

# Parallel solution of large-scale free surface viscoelastic flows via sparse approximate inverse preconditioning

Zenaida Castillo<sup>a</sup>, Xueying Xie<sup>b</sup>, Danny C. Sorensen<sup>a</sup>,  
Mark Embree<sup>\*,a</sup>, Matteo Pasquali<sup>\*,b</sup>

<sup>a</sup>*Department of Computational and Applied Mathematics  
Computer and Information Technology Institute  
Rice University, 6100 Main Street – MS 134, Houston, Texas 77005–1892*

<sup>b</sup>*Department of Chemical and Biomolecular Engineering  
Computer and Information Technology Institute  
Rice University, 6100 Main Street – MS 362, Houston, Texas 77005–1892*

---

## Abstract

Though computational techniques for two-dimensional viscoelastic free surface flows are well developed, three-dimensional flows continue to present significant computational challenges. Fully coupled free surface flow models lead to nonlinear systems whose steady states can be found via Newton's method. Each Newton iteration requires the solution of a large, sparse linear system, for which memory and computational demands suggest the application of an iterative method, rather than the sparse direct methods widely used for two dimensional simulations. The Jacobian matrix of this system is often ill-conditioned, resulting in unacceptably slow convergence of the linear solver; hence preconditioning is essential. In this paper we propose a variant sparse approximate inverse preconditioner for the Jacobian matrix that allows for the solution of problems involving more than a million degrees of freedom in challenging parameter regimes. Construction of this preconditioner requires the solution of small least squares problems that can be simply parallelized on a distributed memory machine. The performance and scalability of this preconditioner with the GMRES solver are investigated for two- and three-dimensional free surface flows on both structured and unstructured meshes in the presence and absence of viscoelasticity. The results suggest that this preconditioner is an extremely promising candidate for solving large-scale steady viscoelastic flows with free surfaces.

*Key words:* free surface, viscoelastic flow, parallel GMRES, SPAI, approximate inverse preconditioner

---

# 1 Introduction

Flows with free surfaces and free boundaries arise in industrial and biological applications as varied as polymer processing, coating, ink-jet printing, spraying, deformation of blood cells, blood flow in arteries and capillaries, and flow in the deep pulmonary alveoli. Most of these flows exhibit two distinguishing features: (1) the fluid is complex, i.e., it has microstructural features, and thus the Cauchy stress is not merely composed of viscous and pressure forces, but includes a viscoelastic term that is important and sometimes controlling; (2) the surface forces are comparable to or dominate the viscous and elastic forces due to the flow of the liquid and the subsequent deformation of the microstructure. Examples include coating and ink-jet flows of polymer solutions, where the flow-induced deformation of the polymer molecules can generate steep layers of elastic stress.

Because surface and viscoelastic forces are often more important than viscous ones, there are large non-diagonal contributions in the momentum equations that arise from the deformation of the free surfaces or elastic boundaries, and from the microstructural elastic stress. Thus, fully coupled algorithms for solving the flow equations are desirable. To solve for the moving boundaries together with velocity, pressure, and stress, an equation to describe the mesh must be incorporated into the model. Several methods for the free surface problem have been developed, chiefly elliptic mesh generation and the domain deformation method. These methods have been successful in describing 2D free surface problems [9,6,19,18,20]. Three dimensional free surface flow computations still present challenges [5,15,27], owing in large part to the scale of the computation.

A domain deformation method is used in this study to solve both 2D and 3D free surface flows. The continuous model is discretized to a set of nonlinear algebraic equations via the DEVSS-TG/SUPG mixed finite element method [14,20,24], which has proved to be an effective and flexible way of studying systems where viscoelasticity and surface forces are important and the physical domains are complex.

The nonlinear algebraic equations are solved by Newton's method. At each Newton iteration a linear system  $\mathbf{Ax} = \mathbf{b}$  needs to be solved with the Jacobian as the coefficient matrix. Direct frontal solvers (see, e.g., [10]) have been widely applied to solve 2D free surface flows [9,6,19]; however, for 3D flows direct solvers are impractical due to the system's dimension (frequently over one million unknowns). Alternatively an iterative method, such as the

---

\* Corresponding authors.

*Email addresses:* embree@rice.edu (Mark Embree), mp@rice.edu (Matteo Pasquali).

GMRES algorithm [23], can be applied [5,1]. In the free surface problems considered here, the Jacobian matrices are highly ill-conditioned, with small or zero diagonal entries due to the coupling of the free surface and elastic stresses with the flow equations. Consequently GMRES converges very slowly when applied to this poorly conditioned system, and an effective preconditioner is essential.

The most commonly used general-purpose preconditioners are based on incomplete factorizations of  $\mathbf{A}$ , such as the ILU preconditioners, which perform an incomplete LU factorization of  $\mathbf{A}$  (see, e.g., [3]). However, the success of an ILU preconditioner depends on its ability to handle several potential problems, including zero pivots, instability of the LU factors, and the challenge of parallel implementation. The latter issue is decisive for our application. Approximate inverse preconditioners can avoid some of these challenges: Instead of factoring  $\mathbf{A}$ , one seeks a sparse approximation to  $\mathbf{A}^{-1}$ , and replaces the forward and backward solves of ILU preconditioning with a low-cost sparse matrix-vector product.

One typical approach for constructing an approximate inverse preconditioner is based on optimization. The idea, introduced by Benson and Frederickson [2], is to build a sparse preconditioner  $\mathbf{M}$  that minimizes  $\|\mathbf{I} - \mathbf{A}\mathbf{M}\|_F$  (or  $\|\mathbf{I} - \mathbf{M}\mathbf{A}\|_F$  for left preconditioning), subject to sparsity constraints on  $\mathbf{M}$ . The use of the Frobenius norm  $\|\cdot\|_F$  provides inherent parallelism because the constrained minimization problem decouples into independent linear least square problems, one for each column (row) of  $\mathbf{M}$  for right (left) preconditioning.

## 2 Mathematical formulation

We begin by describing a mathematical model for viscoelastic free surface flows.<sup>1</sup> The fully-coupled system includes equations for the domain, interpolated velocity gradient, and transport of the mass, momentum, and conformation tensor—twenty-two coupled scalar partial differential equations in three dimensions. These equations are solved simultaneously to obtain the unknown position, pressure, velocity, conformation tensor, and velocity gradient. The following system represents a steady and incompressible viscoelastic flow [20]:

---

<sup>1</sup> Throughout, boldface roman denotes physical vectors and tensors, while sans-serif represents algebraic vectors and tensors. Gibbs' notation [26] is used for operations between physical vectors and tensors; standard linear algebra notation is used elsewhere.

$$0 = \nabla \cdot \mathbf{T}^e \quad (1)$$

$$0 = \nabla \cdot \mathbf{v} \quad (2)$$

$$0 = \rho \mathbf{v} \cdot \nabla \mathbf{v} - \nabla \cdot \mathbf{T} - \nabla \Theta \quad (3)$$

$$0 = \mathbf{L} - \nabla \mathbf{v} + \frac{1}{\text{tr}(\mathbf{I})}(\nabla \cdot \mathbf{v})\mathbf{I} \quad (4)$$

$$0 = \mathbf{v} \cdot \nabla \mathbf{M} - 2\xi \frac{\mathbf{D} : \mathbf{M}}{\mathbf{I} : \mathbf{M}} \mathbf{M} - \zeta (\mathbf{M} \cdot \mathbf{D} + \mathbf{D} \cdot \mathbf{M} - 2 \frac{\mathbf{D} : \mathbf{M}}{\mathbf{I} : \mathbf{M}} \mathbf{M}) \\ - \mathbf{M} \cdot \mathbf{W} - \mathbf{W}^T \cdot \mathbf{M} - \frac{1}{\lambda} (g_0 \mathbf{I} + g_1 \mathbf{M} + g_2 \mathbf{M}^2), \quad (5)$$

where  $\text{tr}(\mathbf{I})$  is the trace of the identity  $\mathbf{I}$ , which is of order two if the flow is two-dimensional and of order three for a three-dimensional flow,  $\mathbf{D} : \mathbf{M} = \sum_i \sum_j D_{ij} M_{ji} = \text{tr}(\mathbf{D} \cdot \mathbf{M})$ , and  $\nabla$  is the gradient vector in space;  $\mathbf{T}^e$  is the stress tensor of the elastic pseudo-solid for the domain,  $\mathbf{v}$  is the velocity vector,  $\rho$  is the material density,  $\mathbf{T}$  is the total stress tensor,  $\Theta$  is the potential body force per unit volume,  $\mathbf{L}$  is the interpolated velocity gradient,  $\mathbf{M}$  is the conformation tensor,  $\xi$  and  $\zeta$  are the polymer resistance to stretching and orientation,  $\mathbf{D} \equiv \frac{1}{2}(\mathbf{L} + \mathbf{L}^T)$  is the rate of strain,  $\mathbf{W} \equiv \frac{1}{2}(\mathbf{L} - \mathbf{L}^T)$  is the vorticity,  $\lambda$  is the characteristic relaxation time, and  $g_0$ ,  $g_1$  and  $g_2$  are relaxation functions.

Notice that for Newtonian free surface flows, only the first three equations (1)–(3) are needed.

Appropriate boundary conditions must be imposed on equations (1)–(5). The mesh generation equation (1) has second-order derivatives of position (as  $\mathbf{T}^e$  is related to position derivatives); thus, boundary conditions must be imposed on all boundaries. The momentum equation (3) is elliptic and hence momentum boundary conditions must also be specified on all boundaries. The transport equation (5) of the conformation tensor is a hyperbolic equation, which requires boundary conditions only at the inflow boundaries, i.e., where  $\mathbf{n} \cdot \mathbf{v} < 0$ , with  $\mathbf{n}$  a unit outward normal vector [21]. The continuity equation (2) and the velocity gradient equation (4) do not require boundary conditions.

In this study we use the following boundary conditions for the mesh generation equation:

- Fixed node:  $\mathbf{x} = \mathbf{x}_0$ , where  $\mathbf{x}_0$  is the fixed position;
- Free surface:  $\mathbf{n} \cdot \mathbf{v} = 0$  and  $\mathbf{t}\mathbf{n} : \mathbf{T}^e = (\mathbf{n} \cdot \mathbf{T}^e) \cdot \mathbf{t} = 0$ , where  $\mathbf{t}$  is the unit tangent vector and  $(\mathbf{n} \cdot \mathbf{T}^e)$  is a row vector.

