A High Performance Algorithm for Incompressible Flows using Local Solenoidal Functions[*]

Sreekanth R. Sambavaram and Vivek Sarin
Department of Computer Science
Texas A&M University
College Station, TX 77843
ssreddy@cs.tamu.edu
sarin@cs.tamu.edu

Abstract

The convergence of iterative methods used to solve the linear systems arising in incompressible flow problems is sensitive to flow parameters such as the Reynolds number, time step and the mesh width. This paper presents a class of algorithms to solve these linear systems using local solenoidal functions. An optimal preconditioner is described via an iterative method to solve the resulting reduced system. This paper also suggests inexpensive parallel matrix-vector products using bounded buffers for interprocessor communication. Experimental results for a three dimensional problem show that the preconditioning step need not be solved accurately at each iteration, thereby decreasing the time spent in the potentially expensive routine. These experiments also show that the proposed algorithm assures a constant rate of convergence across the range of flow parameter variation. Scalability of the algorithm is suggested by the experiments on the SGI Origin 2000.

Keywords. linear system, iterative methods, preconditioners, divergence-free, computational fluid dynamics, parallel performance.

1. Introduction

The physical phenomenon of fluid flow is one of the oldest and highly researched topics. Advances in computing provided a new dimension to the research by enabling the numerical simulation of flow problems. Realistic simulation of flow requires considerable amount of computational power such as that offered by parallel processors. This reinforces the need for parallel algorithms that can be implemented efficiently on these parallel architectures. The principles of classical mechanics, thermodynamics, and laws of conservation of mass, momentum, and energy govern the motion of fluid. Law of conservation of momentum for incompressible, viscous flow in a region Ω with boundary $\partial\Omega$ is captured by Navier-Stokes equation given by

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{R}\Delta \mathbf{u}, \qquad (1)$$

where $p = p(\mathbf{x}, t)$ is the pressure, R is the Reynolds number, and $u = u(\mathbf{x}, t)$ is the velocity vector at $\mathbf{x}$. The law of conservation of mass for incompressible fluids gives rise to

$$\nabla \cdot \mathbf{u} = 0 \ \text{ in } \Omega. \qquad (2)$$

Viscosity of the fluid imposes a boundary condition at the walls given by

$$\mathbf{u} = 0 \ \text{ on } \partial\Omega. \qquad (3)$$

Suitable discretization and linearization of the equations (1-3) result in a linear system given by

$$\begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}. \qquad (4)$$

where B^T is the discrete divergence operator and A is the discrete counterpart of terms with $\mathbf{u}$ in (1):

$$A = \frac{1}{\Delta t}M + C + \frac{1}{R}L,$$

in which M is the mass matrix, L is the Laplace matrix, and C is the matrix arising from the convection term. When operator splitting is used to separate the linear and non-linear terms, we obtain the generalized Stokes problem (GSP) with a symmetric positive definite A given by

$$A = \frac{1}{\Delta t}M + \frac{1}{R}L.$$

[*]This work has been supported in part by NSF under the grants NSF-CCR 9984400, NSF-CCR 9972533, and NSF-CTS 9873236. Computational resources were provided by NCSA, University of Illinois at Urbana-Champaign.

The linear system (4) obtained for a given flow problem was typically solved by direct methods. Solving any practical flow problem involves a large sparse linear system of equations with many unknowns, which requires a large amount of memory. As a result, direct methods are prohibitive in a single processor environment. These shortcomings have paved a way for iterative methods. Although iterative methods are well suited for parallel processing, they may not be as reliable as direct methods. In order to make iterative methods more effective, the system of linear equations is modified to a system that is easier to solve. Preconditioning is a technique to transform the original system into one that is solved faster by iterative methods.

In general, the linear system (4) is large, sparse, and indefinite due to the incompressibility constraints $B^T u = 0$. The indefiniteness of the matrix is the main cause of difficulty in developing preconditioned iterative solvers. The degree of difficulty also depends upon the nature of A which is affected by the Reynolds number R and the choice of time step Δt. Developing effective preconditioners and iterative methods for such systems is a challenging task.

