns or on the particle variables. In this scheme, the matrix is reordered into a narrow banded sparse matrix and partitioned into several block rows. A reduced system is solved for the unknowns common to consecutive blocks via an iterative method. Each iteration requires concurrent projections onto subspaces defined by the block rows of the matrix. Parallelism at the block level as well as within each projection step can be exploited to obtain an efficient parallel algorithm for these saddle-point systems. The algorithm is described in greater detail in [3] and its application to particulate flows is outlined in [10].

The paper is organized as follows. Section 2 describes a class of indefinite linear systems known as saddle-point problems. The multilevel algorithm for saddle-point systems arising in incompressible fluid simulations is outlined in Section 3. In Section 4, we present the extension of the multilevel algorithm to particulate flows. This is accompanied by an alternate scheme to solve the narrow banded systems arising in such problems. We also present experimental results for the numerical and parallel performance of both these schemes. Conclusions are presented in Section 5.

2. Saddle-point problems

Saddle-point problems are a class of linear systems that have indefinite matrices. We consider systems of the form

$$\begin{bmatrix} A & B \\ B^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{1}$$

where A is an $n \times n$ symmetric positive definite or real positive matrix, and B^{T} is a $k \times n$ matrix of constraints imposed on u. Both A and B are large and sparse.

Saddle-point systems occur in a number of different problems including fluid problems, interior point methods, electrostatics, elasticity, etc. The common feature in all these problems is the set of linear constraints on the unknown u that introduces negative eigenvalues in the system.

In this paper, we focus on the solution of saddle-point systems arising from Navier–Stokes equations applied to incompressible fluid flows. The Navier–Stokes equations for incompressible, viscous flow in region Ω with boundary $\partial\Omega$ are

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{R}\Delta \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega, \tag{2}$$

where $p = p(\mathbf{x}, t)$ is the pressure, R is the Reynolds number, $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ is the three-dimensional velocity vector at $\mathbf{x} \in R^3$. The velocity $\mathbf{u}$ is assumed to be zero on the domain boundary. The first equation is an expression of the law of conservation of momentum, while the second enforces incompressibility of the fluid. This is a nonlinear system of equations with linear constraints.

Discretization and linearization of (2) gives the saddle-point system (1), where B^T is the discrete divergence operator, and A is the discrete counterpart of the terms with $\mathbf{u}$: $A = M/\Delta t + N + L/R$. Over here, M is the mass matrix, L is the Laplace matrix, and N is the nonsymmetric matrix arising from the convection term. The time step Δt and the Reynolds number R are scalar quantities. When operator-splitting is used to separate the linear and nonlinear terms, we obtain a generalized Stokes problem with a symmetric positive definite A: $A = M/\Delta t + L/R$. For more details, the reader is referred to a standard text on computational fluid dynamics, e.g. [7].

3. A multilevel algorithm

The saddle-point system (1) can be solved by computing the following decomposition of B

$$P^T B Z = \begin{bmatrix} D \\ 0 \end{bmatrix}, \tag{3}$$

where P is a nonsingular $n \times n$ matrix, D is a $k \times k$ diagonal matrix, and Z is a $k \times k$ orthogonal matrix. When P is orthogonal, this decomposition is identical to the singular value decomposition (SVD) of B. Since an orthogonal P is prohibitively expensive to compute and store, the decomposition only requires P to be nonsingular.

Suppose $P = [P_1, P_2]$, where P_1 consists of the first k columns and P_2 consists of the last $n - k$ columns of P. The linear system (1) is rewritten as

$$\left[\begin{array}{cc|c} P_1^T A P_1 & P_1^T A P_2 & D \\ P_2^T A P_1 & P_2^T A P_2 & 0 \\ \hline D & 0 & 0 \end{array}\right] \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ Z^T p \end{bmatrix} = \begin{bmatrix} P_1^T f \\ P_2^T f \\ Z^T g \end{bmatrix}$$

after multiplication of first block with P^T and the second block with Z^T, and replacing the unknowns u with $\hat{u}$, where $u = P\hat{u}$. With this transformation, the solution is written as

$$\hat{u}_1 = D^{-1} Z^T g, \quad \hat{u}_2 = [P_2^T A P_2]^{-1} P_2^T [f - A P_1 \hat{u}_1], \quad p = Z D^{-1} P_1^T [f - A P \hat{u}].$$

Since D is a diagonal matrix, the only time-consuming step is the solution of the *reduced* system

$$P_2^T A P_2 \hat{u}_2 = P_2^T [f - A P_1 \hat{u}_1]. \tag{4}$$

We have to use an iterative method since the matrix $P_2^T A P_2$ is expensive to compute and store explicitly. The matrix–vector product with $P_2^T A P_2$ is computed as a series of products with the individual matrices P_2, A, and P_2^T in that order.