The momentum boundary conditions used in this study are:

- Fixed velocity:  $\mathbf{v} = \mathbf{v}_0$  where  $\mathbf{v}_0$  is a constant velocity;
- Velocity profile:  $\mathbf{v}(\mathbf{x}) = f(Q, \mathbf{x})$  where  $Q$  is a known flow rate and  $f$  is a function;

- Symmetric boundary:  $\mathbf{n} \cdot \mathbf{v} = 0$  and  $\mathbf{tn} : \mathbf{T} = 0$ ;
- Free surface condition:  $\mathbf{n} \cdot \mathbf{T} = \nabla_{\mathcal{H}} \cdot (\gamma(\mathbf{I} - \mathbf{nn}))$ , where  $\gamma$  is the surface tension and  $\nabla_{\mathcal{H}} = (\mathbf{I} - \mathbf{nn}) \cdot \nabla$  is the surface divergence.

The boundary condition imposed on the transport equation for the conformation tensor is  $\mathbf{v} \cdot \nabla \mathbf{M} = \mathbf{0}$ .

In addition to the above boundary conditions, some related dimensionless parameters are implicitly defined in equations (1)–(5):

- The Reynolds number,  $\text{Re} \equiv \rho v d / \mu$ , characterizes the balance between inertial and surface (viscous and viscoelastic) forces, where  $\mu$  is the total viscosity;
- The capillary number,  $\text{Ca} \equiv \mu v / \gamma$ , measures the relative importance of the surface (viscous and viscoelastic) forces to surface tension forces.

Here,  $v$  and  $d$  are the characteristic velocity and length of a flow, and  $\gamma$  is the surface tension of this flow. For a viscoelastic flow, two more dimensionless numbers are defined:

- The Weissenberg number,  $\text{We} \equiv \lambda \dot{\gamma}_c$ , represents the intensity of the flow on the scale of the relaxation time of the polymer conformation, where  $\dot{\gamma}_c$  is a characteristic shear rate;
- The solvent viscosity ratio,  $\beta \equiv \eta_s / (\eta_s + \eta_p) \equiv \eta_s / \mu$ , characterizes the relative importance of viscous and viscoelastic stresses, where  $\eta_s$  is the solvent viscosity and  $\eta_p$  is the polymer viscosity.

For a detailed description of these parameters and equations (1)–(5), see Xie and Pasquali [27,28].

We discretize equations (1)–(5) by the DEVSS-TG/SUPG mixed finite element method [14,24]; for details, see [19,20]. Structured quadrilateral elements are used for the 2D problems. The position and velocity basis functions are biquadratic and continuous, the velocity gradient and conformation basis functions are bilinear and continuous, and the pressure basis functions are linear and discontinuous. Unstructured tetrahedral elements are used for the 3D problems. The basis functions for position and velocity are quadratic and continuous, those for pressure, velocity gradient and conformation are linear and continuous. The unknowns associated with each node are ordered consecutively, so that the Jacobian matrix has banded structure with entries near the diagonal corresponding to intra-element coupling and nonzero entries farther from the diagonal describing inter-element effects. The fully coupled set of nonlinear algebraic equations is solved by Newton’s method with the analytical Jacobian, requiring at each step the solution of a large scale system of linear algebraic equations,  $\mathbf{J} \Delta \mathbf{x} = -\mathbf{r}$ , where  $\mathbf{J}$  is the Jacobian matrix,  $\Delta \mathbf{x}$  is the Newton update, and  $\mathbf{r}$  is the Newton residual. Here our main concern is

the construction of a preconditioner to facilitate the solution of this system.

### 3 The preconditioner

In this section we describe the preconditioner we propose in terms of the generic linear system  $\mathbf{Ax} = \mathbf{b}$  with  $n$  unknowns. Given the sparsity structure of the free-surface flow problems we aim to solve, the fill-in and organization of conventional incomplete LU factorization preconditioners inhibits overall parallel scalability. For this reason we turned our attention to sparse approximate inverses.

A considerable variety of such preconditioners have been proposed, including factored versions proposed by Kolotilina, Yeremin, and co-authors [17] and Benzi and Tuma [4], and unfactored preconditioners proposed by Cosgrove, Diaz and Griewank [8], Grote and Simon [12], and Chow and Saad [7]. In preliminary experiments (using the global Jacobian matrix on a 2D free-surface viscoelastic flow problem) we found that the Benzi–Tuma preconditioner suffered from an unacceptable loss of sparsity, and thus we focused our attention on the unfactored preconditioner of Grote and Huckle [13].

Because the inverse of a sparse matrix is generally dense, the quality of the sparse approximation  $\mathbf{M}$  to  $\mathbf{A}^{-1}$  depends critically on the sparsity pattern chosen for  $\mathbf{M}$ . This choice can be influenced by the sparsity pattern of  $\mathbf{A}$  and knowledge accumulated from experience with a particular application, or determined through a dynamic algorithm. As the sparsity of  $\mathbf{M}$  decreases, the accuracy of the preconditioner improves, yet  $\mathbf{M}$  becomes more expensive to compute, store, and apply. Thus, to construct an effective preconditioner one must balance the virtues of sparsity with the need for a good approximation to  $\mathbf{A}^{-1}$ . See, for example, the discussion by Tang [25].

One convenient method for constructing a sparse approximate inverse seeks the matrix  $\mathbf{M}$  of a given sparsity pattern that minimizes  $\|\mathbf{I} - \mathbf{AM}\|_F$ . The choice of the Frobenius norm  $\|\cdot\|_F$  leads to inherent parallelism, since

$$\min \|\mathbf{I} - \mathbf{AM}\|_F^2 \equiv \min \sum_{k=1}^n \|\mathbf{e}_k - \mathbf{Am}_k\|_2^2, \quad (6)$$

where  $\mathbf{e}_k$  and  $\mathbf{m}_k$  represent the  $k$ th columns of  $\mathbf{I}$  and  $\mathbf{M}$ . The solution of equation (6) decouples into

$$\min_{\mathbf{m}_k} \|\mathbf{e}_k - \mathbf{Am}_k\|_2, \quad k = 1, \dots, n, \quad (7)$$

each of which can be solved independently.

With no restriction imposed on  $\mathbf{m}_k$ , problem (7) is uniquely solved by the  $k$ th column of  $\mathbf{A}^{-1}$ , and thus one would effectively solve  $n$  linear systems with  $\mathbf{A}$  to construct a preconditioner to expedite the solution of the single system  $\mathbf{A}\mathbf{x} = \mathbf{b}$ . To obtain a more practical algorithm one imposes sparsity constraints upon each column  $\mathbf{m}_k$ . Zero entries in  $\mathbf{m}_k$  prevent the corresponding columns of  $\mathbf{A}$  from contributing to the objective function in (7), and thus we expect many rows of  $\mathbf{e}_k - \mathbf{A}\mathbf{m}_k$  to be zero for all  $\mathbf{m}_k$  with the desired sparsity pattern. Effectively, we thus have a small least squares problem that can be solved exactly and efficiently using a dense QR factorization. The challenge, of course, is to determine a satisfactory sparsity pattern for  $\mathbf{M}$  that produces an effective preconditioner at a minimum cost to construct and apply. Here our approach is a variant of the algorithm of Grote and Huckle [13], which constructs an unfactored approximate inverse based on a preliminary sparsity pattern for  $\mathbf{M}$ . This preconditioner is then improved through refinement steps that increase the number of nonzero entries in each column of  $\mathbf{M}$ .

Throughout we focus on construction of a *right* preconditioner for our matrix  $\mathbf{A}$ , with which the linear system  $\mathbf{A}\mathbf{x} = \mathbf{b}$  is transformed into  $\mathbf{A}\mathbf{M}\mathbf{y} = \mathbf{b}$  with  $\mathbf{x} = \mathbf{M}\mathbf{y}$ . Right preconditioning has the virtue that the residual norm used in the stopping criterion for preconditioned GMRES is also the residual for the original linear system. Once GMRES has determined a suitable approximation to  $\mathbf{y}$ , the desired solution  $\mathbf{x}$  is recovered through a simple matrix-vector multiplication with  $\mathbf{M}$ .

### 3.1 The Grote–Huckle SPAI algorithm

We now recapitulate the basic algorithm of Grote and Huckle [13], focusing on the construction of  $\mathbf{m}_k$ , the  $k$ th column of  $\mathbf{M}$ . Let the set  $\mathcal{J}$  comprise the indices of entries of  $\mathbf{m}_k$  that are initially permitted to be nonzero. Since  $\mathbf{A}$  is sparse, many entries in the product  $\mathbf{A}\mathbf{m}_k$  will also be zero, and we wish to avoid computing them. Toward this end define the set  $\mathcal{I}$  such that  $\mathbf{A}(i, j) \neq 0$  if and only if  $i \in \mathcal{I}$  and  $j \in \mathcal{J}$ .<sup>2</sup> With this notation, the original large optimization problem (7) constrained by the sparsity of  $\mathbf{m}_k$  reduces to the small unconstrained problem

$$\min_{\widehat{\mathbf{m}}_k} \|\widehat{\mathbf{e}}_k - \widehat{\mathbf{A}}\widehat{\mathbf{m}}_k\|_2,$$

where  $\widehat{\mathbf{e}}_k = \mathbf{e}_k(\mathcal{I})$ ,  $\widehat{\mathbf{A}} = \mathbf{A}(\mathcal{I}, \mathcal{J})$ , and  $\widehat{\mathbf{m}}_k = \mathbf{m}_k(\mathcal{J})$  denote the reduced vectors and coefficient matrix. This standard least squares problem can be readily

<sup>2</sup> Here we use MATLAB notation to specify the entries of a matrix; e.g.,  $\mathbf{A}(i, j)$  denotes the  $(i, j)$  entry of  $\mathbf{A}$ ;  $\mathbf{A}(\mathcal{I}, \mathcal{J})$  denotes all entries  $\mathbf{A}(i, j)$  with  $i \in \mathcal{I}$  and  $j \in \mathcal{J}$ ; and  $\mathbf{A}(:, \mathcal{L})$  denotes all entries  $\mathbf{A}(i, \ell)$  for  $i = 1, \dots, n$  and  $\ell \in \mathcal{L}$ .

solved for  $\widehat{\mathbf{m}}_k$  via the QR factorization of  $\widehat{\mathbf{A}}$ , and one can then easily expand  $\widehat{\mathbf{m}}_k$  to obtain  $\mathbf{m}_k$ .