In this paper, we present a scheme to restrict the velocity to divergence-free subspace and outline preconditioners for the reduced system that must be solved iteratively. The preconditioners are optimal in that convergence is assured in fixed number of iterations. We also outline a parallel computational scheme for the solution methodology. Experiments on SGI Origin 2000 suggest that proposed scheme is efficient for solving 3D problems.

2 An Algorithm Using Solenoidal Basis Functions

2.1 A Discrete Divergence-free Approach

In order to simplify the linear system (4), velocity is expressed in terms of divergence-free velocity functions. A solenoidal basis is a basis for divergence-free velocity. A method to construct hierarchical solenoidal basis is given in [10], and its application to 2D particulate flow problems in which matrix A is diagonally dominant is described in [4, 6, 11]. The local divergence-free basis introduced in [9] allows construction of very effective preconditioners for more general instances of A.

A discrete divergence-free basis for fluid velocity has the advantage of automatically satisfying the continuity constraint in the Navier-Stokes equations for incompressible fluids. Hence, flow expressed in terms of divergence-free velocity functions is incompressible by definition. In matrix terms, this conversion of velocities into divergence-free velocity functions is called a projection, and is achieved by a projection matrix P. A matrix P that satisfies the condition $B^T P = 0$ can be used to represent divergence-free space. Velocity defined as $u = Px$ satisfies $B^T Px = 0$, which implies $B^T u = 0$, and the linear system (4) becomes

$$APx + Bp = f. \tag{5}$$

Since, $B^T P = 0$, this system is simplified by multiplying with P^T to eliminate pressure:

$$P^T APx = P^T f. \tag{6}$$

This reduced system can be solved using any suitable iterative scheme like Conjugate Gradients (CG) or GMRES. An appropriate choice of the projection matrix P implicitly preconditions the reduced system and accelerates the convergence of the iterative method. Once x has been calculated, velocity can be computed as

$$u = Px, \tag{7}$$

and pressure is recovered by solving the least squares problem

$$Bp \approx f - APx \tag{8}$$

2.2 Local Solenoidal Functions

A number of issues must be resolved before this approach can be viable. In particular, given that B is large and sparse, (a) the null space basis P must be computed and stored efficiently, (b) the structure of P must allow fast computation of matrix-vector products with the reduced system $P^T AP$, (c) the algorithms for computing and applying P must be efficiently parallelized, and (d) the choice of P must allow effective preconditioning of the reduced system $P^T AP$ so that the iterative method converges rapidly.

A purely algebraic approach such as QR factorization of B cannot be used to compute P due to the prohibitive cost of computation and storage. One can exploit the nature of the problem to develop techniques to compute P with desirable properties. In the continuum space, one can define local solenoidal functions that are circulating flows or vortices at each point in the domain. In the discrete setting, edges and elements of a mesh can be used to define similar circulating flows that are discreetly solenoidal.

We proposed the use of local solenoidal functions for 2D flows in [9], where we presented a scheme to construct a solenoidal basis derived from circulating flows or vortices, and outlined an optimal preconditioning technique for the generalized Stokes problem in [8]. This paper gives an effective parallelization scheme for solving the linear system using preconditioned solenoidal basis methods.

Each such flow is represented as a vector, and the set of these vectors forms the columns of P. The matrix P is sparse due to the localized structure of the discrete solenoidal functions. In fact, P is never actually computed in solving the reduced system, but its localized nature is used to calculate the matrix-vector products involving P directly. This results in a matrix-free implementation of the algorithm which is memory efficient.

2.3 Preconditioning the Reduced System

Effective preconditioning of the reduced system is critical to the overall success of the solenoidal basis method. Since the matrix P^TAP is not computed explicitly, the design of a preconditioner is challenging. Observing that the product Px and P^Ty compute the discrete curl of the functions represented by $\mathbf{x}$ and $\mathbf{y}$, respectively, it can be inferred that the product $y = P^TPw$ represents $\nabla\times\nabla\times$ w in a discrete setting. Thus, the matrix P^TP can be shown to be equivalent to the Laplace operator on the solenoidal function space. Based on this insight, the preconditioner for the generalized Stokes problem is defined as follows:

$$G = \left[\frac{1}{\Delta t}M + \frac{1}{R}L_s\right]L_s,$$

where L_s is the Laplace operator for the local solenoidal functions. Experiments presented in section 4 show that the preconditioner is very effective over a large range of Δt and R values. Since the preconditioned system is spectrally equivalent to a symmetric positive definite matrix, one can use preconditioned CG to solve the reduced system (6).