The success of this method depends upon the choice of P. Ideally, P must have the following characteristics: (i) P must be computed in $O(n)$ operations; (ii) P must require only $O(n)$ memory for storage; (iii) matrix–vector products with P must be computed in $O(n)$ operations; (iv) these matrix–vector products must be parallelizable. In addition, the reduced system matrix $P_2^T A P_2$ must be well conditioned so that the iterative method converges rapidly. Such a P is said to *implicitly* precondition the reduced system.

3.1. A hierarchical scheme for computing the decomposition

In this section, we describe a hierarchical technique to compute the matrix P with the desirable properties listed above. For brevity, we provide only an outline of the multilevel algorithm in the context of a simple 4×4 mesh. More details are available in [11,12].

The structure of the discrete gradient operator matrix B is exploited to obtain a representation of B on a hierarchy of meshes. Linear transformations are used to convert B to a sequence of gradient operators on each of the coarser meshes. The submatrix P_2 turns out to be a well-conditioned basis for $\mathcal{N}(B^T)$, and preconditions the reduced system for the case when A is well conditioned as well. Although P is a dense matrix, we can represent it as a sequence of k sparse matrices, where k is the number of levels in the hierarchy of coarse meshes. This representation allows storage of P in $O(n)$ space and computation of matrix–vector products with P in $O(n)$ steps.

The rows and columns of the discrete gradient operator matrix B correspond to the edges and nodes, respectively, of the underlying mesh. A coarse level mesh can be obtained by combining adjacent nodes into a single node and coalescing edges between the same pair of coarse level nodes. This coarsening step is represented as a linear transformation on B using matrices $P^{(1)}$ and $Z^{(1)}$ such that

$$P^{(1)T} B Z^{(1)} = \begin{bmatrix} D^{(1)} & 0 \\ 0 & B^{(1)} \\ 0 & 0 \end{bmatrix}, \tag{5}$$

where $B^{(1)}$ is analogous to the discrete gradient operator on a coarser mesh. Part of $P^{(1)}$ as well as the matrices $Z^{(1)}$ and $D^{(1)}$ are obtained from the SVD of individual subdomain matrices, where each subdomain corresponds to a group of nodes that combine to form a coarse level node. Subsequent transformations to obtain coarser meshes do not affect any row or column other than those in the second block row and second block column containing $B^{(1)}$.

As an example, let us consider the Poisson equation with Neumann boundary conditions on a two-dimensional unit square Ω

$$-\nabla \cdot \nabla u = f \text{ in } \Omega, \quad \frac{\partial u}{\partial n} = 0 \quad \text{on } \partial\Omega.$$

To derive a saddle-point problem, we introduce the gradient as an unknown function,

$$\mathbf{v} + \nabla u = 0, \quad \nabla \cdot \mathbf{v} = f.$$

For simplicity, we use a uniform grid to discretize the domain. The unknowns for u are computed at the nodes of the mesh. The unknowns for $\mathbf{v}$ are computed at the midpoint of the edges, and are oriented along the edges. The resulting linear system is a saddle-point problem with the following form

$$\begin{bmatrix} I & B \\ B^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}.$$

Fig. 1 shows the coarsening of a 4×4 uniform mesh. The nodes are clustered into four groups: $G_1 = \{1,2,5,6\}$, $G_2 = \{3,4,7,8\}$, $G_3 = \{9,10,13,14\}$, and $G_4 = \{11, 12,15,16\}$. These groups define partitions or subdomains. Solid edges lie in the interior of these partitions, while dashed edges go across partitions. Each partition is coarsened into a single node at the next level. The nodes colored grey in the fine mesh are retained in the coarse mesh. The edges between two groups G_i and G_j are collected into a single edge between nodes i and j in the coarse mesh.