Typically the set  $\mathcal{J}$  is insufficient to yield a column residual  $\mathbf{r}_k \equiv \mathbf{e}_k - \mathbf{A}\mathbf{m}_k$  with small norm, and thus one seeks an automatic way of complementing this index set to most effectively reduce  $\|\mathbf{r}_k\|$ . Grote and Huckle proposed the following heuristic. Let the set  $\mathcal{L}$  consist of the indices of all elements  $\ell$  of  $\mathbf{r}_k$  such that  $|\mathbf{r}_k(\ell)| > tol$  for some user-specified tolerance  $tol$ . Then the new entries of  $\mathbf{m}_k$  that will be permitted to fill-in will be drawn from the set

$$\widetilde{\mathcal{J}} = \{j \in \{1, \dots, n\} : j \notin \mathcal{J} \text{ and } \mathbf{A}(\ell, j) \neq 0 \text{ for some } \ell \in \mathcal{L}\}.$$

We wish to identify those elements in  $\widetilde{\mathcal{J}}$  whose contribution to  $\mathbf{m}_k$  would cause the greatest reduction in the residual norm. Suppose we keep  $\mathbf{m}_k$  the same except for allowing  $\mathbf{m}_k(j)$  to fill in for some  $j \in \widetilde{\mathcal{J}}$ . The optimal value for  $\mathbf{m}_k(j)$  is given by

$$\mu_j = \arg \min_{\mu} \|\mathbf{r}_k - \mu \mathbf{A}\mathbf{e}_j\|_2 = \frac{\mathbf{r}_k^T \mathbf{A}\mathbf{e}_j}{\|\mathbf{A}\mathbf{e}_j\|_2^2},$$

from which follows a measure of the improved residual norm:

$$\rho_j = \|\mathbf{r}_k - \mu_j \mathbf{A}\mathbf{e}_j\|_2^2 = \|\mathbf{r}_k\|_2^2 - \frac{(\mathbf{r}_k^T \mathbf{A}\mathbf{e}_j)^2}{\|\mathbf{A}\mathbf{e}_j\|_2^2}.$$

The indices  $j$  that reduce the residual the most are those for which  $\rho_j$  has the smallest value. Since  $\mathbf{A}$  is invertible,  $\mathbf{A}(\mathcal{L}, \cdot)$  has full row rank, and thus for any nonzero  $\mathbf{r}_k$  and nontrivial  $\mathcal{L}$  and  $\widetilde{\mathcal{J}}$ ,

$$\mathbf{r}_k(\mathcal{L})^T \mathbf{A}(\mathcal{L}, \widetilde{\mathcal{J}}) \neq \mathbf{0},$$

which guarantees that at least one  $j \in \widetilde{\mathcal{J}}$  will yield some improvement in the residual norm. Often many entries in  $\widetilde{\mathcal{J}}$  will yield improvement, and it may prove best to restrict the new nonzero entries to be those  $j \in \widetilde{\mathcal{J}}$  that yield the greatest improvement. One can make multiple passes through this refinement algorithm until  $\|\mathbf{r}_k\|$  is acceptably small or  $\mathbf{m}_k$  contains some maximum number *maxcfill* of nonzero entries.

Grote and Huckle [13] show that the SPAI process is convergent as  $tol \rightarrow 0$  and *maxcfill*  $\rightarrow n$ . Various bounds on the accuracy of the preconditioner exist; for example, if  $\|\mathbf{r}_k\|_2 = \|\mathbf{e}_k - \mathbf{A}\mathbf{m}_k\|_2 < tol$  for all  $k = 1, \dots, n$ , then  $\|\mathbf{I} - \mathbf{A}\mathbf{M}\|_2 \leq tol\sqrt{n}$ .

### 3.2 Modification of the SPAI algorithm for free surface flows

Our primary goal is the solution of linear systems involving the Jacobian matrices arising in free surface flow applications, systems that arise at each step of a Newton iteration. Figure 1 shows the sparsity pattern for a typical Jacobian arising from a 3D rod coater, Problem 2 described below. Though this matrix has a distinct band structure, that bandwidth is a significant proportion of the matrix dimension. Assembly of this Jacobian from the elemental matrices used in the finite element discretization code would be prohibitively expensive, particularly given the distribution of such elemental matrices over processors in a parallel machine. (In 3D problems, the elemental Jacobian matrices alone are of dimension 124.) For this reason, the conventional SPAI algorithm is unappealing for very large problems.

We obtain a more satisfactory preconditioner by applying the SPAI algorithm to a matrix  $\tilde{\mathbf{A}}$  comprising the central bands of  $\mathbf{A}$ . This variant of SPAI requires the definition of the parameters *band*, *nc*, *tol*, and *maxcfill*. The parameter *band* specifies the number of diagonal bands of the Jacobian used for the construction of the preconditioner, so that

$$\tilde{\mathbf{A}}(j, k) = \begin{cases} \mathbf{A}(j, k) & |j - k| \leq \textit{band}; \\ 0 & \textit{otherwise.} \end{cases}$$

The parameter *nc* restricts the number of passes of residual improvement allowed for each column of  $\mathbf{M}$ , and *tol* flags those entries in each column that are eligible for improvement; *maxcfill* denotes the maximum number of entries that are allowed to fill within each column of  $\mathbf{M}$ . Intuitively, by basing  $\mathbf{M}$  on the interior band of  $\mathbf{A}$  we construct a preconditioner that emphasizes the intra-element coupling in equations (1)–(5), while exerting less effort to capture the coupling between elements. The experiments described in the next section demonstrate that this strategy is highly effective for our applications.

## 4 Test problems and computational experiments

In this section we test the effectiveness of our preconditioner on a variety of flows in a 2D slot coater and a 3D rod coater, both with free surface boundaries. The results include the solution of a 3D Oldroyd-B fluid with Weissenberg number  $We = 0.5$  on a grid with more than a million degrees of freedom.

### **Problem 1: A Newtonian free surface flow in a 2D slot coater**

This problem comes from a Newtonian 2D slot coater flow with free surface

boundary as shown in Figure 2. The potential body force is  $\Theta = 0$ , the flow rate per unit width is  $Q = 0.5hv$ , the Reynolds number is  $\text{Re} \equiv \rho v h / \mu = 0$ , and the capillary number is  $\text{Ca} \equiv \mu v / \gamma = 0.1$ , where  $h$  is the gap and  $v$  is the velocity of the bottom wall. Because this is a Newtonian flow, only equations (1)–(3) are required.

**Problem 2: A viscoelastic free surface flow in a 3D rod coater**

This example is a free surface flow with an Oldroyd-B fluid in a 3D rod coater. An Oldroyd-B liquid is a very simple model for a viscoelastic fluid that is commonly used for numerical experimentation. Because the 3D rod coating is axisymmetric, only one quarter of the channel is needed to fully characterize the flow, as shown in Figure 3. The flow conditions are set as follows: volume flow rate  $Q = 0.754$ ,  $\text{Re} \equiv \rho U_0 R_1 / \mu = 0$ , the solvent viscosity ratio  $\beta_\mu \equiv \eta_s / \mu = 0.59$ , and the capillary number  $\text{Ca} \equiv \mu U_0 / \gamma = 1$ , where  $U_0 = 1$  is the velocity of the rod,  $\eta_s$  is solvent viscosity, and  $\lambda$  is the relaxation time of the Oldroyd-B fluid. Our experiments take the Weissenberg number  $\text{We} \equiv \lambda U_0 / (R_2 - R_1)$  equal to 0.5 or 1.

The parameters in the conformation transport equation (5) for the Oldroyd-B fluid are  $\xi = 1$ ,  $\zeta = 1$ ,  $g_0 = -1$ ,  $g_1 = 1$ ,  $g_2 = 0$ , and  $\partial a / \partial \mathbf{M} = (G/2\rho)\mathbf{I}$ , where  $G = \eta_p / \lambda$  is the polymer modulus and  $\eta_p$  is polymer viscosity with  $\eta_p + \eta_s = \mu$ .

*4.1 Experimental environment*

This set of experiments first compares the behavior of three approaches for solving the linear systems arising in our free surface flow problems, and then provides a more detailed analysis of the performance of SPAI-GMRES. The three approaches are:

- **FS**: A direct frontal solver, based on the ideas presented in [10]. The original implementation, due to de Almeida [9], has proved to be highly efficient and reliable in years of testing on a variety of 2D flow problems.
- **ILUT-GMRES**: An iterative solver implemented by Saad in the SPARSKIT library [22]. The incomplete LU preconditioner requires the parameters *L-fill* (the level of fill-in allowed in the factors *L* and *U*) and *drop-tol* (the drop-tolerance). The SPARSKIT GMRES solver requires the Krylov subspace size, *Krylov-size* (the maximum Krylov subspace size) and *restart* (the number of restarts allowed).
- **SPAI-GMRES(*p*)**: A GMRES solver (a parallel adaptation of the serial SPARSKIT implementation) preconditioned by the sparse approximate inverse described in Section 3; here *p* denotes the number of processors used. The preconditioner is specified by the parameters *band*, *nc*, *maxcfill*, and *tol*. For all experiments described here we use  $\text{maxcfill} = 2 \times (\text{band} - 1)$

and  $tol = 0.01$ , and set  $nc$  such that no more than two passes of residual improvements are performed for each column. The initial sparsity pattern  $\mathcal{J}$  for column  $\mathbf{m}_j$  of  $\mathbf{M}$  corresponds to the nonzero pattern of the banded portion of the Jacobian. At each pass of residual refinement for  $\mathbf{m}_j$ , we limit the number of new nonzero entries to  $\lfloor \frac{1}{2}(maxcfill - |\mathcal{J}|) \rfloor$ , where  $|\mathcal{J}|$  denotes the number of entries in the initial sparsity pattern for  $\mathbf{m}_j$ .