3 Parallel Formulation

3.1 Mesh Partitioning

The main components of the algorithm that must be parallelized are (a) computation of the solenoidal basis matrix P, (b) matrix-vector products with P and P^T, (c) vector addition, inner-products and matrix-vector products with A, and (d) the preconditioning step. Parallelization of these operations require the underlying mesh to be distributed across processors. Since, the computation involved in the main components of the algorithm is a function of sub-mesh size on each processor, a load balancing scheme should partition the underlying mesh equally across the processors. On a p processor machine, the mesh is divided into p partitions and the jth partition is given to processor j. The unknowns for velocity, vorticity, and velocity potential defined on the nodes, edges or elements within a partition j reside on processor j that owns the partition. Parallel matrix-vector products require inter-processor communication between processors with neighboring sub-meshes which degrades parallel efficiency.

One can partition the physical mesh equally across processors in several ways. But, the amount of inter-processor communication needed in computing the matrix-vector products varies largely on the mesh partitioning scheme. For uniform grids, partitioning the grid across any one dimension is the simplest scheme. One can achieve this by a simple loop parallelization scheme by distributing the loop control index which corresponds to the dimension along which the grid is partitioned. The resulting communication is $O(n^{1-1/d})$ for a d-dimensional uniform grid with $n^{1/d}$ nodes along each dimension.For a 3D uniform grid with $n^{1/3}$ nodes along each dimension, the inter-processor communication is $O(n^{2/3})$. Hence, the scheme is prohibitive for realistic problems. An optimal scheme for such a grid is obtained by partitioning the grid across all the dimensions equally.

For unstructured grids, one can use a heuristic to partition the mesh to minimize communication and balance the workload. Partitioning of irregular meshes is described in [2, 3]. Several such techniques and software are available. On the other hand, one can partition the problem domain into p partitions, and generate sub-meshes in each partition in parallel. Care must be taken while generating sub-meshes to minimize communication and balance the work on each processor.

3.2 Parallel Matrix-Vector Products

Since the reduced system (6) is never computed explicitly, the product $y = P^TAPx$ is computed as a sequence of three matrix-vector products: $y = Px$, $z = Ay$ and $w = P^Tz$. The choice of local solenoidal functions leads to a sparse P with non-zero pattern similar to the adjacency matrix of the mesh. The matrix-vector products Px and P^Tz can be computed concurrently in a manner similar to that of a matrix-vector product with a Laplace matrix (L_s). Products with A are equally straightforward to parallelize.

The preconditioning step consists of two systems that need to be solved in succession. The system matrices are the Laplace matrix and a convection-diffusion matrix. These linear systems can be solved via direct or iterative methods with the possible use of fast Poisson solvers, domain decomposition [1], multigrid, and multilevel methods. Parallelism in such algorithms is well understood [7]. In this paper the preconditioning step is computed using two successive calls to CG routines.

3.3 Inter-Processor Communication

Several techniques can be used for computing matrix-vector products of sparse matrices arising from the Laplace operator in [7, 5]. Assuming the mesh is partitioned as described previously, a processor is responsible for computing the product with the local sub- matrix that resides with the processor. Matrix-vector product computation for a given node needs values from its neighboring nodes and a processor computes matrix-vector product for all nodes which lie in its local partition. For a boundary node the processor needs nodal values which do not reside in its local memory. To reduce communication overhead, all the values needed by a neighboring processor j are communicated in blocks. This communication precedes the actual computation of the matrix-vector product.

Given any two neighboring processor grids i and j, nodal values of i are needed for matrix-vector product computation in processor j and vice versa. A bounded buffer S_{ij} is created in processor i's memory and all the nodal values residing in i which are adjacent to the boundary nodes of j are stored in it. Similarly, S_{ji} is constructed in processor j's memory. Another bounded buffer R_{ji} is created in processor j's memory which has the same size as S_{ij}. The communication step consists of copying S_{ij} into R_{ji} as a single abstract block. After the communication, all the values which reside in physical grid i and neighboring the boundary nodes in grid j are in local memory of processor j. All the processors do the communication using bounded buffers simultaneously before starting matrix-vector products on their local grid.