To compute the transformation from the 4×4 mesh to the 2×2 mesh, we reorder the nodes according to the groups assigning a natural ordering within a group. The edges are ordered as follows: interior edges are ordered first, according to the groups, followed by edges between partitions. All edges between a particular pair of partitions are numbered consecutively. With this new ordering, the gradient matrix B has the following form:

$$B = \begin{bmatrix} B_{\text{interior}} \\ B_{\text{exterior}} \end{bmatrix},$$

where

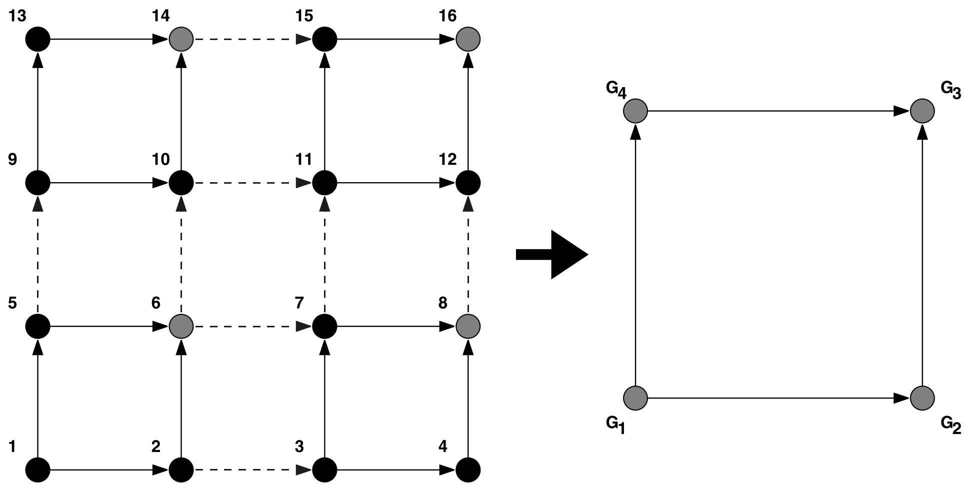

Fig. 1. The coarsening of a 4×4 mesh to a 2×2 mesh.

$$B_{\text{interior}} = \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & B_3 & \\ & & & B_4 \end{bmatrix}, \quad B_{\text{exterior}} = \begin{bmatrix} -C_1 & C_2 & & \\ & & -C_1 & C_2 \\ -C_3 & & C_4 & \\ & -C_3 & & C_4 \end{bmatrix},$$

in which

$$B_i = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, \quad i = 1, \ldots, 4$$

and

$$C_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad C_3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$
$$C_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Since B_{interior} is a block diagonal matrix with four identical blocks, the SVD matrices in the decomposition $B_{\text{interior}} = USV^{\text{T}}$ are also block diagonal, with the ith block having the form

$$U_i = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad S_i = \begin{bmatrix} 2 & & & \\ & \sqrt{2} & & \\ & & \sqrt{2} & \\ & & & 0 \end{bmatrix},$$
$$V_i = \frac{1}{2}\begin{bmatrix} 1 & -\sqrt{2} & 0 & 1 \\ -1 & 0 & -\sqrt{2} & 1 \\ -1 & 0 & \sqrt{2} & 1 \\ 1 & \sqrt{2} & 0 & 1 \end{bmatrix}.$$

The transformation of B into $B^{(1)}$ is accomplished by multiplying B with appropriate matrices as shown below:

$$\begin{bmatrix} I & \\ -B_{\text{exterior}}VS^{\dagger} & I \end{bmatrix} \begin{bmatrix} U^{\text{T}} & \\ & I \end{bmatrix} \begin{bmatrix} B_{\text{interior}} \\ B_{\text{exterior}} \end{bmatrix} V = \begin{bmatrix} S \\ B_{\text{exterior}}V(I - S^{\dagger}S) \end{bmatrix}.$$