We seek steady state solutions for Problems 1 and 2 by solving equations (1)–(5) with Newton’s method. At each Newton iteration, the linear algebraic equations will be solved by the frontal solver (FS), ILUT preconditioned GMRES (ILUT-GMRES), and sparse approximate inverse preconditioned GMRES (SPAI-GMRES( $p$ )). Table 1 contains data describing the ten meshes used on these two problems; Mesh 1 is used for Problem 1 and Meshes 2–10 are used for Problem 2.

Based on practical experience and the recommendations given by Kelley [16], unless otherwise noted we require the GMRES residual norm reach  $10^{-3}$  in the first Newton iteration and  $\min\{10^{-7}, \beta\|\mathbf{r}_{k-1}\|^2/\|\mathbf{r}_{k-2}\|^2\}$  in the  $k$ th Newton iteration ( $k = 2, 3, \dots$ ), where  $\beta$  is a constant set to 0.9 for this study and  $\|\mathbf{r}_k\|$  is the  $k$ th Newton iteration residual norm. The Newton iterations are halted when the Newton update norm plus the residual norm is less than  $10^{-6}$ .

The computations were performed on the Rice Terascale Cluster (RTC), a 1 TeraFLOP Linux cluster based on 900 MHz Intel Itanium2 processors. This distributed memory architecture has 124 nodes with 2 GB memory and 2 processors per node, with 4 additional nodes having 16 GB memory and 4 processors each. Additional information about this machine can be found at <http://www.citi.rice.edu/rtc>.

#### 4.2 SPAI-GMRES( $p$ ) implementation with MPI

The time consuming parts of the computation include: (1) computation of the Jacobian matrix; (2) construction of the inverse preconditioner; (3) GMRES iterations. The memory-intensive storage consists primarily of: (1) elementary Jacobian matrices; (2) Krylov subspace vectors; (3) the preconditioner  $\mathbf{M}$ . The three time consuming parts are parallelized using the Message Passing Interface (MPI) [11]. The entries of the elementary Jacobian matrices, rows of the Krylov subspace, and columns of the preconditioner are uniformly distributed over the processors. The parallel computation proceeds in the following way:

- (1) Each processor computes the elementary Jacobian of its distributed elements, which is the number of total elements divided by the number of processors.

- (2) Each processor assembles its corresponding part of the banded Jacobian matrix from the elementary Jacobian matrices computed in the first step.
- (3) Components of the Jacobian matrix have overlap between the neighboring processors, so the banded Jacobian matrices computed on different processors have to sum up the overlapping part. Each processor sends its banded Jacobian to its neighbors and receives the banded Jacobian from its neighbors, as required for computing preconditioner columns in the next step.
- (4) Each processor computes its distributed columns of the preconditioner (the number of columns is the total number of columns divided by the number of processors).
- (5) During the GMRES iterations, when the algorithm requires the product of the Jacobian matrix and a vector, or the product of the preconditioner and a vector, each processor performs its distributed elementary Jacobian matrix-vector product or its distributed columns of preconditioner-vector product. Each processor communicates with its neighboring processors for the overlapping part of the vector.
- (6) In the Gram–Schmidt orthogonalization procedure, each processor computes the inner product of its corresponding components, which are then summed via the `MPI_allreduce` command.

The sequential and parallel execution times for the entire process are denoted by  $T_s$  and  $T_p$ . Then the parallel speed-up for the entire process,  $S_p$ , is calculated by  $S_p = T_s/T_p$  and the parallel efficiency,  $E_p$ , by  $E_p = S_p/p$ . Similarly, let  $PT_s$  and  $PT_p$  denote the sequential and parallel execution times for the preconditioner. The parallel speed-up for computing the preconditioner,  $PS_p$ , is then given by  $PS_p = PT_s/PT_p$ , and the parallel efficiency,  $PE_p$ , by  $PE_p = PS_p/p$ .

### 4.3 Tests and Results

We now describe the results of seven tests of our preconditioner, the first two of which compare the performance of SPAI-GMRES( $p$ ), a frontal solver, and ILUT-GMRES, and the remaining five of which focus on SPAI-GMRES( $p$ ).

**Test 1:** The first experiment compares memory and CPU requirements of the three solution methods applied to Problem 1. The computation is performed on Mesh 1, with 180 elements and 779 nodes, which produces a problem with 3,656 unknowns. Figure 4 shows Mesh 1 under the defined flow conditions; Table 2 contains the settings of the parameters used for ILUT-GMRES and SPAI-GMRES( $p$ ). The Newton iteration was started from the steady state flow for a fixed domain with a slip wall boundary condition on the free surface section.

**Test 1 Results:** The memory, CPU time to compute the preconditioner, and total CPU time (including all Newton iterations) required by each method are listed in Table 3. GMRES converges with both ILUT and sparse approximate inverse preconditioning. However, because the problem is fairly small the frontal solver has a distinct advantage; the SPAI-based solver only becomes competitive when applied in parallel.

The time spent computing the preconditioner for SPAI-GMRES(1) is 105 sec., a considerable proportion of the total 124 sec. needed to solve the system. Fortunately, the computation of the preconditioner is perfectly parallel, and is thus easily distributed to various processors. Table 3 also shows that for this small case, the parallel speed-up and efficiency are high for the SPAI process and reasonably good for the overall solution.

It is worth mentioning that for this problem, restarted GMRES does not converge without preconditioning (for a variety of restart parameters ranging from 10 to 500) and converges very slowly with ILUT preconditioning for many different values of  $L$ -fill and  $drop$ -tol. (The data reported in Table 3 represent the most favorable combination of parameters we discovered after numerous attempts.)

**Test 2:** The next test compares the performance of the frontal solver, ILUT, and SPAI preconditioning applied to Problem 2 as the computational mesh is refined. The calculations are performed on Mesh 2 and Mesh 3 at  $We = 1$ ; the other flow conditions were defined in Section 3.2. The initial starting point for Newton's method is the result of a viscoelastic free surface flow at  $We = 0.5$  with all the other conditions the same. Table 4 describes the solver parameters used for this test.

### **Test 2 Results:**

Table 5 summarizes the memory usage and execution time for this example. One can see that when the number of unknowns is large, GMRES with the SPAI preconditioner shows a substantial advantage over the frontal solver (FS); here ILUT-GMRES fails to converge. SPAI-GMRES has a much lower memory requirement than the frontal solver and finds the solution much more quickly. When the size of the problem ( $n$ ) increases, both the memory requirements and CPU time increase much more rapidly for the frontal solver than for SPAI-GMRES. The parallel speed-up and the parallel efficiency are high for computing the SPAI preconditioner; they remain high for the entire process provided the number of CPUs is low.

**Test 3:** The construction of the SPAI preconditioner depends on the number of diagonals extracted from the Jacobian matrix. This test investigates the effect of this bandwidth ( $band$ ) on the solution of Problem 2 on Mesh 3; the experimental conditions are the same as in Test 2, except that  $band$  is varied

here.

**Test 3 Results:** Table 6 summarizes the memory usage, total CPU time, and CPU time required to compute the SPAI preconditioner. When too few diagonals are extracted from the Jacobian (e.g.,  $band = 41$ ), the resulting SPAI preconditioner fails to provide an adequate approximation to the inverse of the Jacobian and GMRES does not converge.

On the other hand, for values as small as  $band = 61$  the SPAI preconditioner is sufficiently effective to provide GMRES convergence. Memory requirements do not change much with the growing bandwidth, as the elementary Jacobian and Krylov space dominate the memory usage; the total memory required for the preconditioner and the banded approximation to the Jacobian from which it is derived, is low due to sparsity. However, the CPU time required to compute the preconditioner increases rapidly with  $band$ , degrading the overall performance. Although we have no precise rule for predicting the optimal bandwidth, the tentative choice roughly depends on the number of the unknowns in one element.

Figure 5 compares the CPU times for construction of the SPAI preconditioner and the total overall computation as functions of the bandwidth of the approximate Jacobian. Observe that the former increases linearly with  $band$ , while the latter increases slower than linearly, a result of the decreasing number of GMRES iterations required as the quality of the preconditioner improves with increasing  $band$ . Figure 6 shows that the memory requirements of the total computation grow at a modest linear rate as the bandwidth of the approximate Jacobian grows.

**Test 4:** The previous test fixed the problem size but varied  $band$ . Now we analyze the behavior of SPAI-GMRES(1) as the computational mesh is refined, i.e., as the size of the problem increases. Problem 2 is solved with the same conditions established in Test 2 (using Mesh 2). The mesh is then refined three times in order to produce problems of gradually increasing size. Refer to Table 1 for statistics on Meshes 2–5 used in this experiment.

**Test 4 Results:** Figures 7 and 8 show the CPU time spent computing the preconditioner, CPU time spent on GMRES, the total CPU time for all Newton iterations, and the memory requirement for the preconditioners versus the problem size.

Figure 7 shows that the CPU time spent computing the SPAI preconditioner increases linearly with the problem size, while the CPU time spent on GMRES and the total CPU time appear to increase faster than linear but slower than quadratic. On the other hand, Figure 8 shows that the memory requirement for SPAI increases linearly with the problem size.

**Test 5:** We next investigate the behavior of SPAI-GMRES( $p$ ) on  $p = 16$  processors as the problem size increases. Problem 2 is solved under the same flow conditions as in Test 2 except that  $We = 0.5$ ,  $Krylov-size = 700$ , and  $band = 201$ . (This value of  $Krylov-size$  was sufficient to give convergence for all meshes; smaller values of this parameter would be sufficient for the two smallest meshes.) The initial iterate for Newton’s method is the solution for a Newtonian free surface flow with other conditions identical. Meshes 6–10, described in Table 1, were used for this experiment; the number of unknowns ranged from 165,800 to 1,152,702.

**Test 5 Results:** The CPU time required to compute the preconditioner and the total CPU time for all Newton iterations are plotted against the problem size in Figure 9. This figure shows a roughly linear increase in both quantities.

