Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 227 (2008) 1790–1808
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier–Stokes equations ☆

Howard Elman [a,*,1], V.E. Howle [b], John Shadid [c], Robert Shuttleworth [d], Ray Tuminaro [e]

[a] *Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, United States*
[b] *Sandia National Laboratories, P.O. Box 969, MS 9159 Livermore, CA 94551, United States*
[c] *Sandia National Laboratories, P.O. Box 5800, MS 1111, Albuquerque, NM 87185, United States*
[d] *Applied Mathematics and Scientific Computing Program and Center for Scientific Computation and Mathematical Modeling, University of Maryland, College Park, MD 20742, United States*
[e] *Sandia National Laboratories, P.O. Box 969, MS 9159, Livermore, CA 94551, United States*

Received 11 April 2007; received in revised form 14 September 2007; accepted 27 September 2007
Available online 12 October 2007

Abstract

In recent years, considerable effort has been placed on developing efficient and robust solution algorithms for the incompressible Navier–Stokes equations based on preconditioned Krylov methods. These include physics-based methods, such as SIMPLE, and purely algebraic preconditioners based on the approximation of the Schur complement. All these techniques can be represented as approximate block factorization (ABF) type preconditioners. The goal is to decompose the application of the preconditioner into simplified sub-systems in which scalable multi-level type solvers can be applied. In this paper we develop a taxonomy of these ideas based on an adaptation of a generalized approximate factorization of the Navier–Stokes system first presented in [A. Quarteroni, F. Saleri, A. Veneziani, Factorization methods for the numerical approximation of Navier–Stokes equations, Computational Methods in Applied Mechanical Engineering 188 (2000) 505–526]. This taxonomy illuminates the similarities and differences among these preconditioners and the central role played by efficient approximation of certain Schur complement operators. We then present a parallel computational study that examines the performance of these methods and compares them to an additive Schwarz domain decomposition (DD) algorithm. Results are presented for two and three-dimensional steady state problems for enclosed domains and inflow/outflow systems on both structured and unstructured meshes. The numerical experiments are performed using MPSalsa, a stabilized finite element code.

☆ This work was partially supported by the DOE Office of Science MICS Program and by the ASC Program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.
* Corresponding author.
E-mail addresses: elman@cs.umd.edu (H. Elman), vehowle@sandia.gov (V.E. Howle), jnshadi@cs.sandia.gov (J. Shadid), rshuttle@math.umd.edu (R. Shuttleworth), rstumin@sandia.gov (R. Tuminaro).
[1] The work of this author was supported by the Department of Energy under Grant DEFG0204ER25619.

0021-9991/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2007.09.026

Keywords: Incompressible flow; Navier–Stokes; Iterative methods

1. Introduction

Current leading-edge engineering and scientific flow simulations often entail complex two and three-dimensional geometries with high resolution unstructured meshes to capture all the relevant length scales of interest. After suitable discretization and linearization these simulations can produce large linear systems of equations with on the order 10^5–10^8 unknowns. As a result, efficient and scalable parallel iterative solution methods are required. We consider solution methods for the incompressible Navier–Stokes equations where the equations below represent conservation of momentum and mass, and the constitutive equation for the Newtonian stress tensor,

$$\begin{aligned} &\text{Momentum}: \quad \rho(\mathbf{u}\cdot\nabla)\mathbf{u} = \nabla\cdot\mathbf{T} + \rho\mathbf{g},\\ &\text{Mass}: \quad \nabla\cdot\mathbf{u} = 0,\\ &\text{Stress tensor}: \quad \mathbf{T} = -P\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^{\mathrm{T}}) \end{aligned} \tag{1}$$

in $\Omega \subset \mathbb{R}^d$ ($d = 2$ or 3). Here the velocity, $\mathbf{u}$, satisfies suitable boundary conditions on $\partial\Omega$, P represents the hydrodynamic pressure, ρ the density, μ the dynamic viscosity, and $\mathbf{g}$ the body forces.