The matrix $I - S^{\dagger}S$ is a zero matrix except for ones on the diagonal in those positions that are zero in S. Therefore, $B_{\text{exterior}}V(I - S^{\dagger}S)$ is a matrix with nonzeros in exactly those four columns that have a zero singular value in S. The structure of $B_{\text{exterior}}V(I - S^{\dagger}S)$ is given as

$$B_{\text{exterior}}V(I - S^{\dagger}S) = \frac{1}{2}\begin{bmatrix} -F & F & 0 & 0 \\ 0 & 0 & -F & F \\ -F & 0 & F & 0 \\ 0 & -F & 0 & F \end{bmatrix}, \quad F = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Multiplying the above matrix with the orthogonal matrix

$$Q^{\mathrm{T}} = \begin{bmatrix} Q_1 & & & \\ & Q_2 & & \\ & & Q_3 & \\ & & & Q_4 \end{bmatrix}, \quad Q_i = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad i = 1, \ldots, 4,$$

we get a matrix whose nonzero rows and columns are the coarse grid gradient operator for the 2×2 mesh

$$B^{(1)} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}.$$

Reordering of rows to move the zero rows to the bottom gives the required transformation (5) in which

$$P^{(1)\mathrm{T}} = \begin{bmatrix} I & \\ & Q^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} I & \\ & -B_{\text{exterior}} VS^{\dagger} I \end{bmatrix} \begin{bmatrix} U^{\mathrm{T}} & \\ & I \end{bmatrix}, \quad Z^{(1)} = V,$$

and $D^{(1)}$ is identical to S with the zero columns removed.

A larger mesh is coarsened by repeated application of this process. Recursive coarsening of the mesh to a single node gives a sequence of transformations that collectively represent the hierarchical decomposition of B. Thus, after k levels of coarsening, where $k = \mathrm{O}(\log n)$ for typical meshes, we have

$$P = P^{(1)} P^{(2)} \cdots P^{(k)}, \quad Z = Z^{(1)} Z^{(2)} \cdots Z^{(k)}, \tag{6}$$

such that the transformation at the jth level is given by

$$\tilde{P}^{(j)\mathrm{T}} B^{(j-1)} \tilde{Z}^{(j)} = \begin{bmatrix} D^{(j)} & 0 \\ 0 & B^{(j)} \\ 0 & 0 \end{bmatrix}, \quad j = 1, \ldots, k.$$

The matrices $P^{(j)}$ and $Z^{(j)}$ equal the identity except for diagonal blocks with $\tilde{P}^{(j)}$ and $\tilde{Z}^{(j)}$, respectively, that act on $B^{(j-1)}$.

We mention that the above technique extends easily to gradient matrices obtained from finite element method on unstructured meshes. The details are available in [11,12].

3.2. Parallelism in the multilevel algorithm

The hierarchical representation of P generated by the multilevel algorithm allows an efficient parallel formulation. The matrix P is never explicitly computed or stored; it is available as a product of a sequence of sparse matrices associated with each level in the hierarchy. At each iteration, a matrix–vector product is computed by multiplying the given vector with the set of matrices $P^{(j)}$, $j = 1, \ldots, k$, that represent P. The structure of this computation is similar to the product of a sequence of Laplace matrices defined on a hierarchy of meshes.

The parallelization of a matrix–vector product is quite straightforward. For a multiprocessor with p processors, the domain is partitioned into p subdomains. The unknown variables associated with the ith subdomain are allocated to the ith processor. A processor is responsible for computing the product with the *local* submatrix that resides with the processor. A matrix–vector product is computed in three phases: the first phase consists of local matrix–vector products. In the second phase, data are communicated between processors owning adjacent subdomains. Finally, the communicated data are processed by the receiving processor. (See, e.g. [9] for a description of parallel matrix–vector products.)

Overhead in the form of communication during the second phase causes loss of parallel efficiency. However, there is no way to avoid it since the connectivity of the mesh across subdomains forces communication of data between processors. Fortunately, high parallel efficiency can be obtained if the communication to computation ratio is relatively small. This is indeed the case when using moderate number of processors to solve a large problem.

Since the matrices $P^{(j)}$ are defined on a hierarchy of meshes of exponentially reducing size, parallel efficiency diminishes for coarse level matrix–vector products. Fortunately, the communication overhead for the finest mesh is comparable to the total communication overhead for the remaining levels in the hierarchy. Therefore, the overall parallel efficiency is only a slight reduction from that of a single matrix–vector product with the finest level matrix. The efficiency can be restored further by "coarsening" the processor grid along with coarsening the mesh (for details, see [11]).