**Test 6:** We now examine the parallel performance SPAI-GMRES( $p$ ) applied to our finest discretization, Mesh 10 with 1,152,702 unknowns. We use the same flow conditions as in Test 2, except that now  $We = 0.5$ ,  $Krylov-size = 1000$ , and  $band = 201$ . We start Newton’s method with the steady state flow for a Newtonian free surface with the other flow conditions identical.

**Test 6 Results:** Table 7 shows that memory usage and CPU time decrease almost linearly as the number of CPUs increases. The actual parallel speed-up and efficiency cannot be obtained because this large case cannot run on 1 CPU due to memory limitations. Hence we measure the corresponding speed-up based on our results for 16 CPUs as  $\tilde{S}_p = 16 \times T_{16}/T_p$  and  $\tilde{P}\tilde{S}_p = 16 \times PT_{16}/PT_p$  and efficiency as  $\tilde{E}_p = \tilde{S}_p/p$  and  $\tilde{P}\tilde{E}_p = \tilde{P}\tilde{S}_p/p$ . These values are presented in Table 7; they reflect a high parallel efficiency for SPAI ( $> 0.95$ ), a rate sustained over the entire process. Figure 10 reports the CPU time spent computing the preconditioner and the total CPU time for all Newton iterations versus the number of CPUs. The per-processor memory requirements for the preconditioner and the entire process are plotted against the number of CPUs in Figure 11.

**Test 7:** Finally, since our main objective is to provide a scalable linear solver for Newton’s method, we investigate in detail the performance of the SPAI-GMRES preconditioner over all Newton iterations. In particular, we consider the GMRES residual norms for Problem 2 on Mesh 5 (117,516 unknowns) under the same flow conditions as in Test 2 except that full GMRES (i.e., no restarting,  $Krylov-size = n$ ) is used here. We perform the same test with two different stopping rules for GMRES: the first requires the norm of the residual to reach  $10^{-10}$ ; the second uses a convergence criterion of  $10^{-3}$  for the first Newton iteration and  $\min(10^{-7}, \beta\|\mathbf{r}_{k-1}\|^2/\|\mathbf{r}_{k-2}\|^2)$  for the  $k$ th Newton iteration ( $k = 2, 3, \dots$ ). (Recall that this later scheme was used for all the previous test cases described in this section.)

**Test 7 Results:** Figure 12 shows the convergence history for our two different stopping criteria. In the top figure, where the first stopping criterion is applied, full GMRES requires 1693 Krylov vectors to reduce the residual norm eleven orders of magnitude for the first Newton iteration; 1553 Krylov vectors to reduce the residual norm nine orders of magnitude in the second Newton iteration; 1294 Krylov vectors to reduce the residual norm six orders of magnitude in the third Newton iteration, and 771 Krylov vectors to reduce the residual norm two orders of magnitude. In the bottom figure, using the second stopping criterion, GMRES requires 563 Krylov vectors to reduce the residual norm four orders of magnitude in the first Newton iteration, 1159 Krylov vectors to reduce the residual norm six orders of magnitude in the second Newton iteration, and 759 Krylov vectors to reduce the residual norm 3 orders of magnitude in the third Newton iteration. At the fourth iteration the initial residual nearly satisfies the convergence criterion; 30 Krylov vectors are sufficient to improve that residual the small amount needed to achieve convergence.

For both stopping criteria, a total of four Newton iterations are required to obtain residual convergence. When the first stopping criterion is used, the residual norm after each Newton iteration is:  $3.08 \times 10^1$ ,  $1.61 \times 10^{-1}$ ,  $1.28 \times 10^{-4}$ , and  $1.53 \times 10^{-8}$ . We observed similar behavior for all the previous experiments, indicating that quadratic convergence in Newton’s method is preserved.

This experiment demonstrates that the second stopping criterion allows for a significant overall reduction in the number of GMRES iterations while preserving the convergence behavior of Newton’s method. In the above case, SPAI-GMRES(16) spent 2207 sec. on GMRES iterations for the first stopping criterion, but only 764 sec. for the second criterion.

## 5 Summary of results

The experiments detailed in the last section demonstrate that the variant of the sparse approximate inverse preconditioner described in Section 3 enables the scalable solution of large 3D free surface flow problems. From these experiments we conclude that (1) for small problems a frontal solver provides the best performance, followed by ILUT-GMRES, since for these small problems the computation of the SPAI preconditioner is expensive; (2) for problems of larger size (e.g., problems in three dimensions), the performance of both the frontal solver and ILUT-GMRES is not acceptable, as the size of the factors is prohibitive. Indeed, when the size of the problem exceeds (roughly) 50,000, the memory required for the frontal solver can hardly be accommodated. For these larger problems, the SPAI-GMRES implementation displays its advantage, especially when executed in parallel. The efficacy and efficiency of this

preconditioner depends on both its initial sparsity pattern and the number of diagonals of the Jacobian used in the SPAI algorithm. Thus the initial bandwidth should be sufficient to contain the largest entries of the Jacobian, while maintaining a high level of sparsity. The algorithm demonstrates high parallel scalability: CPU time and memory increase almost linearly as the size of the problem increases. Finally, we note that the same preconditioner can be used at each step of the Newton iteration.

## Acknowledgements

The authors thank M. Behr, L. Musson, K. Zygourakis, D. Arora for useful discussions. This work was supported by the National Science Foundation (CTS-CAREER-0134389, DMS-CAREER-0449973, CTS-ITR-0312764, CCR-0306503, ACI-0325081), by the Department of Energy (DE-FG03-02ER-25531), by Los Alamos National Lab (Contract No. 74837-001-0349), by Universidad Central de Venezuela (CDCH: PG-03.00.5579.2004), and by gifts from ExxonMobil and 3M Corporation. Additional support was provided by Texas ATP grant 003604-0011-2001. Computational resources were provided by the supercomputing facility of the Center for Biological and Environmental Nanotechnology supported by the National Science Foundation Nanoscale Science and Engineering Initiative award EEC-0118007 and by the Rice Terascale Cluster funded by NSF under Grant EIA-0216467, Intel, and HP.

## References

- [1] T. A. Baer, R. A. Cairncross, P. R. Schunk, R. R. Rao, and P. A. Sackinger. A finite element method for free surface flows of incompressible fluids in three dimensions. Part II: dynamic wetting lines. *Int. J. Num. Meth. Fluids*, 33:405–427, 2000.
- [2] M. W. Benson and P. O. Frederickson. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Utilitas Math.*, 22:127–140, 1982.
- [3] M. Benzi. Preconditioning techniques for large linear systems: A survey. *J. Comp. Phys.*, 182:418–477, 2002.
- [4] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comp.*, 19:968–994, 1998.
- [5] R. A. Cairncross, P. R. Schunk, T. A. Baer, R. R. Rao, and P. A. Sackinger. A finite element method for free surface flows of incompressible fluids in three dimensions. Part I: boundary fitted mesh motion. *Int. J. Num. Meth. Fluids*, 33:375–403, 2000.

- [6] M. S. Carvalho. *Roll Coating Flows in Rigid and Deformable Gaps*. PhD thesis, University of Minnesota, 1996.
- [7] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comp.*, 19:995–1023, 1998.
- [8] J. D. F Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *Int. J. Comp. Math.*, 44:91–110, 1992.
- [9] V. F. de Almeida. *Gas-Liquid Counterflow Through Constricted Passages*. PhD thesis, University of Minnesota, 1995.
- [10] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.
- [11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 1994.
- [12] M. Grote and H. D. Simon. Parallel preconditioning and approximate inverses on the Connection Machine. In R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, editors, *Parallel Processing for Scientific Computing*, pages 519–523, Philadelphia, 1993. SIAM.
- [13] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comp.*, 18:838–853, 1997.
- [14] R. Guénette and M. Fortin. A new finite element method for computing viscoelastic flows. *J. Non-Newtonian Fluid Mech.*, 60:27–52, 1995.
- [15] R. W. Hooper. *Drop Dynamics in Polymer Processing Flows*. PhD thesis, University of Minnesota, 2001.
- [16] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 1995.
- [17] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings. I: theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.
- [18] L. C. Musson. *Two-layer Slot Coating*. PhD thesis, University of Minnesota, 2001.
- [19] M. Pasquali. *Polymer Molecules in Free Surface Coating Flows*. PhD thesis, University of Minnesota, 2000.
- [20] M. Pasquali and L. E. Scriven. Free surface flows of polymer solutions with models based on the conformation tensor. *J. Non-Newtonian Fluid Mech.*, 108:363–409, 2002.
- [21] M. Renardy, W. J. Hrusa, and J. A. Nohel. *Mathematical Problems in Viscoelasticity*. Wiley, New York, 1987.

- [22] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical report, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois, 1990.
- [23] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.
- [24] M. Szady, T. Salamon, A. Liu, R. Armstrong, and R. Brown. A new mixed finite element method for viscoelastic flows governed by differential constitutive equations. *J. Non-Newtonian Fluid Mech.*, 59:215–243, 1995.
- [25] W. Tang. Toward an effective sparse approximate inverse preconditioner. *SIAM J. Matrix Anal. Appl.*, 20:970–986, 1999.
- [26] E. B. Wilson. *Vector Analysis : A Text-Book for the Use of Students of Mathematics and Physics, Founded Upon the Lectures of J. Willard Gibbs*, volume 1. Yale University Press, New Haven, 1901.
- [27] X. Xie and M. Pasquali. Computing 3-D free surface viscoelastic flows. In A. A. Mammoli and C. A. Brebbia, editors, *Moving Boundaries VII: Computational Modeling of Free and Moving Boundary Problems*, pages 225–234, Southampton, UK, 2003. WIT Press.
- [28] X. Xie and M. Pasquali. A new, convenient way of imposing open-flow boundary conditions in two- and three-dimensional viscoelastic flows. *J. Non-Newtonian Fluid Mech.*, 122:159–176, 2004.