We focus on solution algorithms for the algebraic system of equations that result from linearization and discretization of these equations. The coefficient matrices have the general form

$$A = \begin{pmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{pmatrix}. \tag{2}$$

The strategies we employ for solving (2) are derived from the LDU block factorization of this coefficient matrix,

$$A = \begin{pmatrix} I & 0 \\ \widehat{B}F^{-1} & I \end{pmatrix} \begin{pmatrix} F & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & F^{-1}B^{\mathrm{T}} \\ 0 & I \end{pmatrix}, \tag{3}$$

where

$$S = C + \widehat{B}F^{-1}B^{\mathrm{T}} \tag{4}$$

is the Schur complement (of F in A). They require methods for approximating the action of the inverse of the factors of (3), which, in particular, requires approximation to the actions of F^{-1} and S^{-1}. For large-scale computations, use of the exact Schur complement is not feasible. Therefore effective approximate block factorization (ABF) preconditioners are often based on a careful consideration of the spectral properties of the component block operators and the approximate Schur complement operators. There has been a great deal of recent work on ABF methods (e.g. [3,4,6,8,17]). These techniques take a purely linear algebraic view of preconditioning. Through these decompositions a simplified system of block component equations is developed that encodes a specific "physics-based" decomposition. Alternatively, one could start with "physics-based" iterative solution methods for the Navier–Stokes equations (e.g. [21,22]) and develop preconditioners based on these techniques as described in [18]. In both these cases, the system has been transformed by the factorization into component systems that are essentially convection–diffusion and Poisson type operators. The result is a system to which multi-level methods, and in our particular case, algebraic multi-level methods (AMG) can be applied successfully for parallel unstructured mesh simulations.

In this paper, we include preconditioners developed from iterative solution strategies based on pressure correction methods, like SIMPLE, proposed by Patankar and Spaulding [22] and previously studied as a preconditioner by Pernice and Tocci in [23]. We interpret these methods in the context of our taxonomy and compare them with some new "approximate commutator methods". These techniques are based on the approximation

of the Schur complement operator by a technique proposed by Kay et al. [17], Silvester et al. [29], and Elman et al. [10].

The paper is organized as follows. Section 2 gives a brief description of the Newton iteration and provides an overview of the discretization and resulting coefficient matrix used for our numerical experiments. Section 3 presents a taxonomy for classifying the approximate block preconditioners. Section 4 provides a brief overview of the parallel implementation of the nonlinear and linear solvers. Details of the numerical experiments and the results of these experiments are described in Section 5. Concluding remarks are provided in Section 6.

2. Background

Our focus is on solution algorithms for the systems of equations that arise after discretization and linearization of the system (1). A nonlinear iteration based on an inexact Newton–Krylov method is used to solve this problem. If the nonlinear problem to be solved is written as $G(x) = 0$, where $G : \mathbb{R}^n \to \mathbb{R}^n$, then at the kth step of Newton's method, the solution of the linear *Newton equation*,

$$J(x_k)s_k = -g(x_k) \tag{5}$$

is required, where x_k is the current solution and $J(x_k)$ denotes the Jacobian matrix of G at x_k. Once the Newton update, s_k, is determined, the current approximation is updated via

$$x_{k+1} = x_k + s_k.$$

Newton–Krylov methods [7] relax the requirement to compute an exact solution to (5). Instead, a Krylov subspace method, such as GMRES, is applied until an iterate s_k is found that satisfies the *inexact Newton condition*,

$$\|g(x_k) + J(x_k)s_k\| \leqslant \eta_k \|g(x_k)\|, \tag{6}$$

where, $\eta_k \in [0, 1]$, is a tolerance. If $\eta_k = 0$, this is an exact Newton method. For a discussion of the merits of different choices of η_k, see [7]. For this computational study, η_k is chosen to be a constant and our attention is focused on preconditioning methods for use with GMRES solving for the Newton update.

For the discrete Navier–Stokes equations, the Jacobian system at the kth step that arises from Newton's method is

$$\begin{pmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{pmatrix} \begin{pmatrix} \mathbf{\Delta u_k} \\ \Delta p_k \end{pmatrix} = \begin{pmatrix} \mathbf{g_u^k} \\ g_{\mathrm{p}}^k \end{pmatrix}, \tag{7}$$

where F is a convection–diffusion-like operator, B^{T} is the gradient operator, $\widehat{B}$ is the divergence operator that for some higher-order stabilized formulations can include a contribution from non-zero higher-order derivative operators in the stabilized formulation [5], and C is the operator that stabilizes the finite element discretization. The right-hand side vector, $(\mathbf{g_u}, g_p)^{\mathrm{T}}$, contains, respectively, the nonlinear residual for the momentum and continuity equations. This Newton procedure starts with some initial iterate $\mathbf{u}_0$ for the velocities, p_0 for the pressure; then updates for the velocities and pressures are computed by solving the Newton equations (7). Further details on the particular discretization used in our experiments can be found in Section 4.