The execution time for a matrix–vector product with an $n \times n$ matrix P on a parallel computer with p processors is given by

$$T(n,p) = t_{\mathrm{c}}\left[\frac{n}{p} + \log p\right] + t_{\mathrm{s}} \log n + t_{\mathrm{b}}\sqrt{\frac{n}{p}} + t_{\mathrm{h}}\sqrt{p},$$

where t_{c} is the unit computation time, t_{s} is communication startup time, t_{b} is byte transfer time, and t_{h} is the per-hop time. The speed improvement over a uniprocessor matrix–vector product can be written as

$$S(n,p) = \frac{T(n,1)}{T(n,p)} = p\left[1 + \frac{p \log n}{n} \times \frac{t_{\mathrm{s}}}{t_{\mathrm{c}}} + \sqrt{\frac{p}{n}} \times \frac{t_{\mathrm{b}}}{t_{\mathrm{c}}} + \frac{p\sqrt{p}}{n} \times \frac{t_{\mathrm{h}}}{t_{\mathrm{c}}}\right]^{-1}.$$

The above analysis uses a linear cost model to estimate parallel performance (see, e.g. [6]). For a fixed number of processors, increase in the problem size yields proportionally higher speedup. Moreover, for fixed efficiency or speedup, the problem size needs to be increased proportionally with the number of processors, suggesting good scalability. The algorithm also has a favorable ratio of computation to data transfer that leads to good efficiency.

4. Particulate flow simulations

4.1. Multilevel algorithm for particulate flows

The multilevel algorithm has been applied successfully to the simulation of rigid particles in viscous incompressible flows [5]. Along with Navier–Stokes equations for fluid, we need to solve equations for Newton's laws for the particles

$$\mathbf{M}\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}t} = \mathbf{F}, \quad \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \mathbf{U}, \quad \mathbf{u} = \mathbf{U} + \mathbf{r} \times \Omega, \tag{7}$$

where $\mathbf{X}$ and $\mathbf{U}$ are the generalized position and velocity vectors of the particles, respectively, that include both translational and angular components, $\mathbf{M}$ is the generalized mass matrix, and $\mathbf{F}$ is the vector of forces and torques acting on the particles. The last equation couples the fluid velocity at the surface of a particle with the particle velocity by imposing a no-slip condition. In this equation, $\mathbf{r}$ denotes the vector from the center of the particle to a point on its surface, and Ω denotes the angular velocity of the particle.

To understand the structure of the linear system in particulate flows, let us consider two matrices: the first, denoted by J, represents the fluid and particle systems decoupled from each other, and the second, denoted by $\hat{Q}$, provides the coupling between the fluid and particles at the particle surface

$$J = \begin{bmatrix} A_I & A_{I\Gamma} & B_I & 0 \\ A_{\Gamma I} & A_\Gamma & B_\Gamma & 0 \\ B_I^{\mathrm{T}} & B_\Gamma^{\mathrm{T}} & 0 & 0 \\ 0 & 0 & 0 & A_{\mathrm{p}} \end{bmatrix}, \qquad \hat{Q} = \begin{bmatrix} I & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & I \\ 0 & I & 0 \end{bmatrix}.$$

Here, the subscripts I and Γ represent the submatrices for the fluid interior and the particle surface, respectively, and A_{p} represents the particle matrix. The fully coupled system matrix is

$$\tilde{J} = \hat{Q}^{\mathrm{T}} J \hat{Q} = \left[\begin{array}{cc|c} A_I & A_{I\Gamma}Q & B_I \\ Q^{\mathrm{T}}A_{\Gamma I} & Q^{\mathrm{T}}A_\Gamma Q + A_{\mathrm{p}} & Q^{\mathrm{T}}B_\Gamma \\ \hline B_I^{\mathrm{T}} & B_\Gamma^{\mathrm{T}}Q & 0 \end{array}\right],$$

and the corresponding linear system has the form

$$\left[\begin{array}{cc|c} A_I & \tilde{A}_{I\Gamma} & B_I \\ \tilde{A}_{\Gamma I} & \tilde{A}_\Gamma + A_{\mathrm{p}} & \tilde{B}_\Gamma \\ \hline B_I^{\mathrm{T}} & \tilde{B}_\Gamma^{\mathrm{T}} & 0 \end{array}\right] \begin{bmatrix} u_I \\ U \\ p \end{bmatrix} = \begin{bmatrix} f_{\mathrm{I}} \\ \tilde{f}_{\mathrm{p}} \\ 0 \end{bmatrix}. \tag{8}$$