## List of Figures

1	Sparsity pattern for slot-coater problem	21
2	Schematic of Problem 1: Flow downstream of 2D slot coater with free surface.	21
3	Schematic of Problem 2: 3D rod coating flow and boundary conditions, $R_1 = 1$ , $R_2 = 2R_1$ , $L_1 = 2R_1$ , and $L_2 = 6R_1$ .	22
4	Mesh 1 for the 2D slot coater problem used in Test 1.	22
5	Time for constructing the SPAI preconditioner and the total run time versus <i>band</i> for Test 3.	22
6	Total memory versus <i>band</i> : memory increases at a modest linear rate for a fixed problem size in Test 3.	23
7	CPU time versus problem size for SPAI-GMRES(1) applied to Problem 2 on Meshes 2–5.	23

8	Memory requirements for SPAI versus problem size for SPAI-GMRES(1) applied to Problem 2 on Meshes 2–5.	24
9	CPU time versus problem size for SPAI-GMRES(16) on Problem 2.	24
10	Test 6. CPU time for SPAI-GMRES( $p$ ) and the total computation as a function of the number of CPUs for Problem 2 on Mesh 10: $n = 1,152,702$ , $Krylov\text{-size} = 1000$ , and $band = 201$ .	25
11	Test 6. Memory requirements per CPU for the SPAI preconditioner and total computation, as a function of the number of CPUs for SPAI-GMRES( $p$ ) for Problem 2 on Mesh 10: $n = 1,152,702$ , $Krylov\text{-size} = 1000$ , and $band = 201$ .	26
12	Convergence history of GMRES for 3D rod coating flow on Mesh 5. Top: using the first stopping criterion ( $10^{-10}$ ). Bottom: using the second (variable) stopping criterion.	27

## List of Tables

1	Definition of the ten meshes used in the tests. Mesh 1 is used for Problem 1 (2D slot coating flow); the rest are used for Problem 2 (3D rod coating flow).	28
2	Parameter settings for Test 1.	28
3	Comparison of frontal solver, ILUT-GMRES, and SPAI-GMRES( $p$ ) for 2D slot coater (Problem 1) on Mesh 1.	29
4	Parameter settings for Test 2.	29
5	Comparison of frontal solver, ILUT-GMRES, and SPAI-GMRES( $p$ ) for 3D rod coating flow problem (Problem 2) with $We = 1$ and $Ca = 1$ : Frontal solver performance degrades; ILUT-GMRES fails to converge ( $\times$ ); SPAI-GMRES( $p$ ) performs well.	29
6	The effect of $band$ on the performance of SPAI-GMRES(1) for Problem 2 on Mesh 3. (When $band = 41$ , GMRES does not converge.)	30

7 Test 6. Parallel speed-up and efficiency for Problem 2 on Mesh 10:  $n = 1,152,702$ ,  $Krylov\text{-size} = 1000$ , and  $band = 201$ . 30

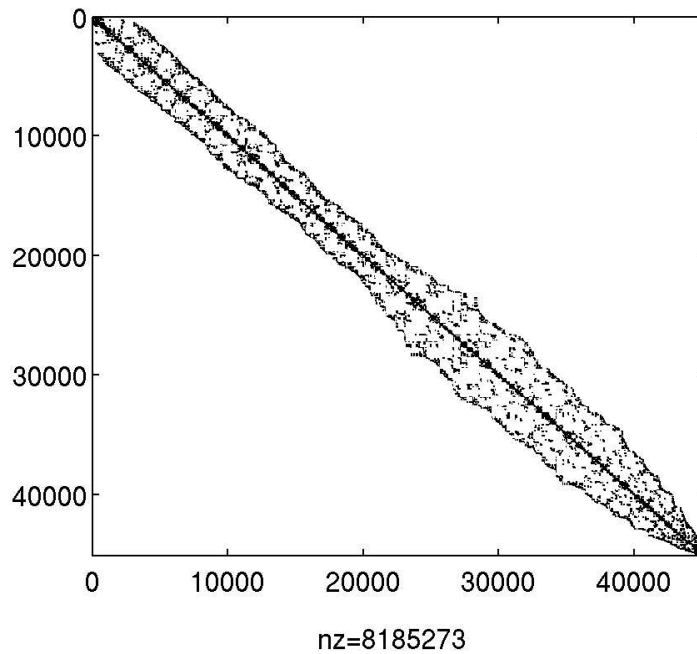


Fig. 1. Sparsity pattern of a Jacobian matrix arising from a 3D rod coating free surface flow (Problem 2, Mesh 2). The matrix is of dimension 45,146 and contains 8,185,273 nonzero entries.

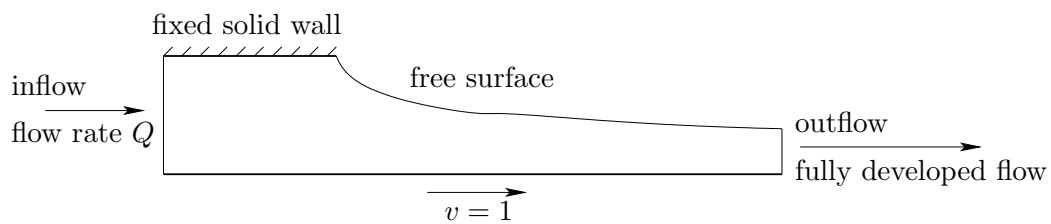


Fig. 2. Schematic of Problem 1: Flow downstream of 2D slot coater with free surface.

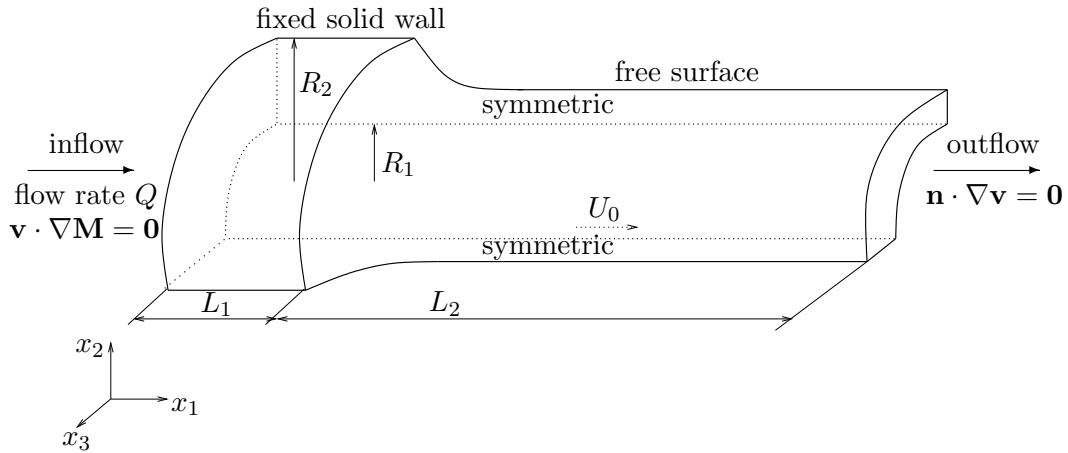


Fig. 3. Schematic of Problem 2: 3D rod coating flow and boundary conditions,  $R_1 = 1$ ,  $R_2 = 2R_1$ ,  $L_1 = 2R_1$ , and  $L_2 = 6R_1$ .

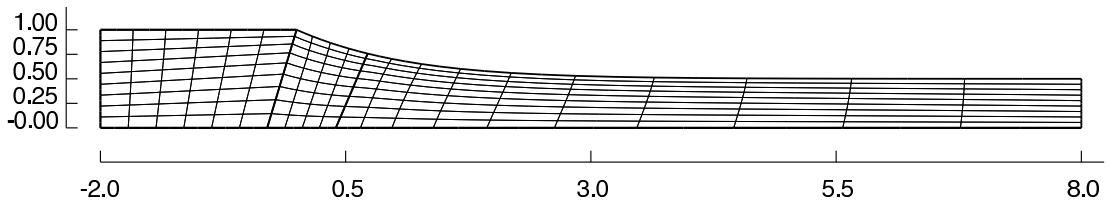


Fig. 4. Mesh 1 for the 2D slot coater problem used in Test 1.

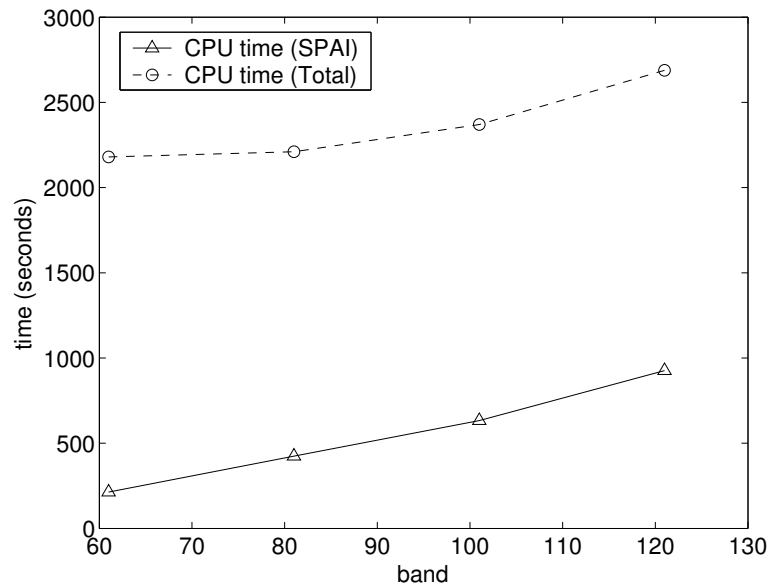


Fig. 5. Time for constructing the SPAI preconditioner and the total run time versus *band* for Test 3.

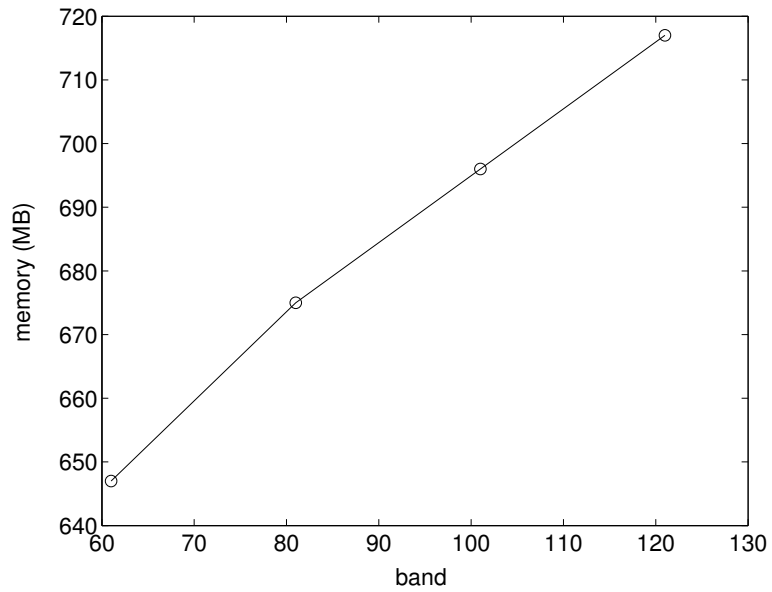


Fig. 6. Total memory versus *band*: memory increases at a modest linear rate for a fixed problem size in Test 3.