All of the methods we describe in Section 3 generate some approximation, $\widetilde{A}$, to the Jacobian system found in (5). Some of the methods considered in this paper have been traditionally derived as stationary iterative solvers, and we use this mode of description in parts of the paper. That is, for

$$A\mathbf{u} = \mathbf{f} \tag{8}$$

stationary iterations have the form

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \widetilde{A}^{-1}(\mathbf{f} - A\mathbf{u}_n)$$

where $\widetilde{A}$ is a Jacobian approximation, $A = \widetilde{A} - E$, and $\mathbf{u}_n$ is the approximate solution at the nth iteration. All of our experiments use the splitting operators as preconditioners in a Krylov subspace method.

3. Taxonomy of approximate block factorization preconditioners

We adopt a nomenclature developed by Quarteroni et al. [25] for algebraic splittings of A for projection type methods. Let H_1 represent an approximation to F^{-1} in the Schur complement (4) and let H_2 be an approximation to F^{-1} in the upper triangular block of the factorization (3). This results in the following decomposition:

$$A_s = \begin{bmatrix} I & 0 \\ \widehat{B}F^{-1} & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & -(C + \widehat{B}H_1B^{\mathrm{T}}) \end{bmatrix} \begin{bmatrix} I & H_2B^{\mathrm{T}} \\ 0 & I \end{bmatrix} = \begin{bmatrix} F & FH_2B^{\mathrm{T}} \\ \widehat{B} & -(C + \widehat{B}(H_1 - H_2)B^{\mathrm{T}}) \end{bmatrix}. \tag{9}$$

The error matrix, $E_s = A - A_s$ is

$$E_s = \begin{bmatrix} 0 & (I - FH_2)B^{\mathrm{T}} \\ 0 & -\widehat{B}(H_2 - H_1)B^{\mathrm{T}} \end{bmatrix}.$$

This decomposition is used in [25] to illuminate the structure of several projection techniques for solving the time-dependent Navier–Stokes equations. By examining the error, we can determine which equation (momentum or continuity) in the original problem is perturbed by the approximations H_1 or H_2. For example, if $H_1 = F^{-1}$ and $H_1 \neq H_2$, then the operators applied to the pressure in both the momentum equation and continuity equation are perturbed, whereas operators applied to the velocity are not perturbed. On the other hand, if $H_2 = F^{-1}$ and $H_1 \neq H_2$, then the (1, 2) block of the error matrix is zero. So, the momentum equation is unperturbed, thus giving a "momentum preserving strategy", whereas a perturbation of the incompressibility constraint occurs [25]. If $H_1 = H_2 \neq F^{-1}$, then the scheme is "mass preserving" because the (2,2) block of the error matrix is zero, so the continuity equation is not modified. Finally, if $H_1 \neq H_2 \neq F^{-1}$, then both the momentum and continuity equations are modified.

The above factorization can be generalized to incorporate "classical" methods used for these problems such as SIMPLE, SIMPLEC, SIMPLER [22,23], as well as newer approximate commutator methods devised to generate good approximations to the Schur complement [17,29]. Let us modify (9) using some approximation H_1 in place of F^{-1} in the lower triangular block. In addition, let $\hat{\mathrm{S}}$ represent an approximation of the Schur complement. This gives

$$\widetilde{A} = \begin{bmatrix} I & 0 \\ \widehat{B}H_1 & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & -\widehat{S} \end{bmatrix} \begin{bmatrix} I & H_2B^{\mathrm{T}} \\ 0 & I \end{bmatrix} = \begin{bmatrix} F & FH_2B^{\mathrm{T}} \\ \widehat{B}H_1F & \widehat{B}H_1FH_2B^{\mathrm{T}} - \widehat{S} \end{bmatrix}. \tag{10}$$

The error, denoted $\widetilde{E} = A - \widetilde{A}$, is

$$\widetilde{E} = \begin{bmatrix} 0 & B^{\mathrm{T}} - FH_2B^{\mathrm{T}} \\ \widehat{B} - \widehat{B}H_1F & \widehat{S} - (C + \widehat{B}H_1FH_2B^{\mathrm{T}}) \end{bmatrix}.$$