This is also a saddle-point problem with a structure identical to (1). Since the matrix $[B_I^{\mathrm{T}}, B_\Gamma^{\mathrm{T}}Q]$ is similar to the discrete divergence matrix B^{T}, the decomposition (3) can be computed as discussed earlier. Thus, the solution process remains unchanged in the presence of particles.

Table 1
Performance of the multilevel algorithm for 1920 sedimenting particles

Linear system solver stage	$p = 4$	$p = 16$	
	Time	Time	T_4/T_{16}
Assembly	65	17	3.8
Preconditioning	903	234	3.9
Matvec	186	49	3.8
Orthog. (GMRES)	23	8	2.9
Total	1177	308	3.8

We conducted simulations to mimic the sedimentation of particles in water in a vertical channel. The simulations were conducted for a two-dimensional channel with particles that were two-dimensional circular disks with diameter 0.25 in. and specific gravity 1.14. All the particles were arranged in an array at the top of the channel before being released.

Table 1 shows the performance of the multilevel algorithm for 1920 sedimenting particles on 16 processors of the SGI Origin 2000. We have reported the average time taken for critical steps in the simulation process, namely, *assembly* of the linear system using the finite element method, *preconditioning* via the hierarchical basis for divergence-free velocity including computation of P and matrix–vector products with P, *matrix–vector product* with A, and *orthogonalization* with respect to the Krylov subspace in GMRES. The size of a typical linear system was 250,472 with 4,784,928 nonzeros. The 16-processor performance is compared to that on 4 processors.

Table 2 shows the performance of the multilevel algorithm for 3840 sedimenting particles on 32 processors of the SGI Origin 2000. The size of a typical linear system was 501,120 with 9,566,808 nonzeros. In this case, the 32-processor performance is compared to that on 4 processors.

The experiments indicate that the important steps in the simulation are scalable and efficient even on a moderate sized problem. The most time-consuming step is

Table 2
Performance of the multilevel algorithm for 3840 sedimenting particles

Linear system solver stage	$p = 4$	$p = 32$	
	Time	Time	T_4/T_{32}
Assembly	105	15	7.0
Preconditioning	3842	362	10.6
Matvec	98	14	7.0
Orthog. (GMRES)	50	13	3.8
Total	4095	404	10.1

the preconditioning step which demonstrates superlinear speedup. Preconditioning time is mainly spent in computing SVD of small matrices and computing dense matrix–vector products with small matrices. These are inherently cache-friendly operations that improve the performance dramatically as the processors are increased. This superlinear speedup is seen despite the fact that the effectiveness of the preconditioner used in the present code degrades as the number of processors are increased. In principle, one can use the *same* preconditioner on multiple processors without any increase in iterations, but with marginal decrease in efficiency (see, e.g. [11] for a scalable implementation of the preconditioner). The decrease in parallel performance is also an artifact of the partitioning strategy which aims at good load balance at the expense of preconditioning. The performance of the orthogonalization step in GMRES may be further improved by using a more efficient scheme.

It should be noted that these experiments define *speedup* as the improvement in speed over the *best* implementation of the algorithm on 4 processors. This implies that although the parallel algorithm demonstrates good *speed improvement* on multiple processors, the speedup may be modest because of a less effective preconditioner that slows down the convergence. We believe that this is the most realistic metric for any parallel algorithm.