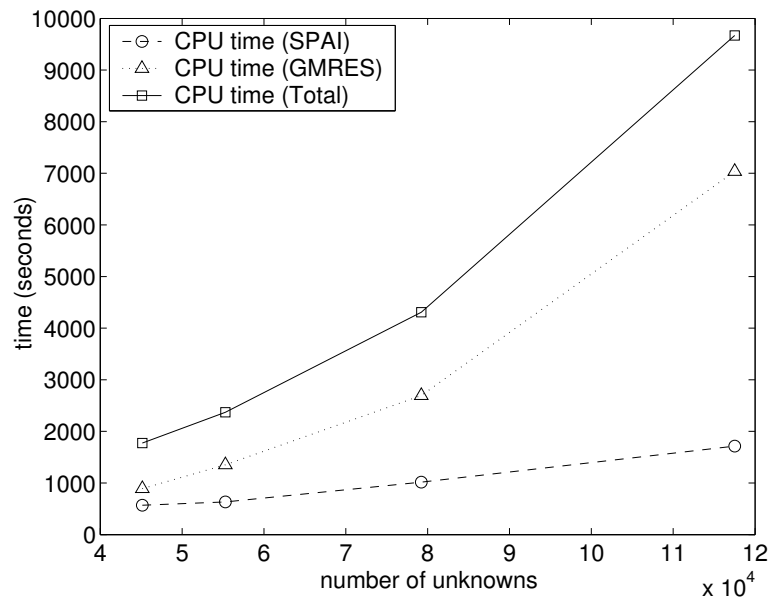


Fig. 7. CPU time versus problem size for SPAI-GMRES(1) applied to Problem 2 on Meshes 2–5.

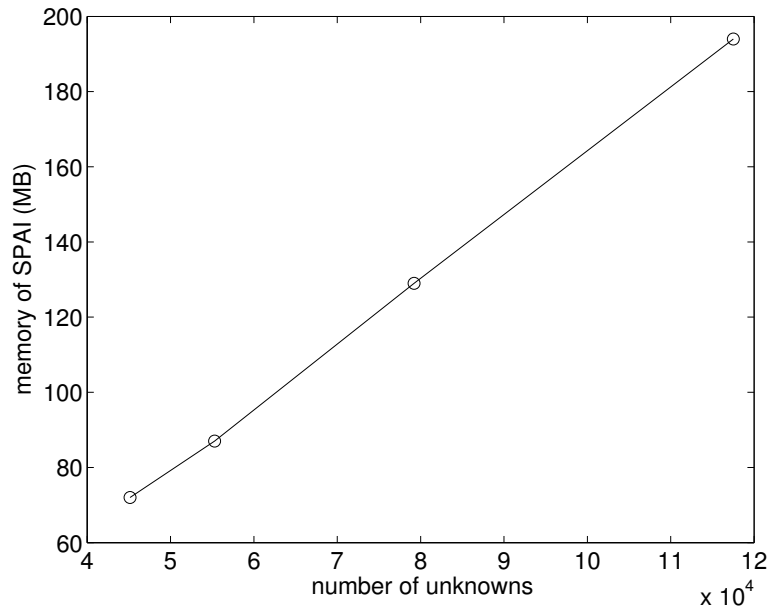


Fig. 8. Memory requirements for SPAI versus problem size for SPAI-GMRES(1) applied to Problem 2 on Meshes 2–5.

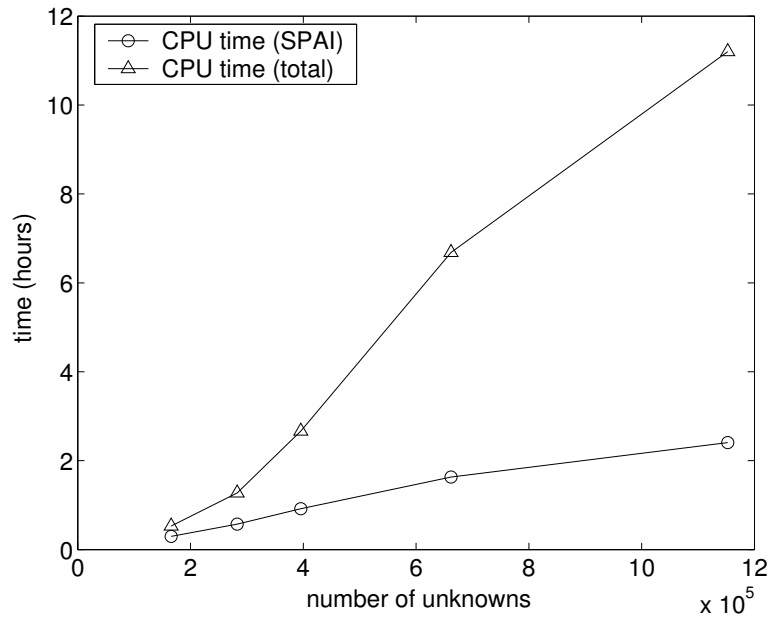


Fig. 9. CPU time versus problem size for SPAI-GMRES(16) on Problem 2.

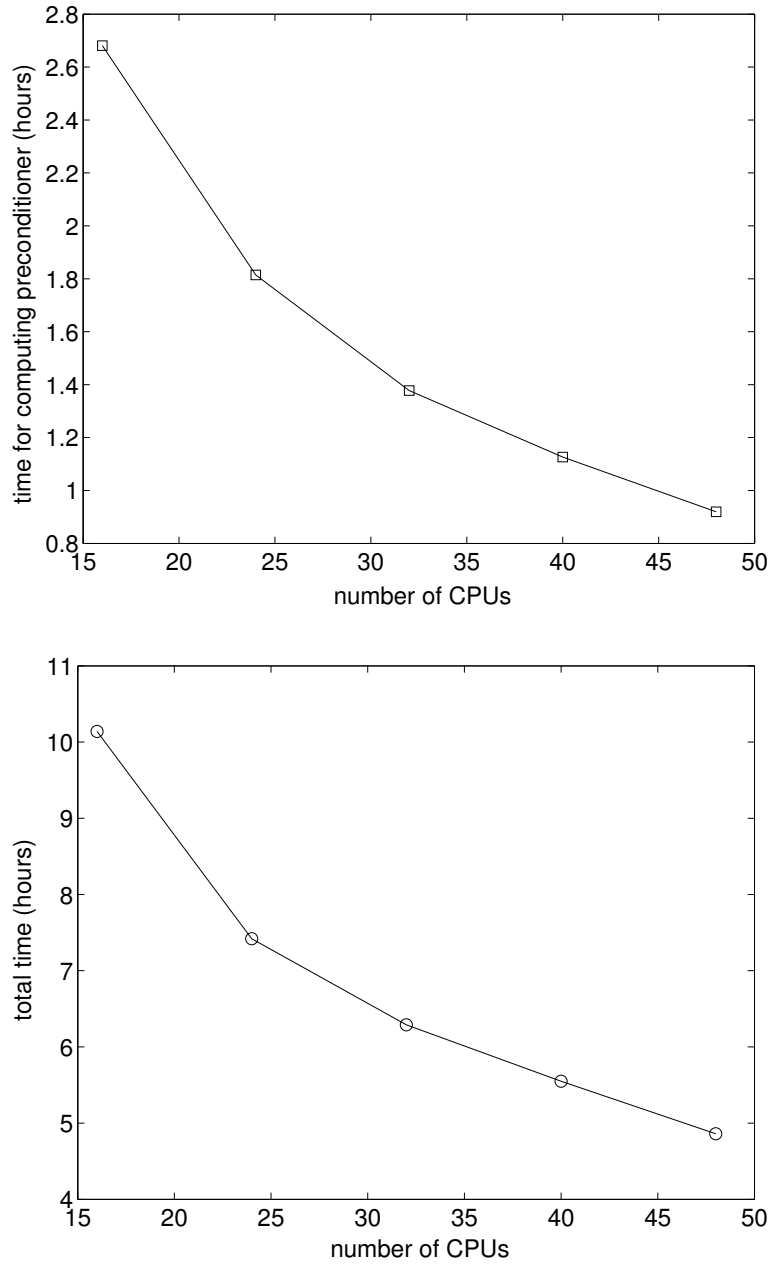


Fig. 10. Test 6. CPU time for SPAI-GMRES( $p$ ) and the total computation as a function of the number of CPUs for Problem 2 on Mesh 10:  $n = 1,152,702$ ,  $Krylov\text{-size} = 1000$ , and  $band = 201$ .