Techniques explored in this study can be classified into two categories: those whose factorization groups the lower triangular and the diagonal components as $[(LD)U]$, and those that group the diagonal and lower triangular components as $[L(DU)]$. Methods with the $(LD)U$ grouping have the factorization

$$\widetilde{A}_{(LD)U} = \begin{bmatrix} F & 0 \\ \widehat{B}H_1F & -\widehat{S} \end{bmatrix} \begin{bmatrix} I & H_2B^{\mathrm{T}} \\ 0 & I \end{bmatrix}. \tag{11}$$

Methods with the $L(DU)$ grouping have the factorization

$$\widetilde{A}_{L(DU)} = \begin{bmatrix} I & 0 \\ \widehat{B}H_1 & I \end{bmatrix} \begin{bmatrix} F & FH_2B^{\mathrm{T}} \\ 0 & -\widehat{S} \end{bmatrix}. \tag{12}$$

Some of the techniques considered do not use the complete factorization (11) or (12), but rather use only triangular components of the factorization. SIMPLE uses the block $(LD)U$ grouping. The approximate commutator methods are derived from the block $L(DU)$ grouping and use just the diagonal and upper triangular

(DU) components in the method. Finally, these classifications are further refined by specifying strategies for approximating the Schur complement.

3.1. Pressure correction

The pressure correction family of Navier–Stokes preconditioners is derived from the divergence free constraint with decoupling of the incompressible Navier–Stokes equations. In the following sections, three pressure correction methods are derived, SIMPLE (Semi-Implicit Method for Pressure Linked Equations), SIMPLEC, and SIMPLER (Semi-Implicit Method for Pressure Linked Equations Revised) [22–24].

3.1.1. The SIMPLE preconditioner

The SIMPLE-like algorithm described here begins by solving a variant of the momentum equation for an intermediate velocity using a previously generated pressure; then the continuity equation is solved using the intermediate velocity to calculate the pressure update. This value is used to update the velocity component. The SIMPLE algorithm expressed as a stationary iteration is as follows:

1. Solve: $F\mathbf{u}_{n+\frac{1}{2}} = f - B^{\mathrm{T}}p_n$ for the velocity, $\mathbf{u}$.
2. Solve: $-(C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}})\delta p = \widehat{B}\mathbf{u}_{n+\frac{1}{2}} + Cp_n$ for δp.
3. Calculate the velocity correction: $\delta\mathbf{u} = \mathbf{u}_{n+1} - \mathbf{u}_{n+\frac{1}{2}} = (-\mathrm{diag}(F)^{-1}B^{\mathrm{T}})\delta p$.
4. Update the pressure: $p_{n+1} = p_n + \alpha\delta p$
5. Update the velocity: $\mathbf{u}_{n+1} = \mathbf{u}_{n+\frac{1}{2}} + \delta\mathbf{u}$

The quantity α is a parameter in $(0, 1]$ that damps the pressure update. An alternative derivation is obtained using the LDU framework described above. The block lower triangular factor (L) and the block diagonal (D) are grouped together. In terms of the taxonomy described above, this corresponds to the choices $H_1 = F^{-1}$, $H_2 = (\mathrm{diag}(F))^{-1}$, and $\widehat{S} = C + \widehat{B}(\mathrm{diag}(F))^{-1}B^{\mathrm{T}}$ in (11). The decomposition is

$$\begin{bmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \widehat{B}F^{-1} & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & \widehat{S} \end{bmatrix} \begin{bmatrix} I & (\mathrm{diag}(F))^{-1}B^{\mathrm{T}} \\ 0 & \alpha I \end{bmatrix} = \begin{bmatrix} F & 0 \\ \widehat{B} & -\widehat{S} \end{bmatrix} \begin{bmatrix} I & (\mathrm{diag}(F))^{-1}B^{\mathrm{T}} \\ 0 & \alpha I \end{bmatrix} = \widetilde{A}_{\mathrm{SIMPLE}}.$$