4.2. A Schur-complement algorithm

An alternate approach for (8) solves the Schur-complement via an iterative method. The matrix is reordered to get the following system

$$\left[\begin{array}{cc|c} A_I & B_I & \tilde{A}_{I\Gamma} \\ B_I^{\mathrm{T}} & 0 & \tilde{B}_\Gamma^{\mathrm{T}} \\ \hline \tilde{A}_{\Gamma I} & \tilde{B}_\Gamma & \tilde{A}_\Gamma + A_{\mathrm{p}} \end{array}\right] \begin{bmatrix} u_I \\ p \\ U \end{bmatrix} = \begin{bmatrix} f_1 \\ 0 \\ \tilde{f}_{\mathrm{p}} \end{bmatrix}. \tag{9}$$

Next, we reorder the velocity unknowns u_I and pressure unknowns p such that velocity and pressure unknowns for each node are ordered consecutively. The reordered matrix is expressed in a block form

$$\begin{bmatrix} C & E \\ F^{\mathrm{T}} & G \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix},$$

where C corresponds to the saddle-point matrix in the upper left corner of the system (9). When a bandwidth reducing ordering is used, C takes the form of a narrow banded matrix. The bandwidth depends on a number of factors including the dimension of the problem, number of particles, etc. In the particulate flow experiments, C is a narrow banded matrix, giving the appearance of an arrow matrix to the above system.

From this point onwards, we can follow two strategies depending upon the number of unknowns for the particles, i.e., the size of G. When G is small compared to C, we solve the Schur-complement system for y

$$[G - F^{\mathrm{T}}C^{-1}E]y = [b - F^{\mathrm{T}}C^{-1}a].$$

Since E has small number of columns as well, we solve the system $CZ = E$ with multiple right-hand sides to obtain Z. The Schur-complement system is formed explicitly by computing $G - F^{\mathrm{T}}Z$, and solved via a direct or iterative method. Once y is known, we solve the following system for x

$$Cx = [a - Ey].$$

The situation is reversed if the number of particle unknowns is large. In this case, we solve the Schur-complement system

$$[C - EG^{-1}F^{\mathrm{T}}]x = [a - EG^{-1}a]$$

to get x, and then solve

$$Gy = [b - F^{\mathrm{T}}x]$$

to recover y.

The Schur-complement systems are solved via an iterative method such as GMRES. The most expensive step is the solution of a saddle-point problem of the form $Cw = b$ at each iteration. This approach is feasible only if one has a fast parallel method to solve these saddle-point problems.

We can use a simple scheme to partition the rows of an $n \times n$ narrow banded matrix into two blocks. These two blocks represent two underdetermined systems of equations that can be solved concurrently. To obtain the overall solution, however, the unknowns common to these blocks must be evaluated to the same value by both underdetermined systems. This gives rise to a reduced system that must be solved to obtain a consistent solution to the two systems. Such a solution is also the solution of the overall system. Details of this approach are available in [3].

In general, one can partition the matrix into p blocks, and solve the corresponding underdetermined systems. A reduced system is generated implicitly, defined only on unknowns common to consecutive block rows, and is solved by an iterative method in which matrix–vector products are computed directly from appropriate projections. The p-way parallelism at the block level can be exploited to get high efficiency on a parallel architecture. Although p is limited by the bandwidth of the system, one can use multiple processors for each underdetermined system and the associated projection to further increase parallelism. The resulting algorithm is a scalable parallel solver for the saddle-point problems.

Table 3 illustrates the performance of this approach for the solution of the saddle-point problem (1). The matrix A is symmetric positive definite. The saddle-point system is symmetric because $C = B$. The Schur-complement system

$$B^{\mathrm{T}}A^{-1}Bp = g - B^{\mathrm{T}}A^{-1}f$$

was solved by the CG method which was preconditioned with incomplete Cholesky factorization of $BD^{-1}B^{\mathrm{T}}$, where $D = \mathrm{diag}(A)$. At each iteration, our scheme was used to solve linear systems of the form $Ax = b$. The projections were computed in parallel using preconditioned CG algorithm as well. The first part of Table 3 shows the performance of our scheme on a typical instance of the system $Ax = b$. In the second part, we present results for the solution of the Schur-complement system.