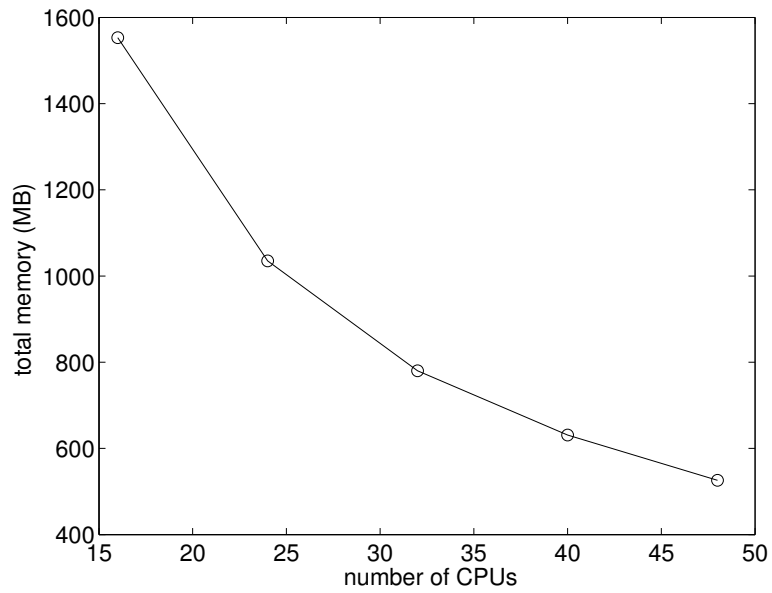
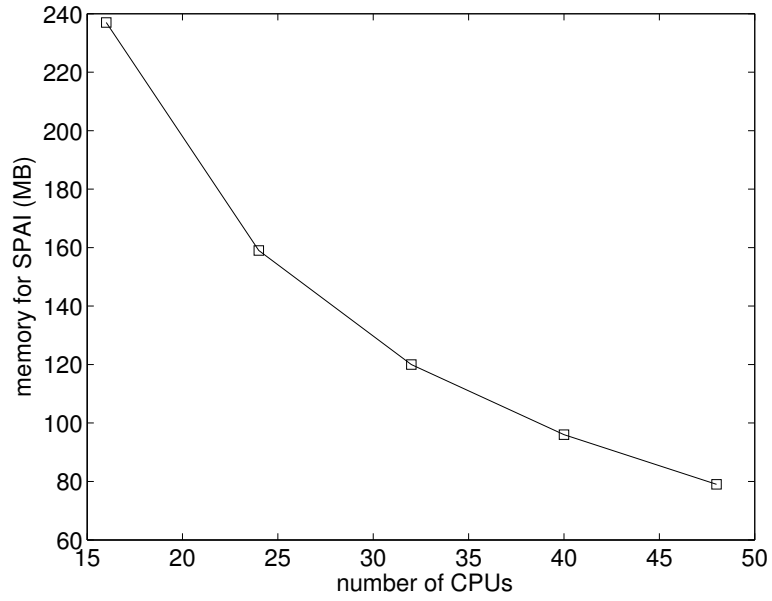


Fig. 11. Test 6. Memory requirements per CPU for the SPAI preconditioner and total computation, as a function of the number of CPUs for SPAI-GMRES( $p$ ) for Problem 2 on Mesh 10:  $n = 1,152,702$ ,  $Krylov-size = 1000$ , and  $band = 201$ .

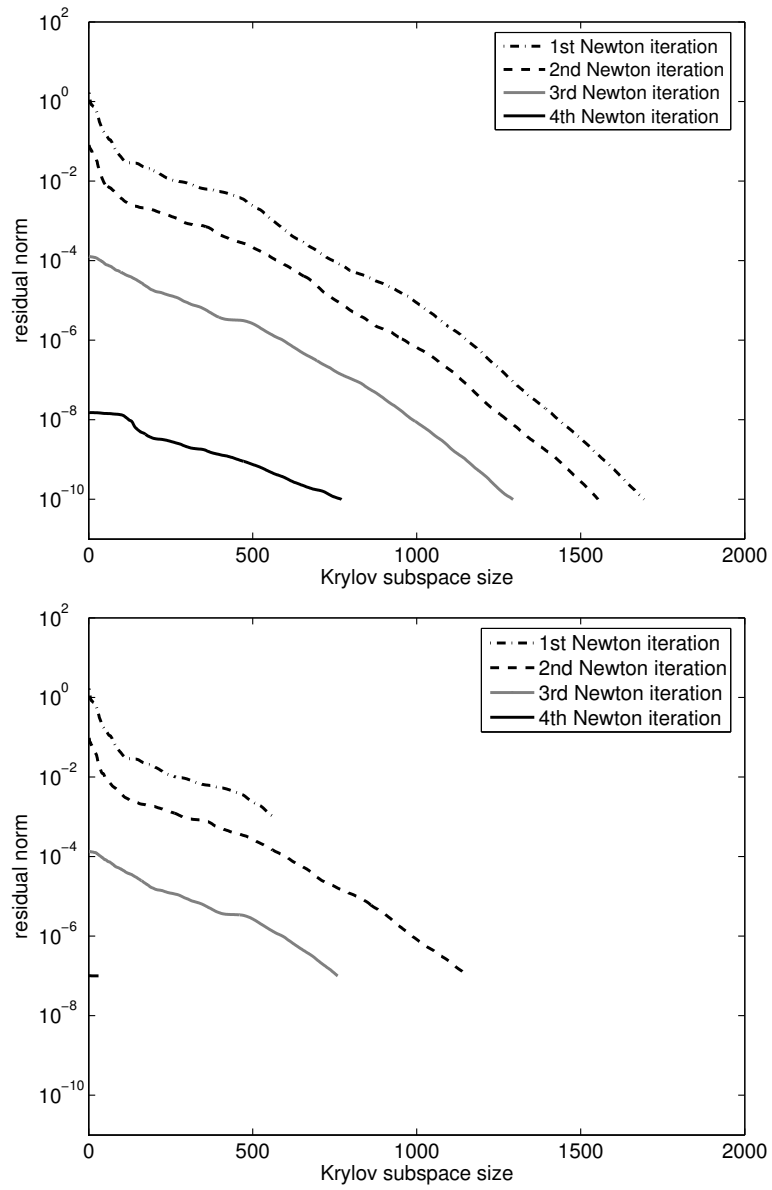


Fig. 12. Convergence history of GMRES for 3D rod coating flow on Mesh 5. Top: using the first stopping criterion ( $10^{-10}$ ). Bottom: using the second (variable) stopping criterion.

Table 1

Definition of the ten meshes used in the tests. Mesh 1 is used for Problem 1 (2D slot coating flow); the rest are used for Problem 2 (3D rod coating flow).

mesh	elements	nodes	unknowns, $n$
1	180	779	3,656
2	2,939	5,271	45,146
3	3,598	6,458	55,292
4	5,368	9,323	79,218
5	8,235	13,906	117,516
6	11,876	19,700	165,800
7	21,059	33,838	282,868
8	29,713	47,402	395,596
9	50,641	79,601	662,006
10	89,133	135,672	1,152,702

Table 2

Parameter settings for Test 1.

$Krylov-size$	$restart$	$L-fill$	$drop-tol$	$band$
100	15	45	$10^{-4}$	101

Table 3

Comparison of frontal solver, ILUT-GMRES, and SPAI-GMRES( $p$ ) for 2D slot coater (Problem 1) on Mesh 1.

solver	memory required	total time	SPAI time	$S_p$	$E_p$	$PS_p$	$PE_p$
Frontal solver	16 MB	11 s	N/A	N/A	N/A	N/A	N/A
ILUT-GMRES	23 MB	23 s	N/A	N/A	N/A	N/A	N/A
SPAI-GMRES(1)	23 MB	124 s	105 s	N/A	N/A	N/A	N/A
SPAI-GMRES(2)	13 MB	75 s	63 s	1.65	0.83	1.67	0.83
SPAI-GMRES(4)	8 MB	46 s	34 s	2.70	0.68	3.09	0.77
SPAI-GMRES(8)	6 MB	31 s	17 s	4.00	0.5	6.18	0.77

Table 4

Parameter settings for Test 2.

<i>Krylov-size</i>	<i>restart</i>	<i>L-fill</i>	<i>drop-tol</i>	<i>band</i>
350	10	150	$10^{-4}$	101

Table 5

Comparison of frontal solver, ILUT-GMRES, and SPAI-GMRES( $p$ ) for 3D rod coating flow problem (Problem 2) with  $We = 1$  and  $Ca = 1$ : Frontal solver performance degrades; ILUT-GMRES fails to converge ( $\times$ ); SPAI-GMRES( $p$ ) performs well.

mesh	solver	memory required	total time	SPAI time	$S_p$	$E_p$	$PS_p$	$PE_p$
Mesh 2 $n = 45,146$	FS	1966 MB	6381 s	N/A	N/A	N/A	N/A	N/A
	ILUT-GMRES	831 MB	$\times$	N/A	N/A	N/A	N/A	N/A
	SPAI-GMRES(1)	573 MB	1773 s	570 s	N/A	N/A	N/A	N/A
Mesh 3 $n = 55,292$	FS	2692 MB	9435 s	N/A	N/A	N/A	N/A	N/A
	ILUT-GMRES	1016 MB	$\times$	N/A	N/A	N/A	N/A	N/A
	SPAI-GMRES(1)	696 MB	2370 s	633 s	N/A	N/A	N/A	N/A
	SPAI-GMRES(2)	349 MB	1315 s	347 s	1.80	0.90	1.82	0.91
	SPAI-GMRES(4)	175 MB	770 s	176 s	3.08	0.77	3.60	0.90
	SPAI-GMRES(8)	91 MB	526 s	89 s	4.51	0.56	7.11	0.89

Table 6

The effect of *band* on the performance of SPAI-GMRES(1) for Problem 2 on Mesh 3. (When *band* = 41, GMRES does not converge.)

<i>band</i>	41	61	81	101	121
memory for SPAI	32 MB	50 MB	72 MB	87 MB	104 MB
total memory	621 MB	647 MB	675 MB	696 MB	717 MB
time for SPAI	60 s	213 s	424 s	633 s	926 s
total time	×	2180 s	2210 s	2370 s	2688 s

Table 7

Test 6. Parallel speed-up and efficiency for Problem 2 on Mesh 10:  $n = 1,152,702$ , *Krylov-size* = 1000, and *band* = 201.

number of CPUs	16	24	32	40	48
memory for SPAI	237 MB	159 MB	120 MB	96 MB	79 MB
total memory	1553 MB	1053 MB	780 MB	631 MB	526 MB
time for SPAI	2.68 h	1.81 h	1.38 h	1.13 h	0.92 h
total time	10.14 h	7.42 h	6.29 h	5.55 h	4.86 h
$\tilde{P}\tilde{S}_p$	16	23.69	31.07	37.95	46.61
$\tilde{P}\tilde{E}_p$	1	0.987	0.971	0.952	0.971
$\tilde{S}_p$	16	21.87	25.79	29.23	33.38
$\tilde{E}_p$	1	0.911	0.806	0.731	0.695