Thus, one iteration of SIMPLE corresponds to

$$\begin{bmatrix} \mathbf{u}_{n+1} \\ p_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} + \widetilde{A}^{-1}_{\mathrm{SIMPLE}} \left(\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} - A \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} \right)$$

where A is defined in (2). The error for this method (when $\alpha = 1$) is

$$E_{\mathrm{SIMPLE}} = A - \widetilde{A}_{\mathrm{SIMPLE}} = \begin{bmatrix} 0 & B^{\mathrm{T}} - F(\mathrm{diag}(F))^{-1}B^{\mathrm{T}} \\ 0 & 0 \end{bmatrix}.$$

SIMPLE does not affect the terms that operate on the velocity, but it perturbs the pressure operator in the momentum equation. This results in a method that is "mass preserving". When $\mathrm{diag}(F)^{-1}$ is a good approximation to F^{-1}, then E_{SIMPLE} is close to a zero matrix, so this method generates a very close approximation to the original Jacobian system. From our computational experiments in Section 5, we have found that the diagonal approximation can yield poor results because the diagonal approximation does not capture enough information about the convection operator.

3.1.2. The SIMPLEC preconditioner

The SIMPLEC algorithm is a variant of SIMPLE [23]. It replaces the diagonal approximation of the inverse of F with the diagonal matrix whose entries contain the absolute value of the row sums of F. The matrix structure is the same $(LD)U$ as that of SIMPLE. The symbol $\sum(|F|)$ denotes a matrix whose entries are equal to the absolute value of the row sum of F. With the choices $H_1 = F^{-1}$, $H_2 = (\sum|F|)^{-1}$, and

$\widehat{S} = C + \widehat{B}(\sum |F|)^{-1}B^{\mathrm{T}}$, the SIMPLEC method can be expressed in terms of the block factorization (11). The decomposition is

$$\begin{bmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{bmatrix} \approx \begin{bmatrix} F & 0 \\ \widehat{B} & -\widehat{S} \end{bmatrix} \begin{bmatrix} I & (\sum(|F|)^{-1}B^{\mathrm{T}} \\ 0 & \alpha I \end{bmatrix} = \widetilde{A}_{\mathrm{SIMPLEC}}$$

where $\widehat{S} = C + \widehat{B}(\sum |F|)^{-1}B^{\mathrm{T}}$. The error for this method (when $\alpha = 1$) is

$$E_{\mathrm{SIMPLEC}} = A - \widetilde{A}_{\mathrm{SIMPLEC}} = \begin{bmatrix} 0 & B^{\mathrm{T}} - F(\sum |F|)^{-1}B^{\mathrm{T}} \\ 0 & -\widehat{B}(\sum |F|)^{-1}B^{\mathrm{T}} + \widehat{B}F^{-1}B^{\mathrm{T}} \end{bmatrix}.$$

This method perturbs the pressure operator in both the momentum and continuity equations. The choice of the absolute value of the row sum tends to provide a better approximation to the matrix F, therefore reducing the error associated with this method [23]. We have found that this choice works reasonably well and is easy to construct. Further variations of this class of methods can be determined by choosing different approximations to F^{-1}, such as sparse approximate inverses. For our computational results in Section 5, we use the absolute value of the row sum.

3.1.3. The SIMPLER preconditioner

The SIMPLER algorithm is very similar to SIMPLE, except that it first determines $\hat{p}_{n+1}$ using $\mathbf{u}_n$, then it calculates an intermediate velocity value, $\mathbf{u}_{n+\frac{1}{2}}$. This intermediate velocity is projected to enforce the continuity equation, which determines $\mathbf{u}_{n+1}$. The steps required are as follows:

1. Solve: $(C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}})\hat{p}_{n+1} = -\widehat{B}\mathrm{diag}(F)^{-1}(f + F\mathbf{u}_n - B^T p_n)$ for the pressure, $\hat{p}_{n+1}$.
2. Solve: $F\mathbf{u}_{n+\frac{1}{2}} = f - B^{\mathrm{T}}(\hat{p}_{n+1} - p_n)$ for the velocity, $\mathbf{u}$.
3. Project $\mathbf{u}_{n+\frac{1}{2}}$ to obtain $\mathbf{u}_{n+1} = [I + (\mathrm{diag}(F)^{-1})^{-1}\widehat{B}(C + B\mathrm{diag}(F)^{-1}B^{\mathrm{T}})^{-1}B^{\mathrm{T}})]\mathbf{u}_{n+\frac{1}{2}}$
4. Update the pressure: $p_{n+1} = \alpha\hat{p}_{n+1}$

Once again, α is a parameter in (0, 1] that damps the pressure update. SIMPLER can also be expressed using the LDU framework. The block diagonal (D) and the block upper triangular (U) factors are grouped together and an additonal matrix, P, a projection matrix for the velocity projection in step 3, is added to the factorization.