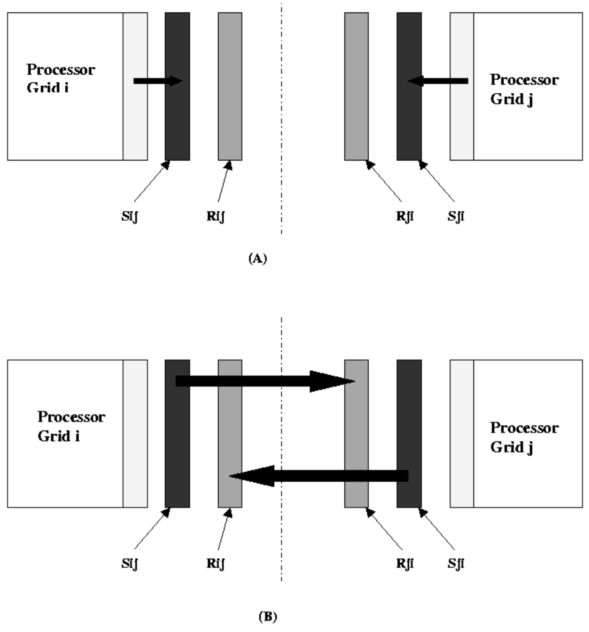

Figure 1. Inter-processor communication is performed in two steps: (a) Every processor copies its local nodal values needed by the neighboring processor into a communication buffer S (b) the nodal values from S in a neighboring processor are copied into a processor's local buffer R.

Figure 1 illustrates the steps involved in the inter processor communication. Thus, matrix-vector product is computed in two stages: the first stage consists of communication using buffers and in the second stage, matrix-vector product is computed by the processors for their local grids.
Overhead in the form of communication during the first stage causes loss of parallel efficiency. There is no way to avoid it since the connectivity of the mesh across subdomains forces communication of data between processors. In spite of this, high parallel efficiency can be achieved when the communication to computation ration is relatively small. This is indeed the case when using moderate number of processors to solve a large problem as shown by the experimental results in the next section.

4 Experiments

In order to validate the proposed algorithm and the effectiveness of of the preconditioner, a driven-cavity problem is solved on a three dimensional unit cube. The Marker and Cell (MAC) scheme is used to discretize the domain. The linear system arising from a generalized Stokes problem was solved under various physical conditions by changing the ratio $h^2R/\Delta t$. The nature of linear systems in incompressible flow problems depends on various parameters such as mesh width (h), Reynolds number (R) and time step (Δt). Convergence of the iterative scheme is sensitive to the variation in the condition number of the matrix A. For the 3D driven cavity problem, the condition number is approximated by

$$\kappa(A) = \frac{h^2R/\Delta t + 12}{h^2R/\Delta t + h^2}.$$

Hence, $\kappa(A) < 2$ when $h^2R/\Delta t > 12$, and $\kappa(A) \approx h^{-2}$ when $h^2R/\Delta t \ll 12$.

Mesh Size	$h^2R/\Delta t$				
	10^{-3}	10^{-1}	10^{0}	10^{+1}	10^{+3}
With Preconditioning					
$8 \times 8 \times 8$	8	8	6	5	5
$16 \times 16 \times 16$	12	10	8	6	6
$32 \times 32 \times 32$	17	13	10	7	7
Without Preconditioning					
$8 \times 8 \times 8$	66	64	54	29	21
$16 \times 16 \times 16$	208	188	114	58	42
$32 \times 32 \times 32$	772	552	225	106	77

Table 1. Preconditioning is highly effective in reducing the number of iterations. The iterations for the preconditioned case are almost invariant with respect to the flow parameters h, R, and Δt.