Table 3
Performance of the Schur-complement approach for solving the particulate flow linear systems [10]

Linear system		$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
$Ax = b$	Time	9.1	4.0	2.0	1.2	0.9
	Speed improv.		2.3	4.6	7.6	10.1
	Efficiency		1.1	1.1	0.9	0.6
$B^{T}A^{-1}Bp = g$	Time	109.4	55.4	28.9	15.1	12.3
	Speed improv.		1.9	3.8	7.3	8.9
	Efficiency		1.0	0.9	0.9	0.6

The matrix A is of size 17,025 with bandwidth 1450 and 362,641 nonzeros whereas the matrix $B^{T}A^{-1}B$ is of size 2199. The rows of A were partitioned into 8 blocks resulting in a reduced system of size 3680. These experiments show almost linear speed improvement, with isolated superlinear performance due to cache effects. The 16-processor case partitioned the rows into 16 blocks resulting in a reduced system of size 5508. The performance degradation is explained by the fact that increase in processors without proportional increase in the problem size causes decrease in parallel performance. Since additional parallelism is available in computing individual projections, parallel speedup can be improved substantially by assigning multiple processors to each block row. In other words, one may use 16 processors for a matrix partitioned into 8 block rows by assigning two processors to each block. This is clearly more advantageous for cluster-based multiprocessors.

5. Concluding remarks

In this paper, we have discussed two algorithms for the solution of indefinite saddle-point problems arising in incompressible fluid simulations. These techniques have been successfully applied to the simulation of particulate flows in which a coupled system of equations is solved for the fluid and particle system. The first algorithm uses a hierarchical basis for divergence-free velocity to satisfy the incompressibility constraints that are the cause of indefiniteness. The resulting reduced system is solved by suitable iterative methods such as CG or GMRES. Appropriate selection of the basis can act as a preconditioner that accelerates the convergence of the iterative scheme. The preconditioning action of the well-conditioned hierarchical basis is very effective for simulations with small time-steps in which the saddle-point systems have a diagonally dominant nonzero diagonal block. The second algorithm circumvents the indefiniteness of the saddle-point system by solving the reduced system known as the Schur-complement system that is defined either on pressure unknowns or the particle unknowns. Both these algorithms have been parallelized for shared-memory architectures, and have demonstrated good parallel efficiency on modest number of processors. Analysis of the parallel formulation indicates that the performance of these algorithms will scale for larger number of processors when the problem size is increased proportionally.

References

[1] O. Axelsson, Iterative Solution Methods, Cambridge University Press, Cambridge, 1994.
[2] R. Bank, B. Welfert, H. Yserentant, A class of iterative methods for solving saddle point problems, Numer. Math. 56 (1990) 645–666.
[3] G.H. Golub, A.H. Sameh, V. Sarin, A parallel balance method for sparse linear systems, Numer. Linear Algebra Appl. 8 (2001) 297–316.
[4] K. Gustafson, R. Hartman, Divergence-free bases for finite element schemes in hydrodynamics, SIAM J. Numer. Anal. 20 (4) (1983) 697–721.
[5] M.G. Knepley, V. Sarin, A.H. Sameh, Parallel simulation of particulate flows, in: Lecture Notes in Computer Science, vol. 1457, Springer, Berlin, 1998, pp. 226–237.
[6] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin Cummings, Menlo Park, CA, 1993.
[7] R. Peyret, T. Taylor, Computational Methods for Fluid Flows, Springer Series in Comp. Physics, Springer, Berlin, 1983.
[8] T. Rusten, R. Winther, A preconditioned iterative method for saddle-point problems, SIAM J. Matrix Anal. Appl. 13 (3) (1992) 887–904.
[9] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS, Boston, 1996.
[10] A.H. Sameh, V. Sarin, Hybrid parallel linear system solvers, Int. J. Comp. Fluid Dyn. 12 (1999) 213–223.
[11] V. Sarin, Efficient iterative methods for saddle point problems, Ph.D. Thesis, University of Illinois, Urbana-Champaign, 1997.
[12] V. Sarin, A.H. Sameh, An efficient iterative method for the generalized Stokes problem, SIAM J. Sci. Comput. 19 (1) (1998) 206–226.
[13] D. Silvester, A. Wathen, Fast iterative solution of stabilised Stokes systems. Part 2: Using general block preconditioners, SIAM J. Numer. Anal. 31 (5) (1994) 1352–1367.
[14] A. Wathen, D. Silvester, Fast iterative solution of stabilised Stokes systems. Part 1: Using simple diagonal preconditioners, SIAM J. Numer. Anal. 30 (3) (1993) 630–649.
[15] X. Ye, C.A. Hall, A discrete divergence-free basis for finite element methods, Numer. Algorithms 16 (1997) 365–380.