In terms of the taxonomy, this corresponds to the choices of $H_1 = \mathrm{diag}\ (F)^{-1}$, $H_2 = F^{-1}$, and $\widehat{S} = C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}}$ in (12). Then

$$\begin{bmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \widehat{B}F^{-1} & I \end{bmatrix} \begin{bmatrix} F & B^{\mathrm{T}} \\ 0 & S \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \widehat{B}(\mathrm{diag}(F))^{-1} & I \end{bmatrix} \begin{bmatrix} F & B^{\mathrm{T}} \\ 0 & -\widehat{S} \end{bmatrix}$$

where $\widehat{S} = C + \widehat{B}(\mathrm{diag}(F))^{-1}B^{\mathrm{T}}$. Now, the projection matrix is added to give the SIMPLER algorithm in matrix form. This results in

$$\widetilde{A}_{\mathrm{SIMPLER}} = \begin{bmatrix} I + (\mathrm{diag}(F))^{-1}\widehat{B}\widehat{S}^{-1}B^{\mathrm{T}} & 0 \\ 0 & \alpha I \end{bmatrix} \begin{bmatrix} I & 0 \\ \widehat{B}(\mathrm{diag}(F))^{-1} & I \end{bmatrix} \begin{bmatrix} F & B^{\mathrm{T}} \\ 0 & -\widehat{S} \end{bmatrix} \tag{13}$$

[23]. Thus, one iteration of SIMPLER corresponds to

$$\begin{bmatrix} \mathbf{u}_{n+1} \\ p_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} + \widetilde{A}^{-1}_{\mathrm{SIMPLER}} \left(\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} - A \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} \right)$$

where A is defined in (2) and $\widetilde{A}_{\mathrm{SIMPLER}}$ is defined in (13). The use of the projection matrix, which has subsidiary solves that must be performed to very high accuracy, greatly degrades the performance of this method when compared to SIMPLE. However, the projection matrix is needed to enforce the continuity equation, and therefore produce a solution that is divergence free [23]. This method perturbs the pressure operator in both the momentum and continuity equations.

3.1.4. Remarks on pressure correction methods

In this section, the pressure correction methods (SIMPLE/SIMPLEC) that begin with the underlying factorization, $(LD)U$ and use approximations to the components of the factors to define the preconditioner have been given. SIMPLER is based on the decomposition $L(DU)$ with approximations to $P^{-1}(DU)^{-1}L^{-1}$ as the preconditioner. These methods are useful for steady-state flow problems. However, these methods tend to converge slowly and require the user to input a relaxation parameter to improve convergence.

3.2. Approximate commutator methods

The pressure convection–diffusion preconditioners group together the diagonal and upper triangular factors and omit the lower triangular factor. Let $H_1 = H_2 = F^{-1}$. Then the block factorization of the coefficient matrix is

$$\begin{pmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ \widehat{B}H_1 & I \end{pmatrix} \begin{pmatrix} F & FH_2B^{\mathrm{T}} \\ 0 & -S \end{pmatrix} = \begin{pmatrix} I & 0 \\ \widehat{B}F^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^{\mathrm{T}} \\ 0 & -S \end{pmatrix}. \tag{14}$$

where the diagonal (D) and upper triangular (U) factors are grouped together. For our computations, we only use the upper triangular factor, and replace the Schur complement S by some approximation $\hat{\mathrm{S}}$ (to be specified later). The efficacy of this strategy can be seen by analyzing the following generalized eigenvalue problem:

$$\begin{pmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & B^{\mathrm{T}} \\ 0 & \widehat{S} \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}.$$

If $\hat{\mathrm{S}}$ is the Schur complement, then all the eigenvalues of the preconditioned matrix are identically one. This operator contains Jordan blocks of dimension at most 2, and consequently at most two iterations of a preconditioned GMRES iteration would be needed to solve the system [20].