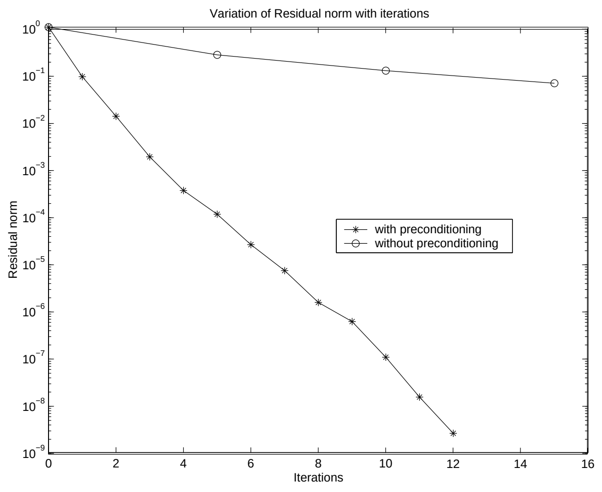

Figure 2. The relative residual norm decreases by a constant factor at each iteration of the preconditioned CG method. Experiment shown is for $16 \times 16 \times 16$ grid problem with $h^2R/\Delta t = 10^{-3}$.

	Mesh size = 16 × 16 × 16	
Inner tolerance	Outer iterations	Inner iterations
10^{-4}	10	34
10^{-3}	10	28
10^{-2}	10	20
$5X10^{-2}$	12	12
10^{-1}	16	8

Table 2. Lower accuracy suffices for the preconditioning step.

Number of processors	Processor grid	Time (secs)	Speed-up	Efficiency
1	$1 \times 1 \times 1$	1760	–	1.0
2	$2 \times 1 \times 1$	946	1.86	0.93
4	$2 \times 2 \times 1$	496	3.54	0.86
8	$2 \times 2 \times 2$	259	6.79	0.85
16	$4 \times 2 \times 2$	136	12.94	0.81
32	$4 \times 4 \times 2$	73	24.11	0.75
64	$4 \times 4 \times 4$	49	35.92	0.56

Table 3. Parallel results for $64 \times 64 \times 64$ grid.

Preconditioned CG is used to solve the reduced system (6) with the stopping criteria of relative residual norm being less than 10^{-4}. The preconditioner step is also an inner CG solve with a lower stopping criterion of 10^{-2}. The first set of experiments highlights the effectiveness of the preconditioner in accelerating the convergence of the CG method. Figure. 2 shows that the relative residual norm decreases by a constant factor in each iteration of the preconditioned CG method.

Table 1 presents the iterations required by the preconditioned CG method for several instances of $h^2R/\Delta t$. The preconditioner ensures a stable convergence rate independent of the values of h, R and Δt. This suggests that the preconditioning is optimal. At each iteration, the preconditioning system was solved by CG as well, resulting in an inner-outer scheme. The lower accuracy of the inner solve had no adverse effect on the outer iterations which reduces the computational cost of the expensive preconditioning step. Table 2 shows that a tolerance of 10^{-2} for the inner iterations is sufficient for convergence of the outer iterations.

A second set of experiments focused on the parallelization of the proposed algorithm. All the elementary matrix-vector products and vector dot products were parallelized by distributing the grid equally among the processors by partitioning in all the physical dimensions. Communication buffers are used to reduce the inter-processor communication overhead. Experiments were performed on SGI Origin 2000 using OpenMP. An MPI based distributed memory implementation can be developed with minimal changes as the underlying communication is performed using buffers. Problem sizes of $64 \times 64 \times 64$ and $128 \times 128 \times 128$ were solved for $h^2R/\Delta t = 10^{+3}$ with stopping criteria of 10^{-4}. Preconditioner step is solved with a low accuracy of 10^{-2}.

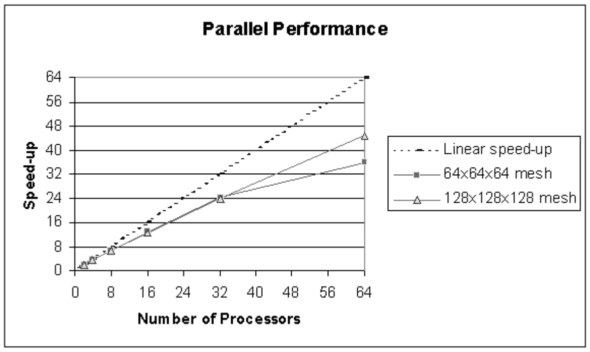

Figure 3. Experiments performed on SGI Origin 2000 show that the parallelization scheme is highly efficient on a multi processor platform.

Figure 3 shows that the parallelization scheme is highly efficient on a multi processor platform.

As seen from Table 3, high efficiency is obtained on the $64 \times 64 \times 64$ sized problem on 32 processors. The degradation in performance on 64 processors is due to the commu-

Number of processors	Processor grid	Time (secs)	Speed-up	Efficiency
1	$1 \times 1 \times 1$	27754	–	1.0
2	$2 \times 1 \times 1$	14946	1.86	0.93
4	$2 \times 2 \times 1$	8064	3.44	0.86
8	$2 \times 2 \times 2$	4295	6.46	0.81
16	$4 \times 2 \times 2$	2212	12.54	0.78
32	$4 \times 4 \times 2$	1164	23.84	0.74
64	$4 \times 4 \times 4$	620	44.77	0.70

Table 4. Parallel results for $128 \times 128 \times 128$ grid.

nication over-head dominating the computation per processor. Increasing the problem size to $128 \times 128 \times 128$ ensures higher efficiency on 64 processors. This can be observed in Table 4 and Figure 3. Thus, efficiency can be maintained on larger number of processors by increasing the problem size.

5 Conclusions and Ongoing Work

This paper presents a high performance algorithm for solving the linear systems arising from incompressible flow problems. An algebraic scheme is outlined to compute discrete local solenoidal functions that are used to represent divergence-free velocity. A reduced system is solved in the divergence-free subspace via a preconditioned iterative scheme. An optimal preconditioner has been suggested which assures stable convergence regardless of parameters such as the mesh width, Reynolds number, and the time step. An inexpensive low accuracy iterative solve for the preconditioner appears to be sufficient for optimal convergence. An efficient parallel computation of matrix-vector products is given. Mesh partitioning and the ratio of computation to communication affect the speed-up on multiprocessor runs. Algorithm is shown to be scalable even for a smaller sized problem of $64 \times 64 \times 64$ mesh. Research is under way in extending the algorithm to finite element unstructured grids.

References

[1] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In *Acta Numerica*, pages 61–143. Cambridge University Press, 1994.

[2] B. Hendrickson and R. Leland. The chaco user's guide version 2.0, technical report sand94-2692. Sandia National Laboratories, Albuquerque, NM, 1994.

[3] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Rev*, 41(2):278–300, 1999.

[4] M. G. Knepley, A. H. Sameh, and V. Sarin. Design of large scale parallel simulations. In D. Keyes, A. Ecer, J. Périaux, N. Satofuka, and P. Fox, editors, *Parallel Computational Fluid Dynamics*. Elsevier, 2000.

[5] V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to parallel computing: Design and analysis of algorithms. Benjamin/Cummings, 1994.

[6] T.-W. Pan, V. Sarin, R. Glowinski, A. H. Sameh, and J. Périaux. A fictitious domain method with distributed Lagrange multipliers for the numerical simulation of particulate flow and its parallel implementation. In C. A. Lin, A. Ecer, N. Satofuka, P. Fox, and J. Périaux, editors, *Parallel Computational Fluid Dynamics, Development and Applications of Parallel Technology*, pages 467–474. North-Holland, Amsterdam, 1999.

[7] Y. Saad. Iterative methods for sparse linear systems. PWS publishing company, 1996.

[8] S. R. Sambavaram and V. Sarin. A parallel solenoidal basis method for incompressible fluid flow problems. In *Proceedings of the Parallel CFD Conference*. Egmond aan Zee, The Netherlands (in press), 2001.

[9] V. Sarin. Parallel linear solvers for incompressible fluid problems. In *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*. Portsmouth, VA, Mar. 2001.

[10] V. Sarin and A. H. Sameh. An efficient iterative method for the generalized Stokes problem. *SIAM Journal of Scientific Computing*, 19(1):206–226, 1998.

[11] V. Sarin and A. H. Sameh. Large scale simulation of particulate flows. In *Proceedings of the Second Merged Symposium IPPS/SPDP*, pages 660–666, Apr. 1999.