We motivate the Approximate Commutator Methods by examining the computational issues associated with applying this preconditioner $\mathcal{Q}$ in a Krylov subspace iteration. At each step, the application of $\mathcal{Q}^{-1}$ to a vector is needed. By expressing this operation in factored form,

$$\begin{pmatrix} F & B^{\mathrm{T}} \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^{\mathrm{T}} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -S^{-1} \end{pmatrix}$$

two potentially difficult operations can be seen: S^{-1} must be applied to a vector in the discrete pressure space, and F^{-1} must be applied to a vector in the discrete velocity space. The application of F^{-1} can be performed relatively cheaply using an iterative technique, such as multigrid. However applying S^{-1} to a vector is too expensive. An effective preconditioner can be built by replacing this operation with an inexpensive approximation. We discuss three preconditioning strategies, the pressure convection–diffusion (P–CD), the Least Squares Commutator (LSC), and the approximate SIMPLE commutator (ASC).

3.2.1. The pressure convection–diffusion (PCD) preconditioner

Pressure convection–diffusion preconditioners take a fundamentally different approach to approximate the inverse Schur complement than SIMPLE. The basic idea hinges on the notion of an approximate commutator. Consider a discrete version of the convection–diffusion operator,

$$(\nu\nabla^2 + (\mathbf{w}\cdot\mathrm{grad})) \tag{15}$$

where $\mathbf{w}$ is a constant vector. When $\mathbf{w}$ is an approximation to the velocity obtained from the previous nonlinear step, (15) is an Oseen linearization of the nonlinear term in (1). Suppose there is an analogous operator defined on the pressure space,

$$(\nu\nabla^2 + (\mathbf{w}\cdot\mathrm{grad}))_{\mathrm{p}}$$

Consider the commutator of these operators with the gradient:

$$\epsilon = (v\nabla^2 + (\mathbf{w}\cdot\mathrm{grad}))\nabla - \nabla(v\nabla^2 + (\mathbf{w}\cdot\mathrm{grad}))_{\mathrm{p}} \tag{16}$$

Supposing that ϵ is small, multiplication on both sides of (16) by the divergence operator gives

$$\nabla^2(v\nabla^2 + (\mathbf{w}\cdot\mathrm{grad}))_{\mathrm{p}}^{-1} \approx \nabla\cdot(v\nabla^2 + (\mathbf{w}\cdot\mathrm{grad}))^{-1}\nabla \tag{17}$$

In discrete form, using finite elements, this usually takes the form,

$$(Q_{\mathrm{p}}^{-1}A_{\mathrm{p}})(Q_{\mathrm{p}}^{-1}F_{\mathrm{p}})^{-1} \approx (Q_{\mathrm{p}}^{-1}B)(Q_{\mathrm{v}}^{-1}F)^{-1}(Q_{\mathrm{v}}^{-1}B^{\mathrm{T}})$$
$$A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}} \approx BF^{-1}B^{\mathrm{T}}$$

where here F represents a discrete convection–diffusion operator on the velocity space, F_{p} is the discrete convection–diffusion operator on the pressure space, A_{p} is a discrete Laplacian operator, Q_{v} the velocity mass matrix, and Q_{p} is the lumped pressure mass matrix. This suggests the approximation for the Schur complement

$$S \approx \widehat{S} = A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}} \tag{18}$$

for a stable finite element discretization when $C = 0$. In the case of our pressure stabilized finite element discretizations, the same type of approximation is required [8]:

$$S = C + \widehat{B}F^{-1}B^{\mathrm{T}} \approx A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}}. \tag{19}$$

Applying the action of the inverse of $A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}}$ to a vector requires solving a system of equations with a discrete Laplacian operator, then multiplication by the matrix F_{p}, and solving a system of equations with the pressure mass matrix. In practice, Q_{p} can be replaced by its lumped approximation with little deterioration of effectiveness. Both the convection–diffusion-like system, F, and the Laplace system, A_{p}, can also be handled using multigrid with little deterioration of effectiveness.

In our taxonomy, the pressure convection–diffusion method is generated by grouping together the upper triangular and diagonal factors as in (12), choosing $H_2 = F^{-1}$ and $\hat{\mathrm{S}}$ as in (19). In matrix form this is

$$\widetilde{A}_{\mathrm{PCD}} = \begin{bmatrix} F & FH_2B^{\mathrm{T}} \\ 0 & -\widehat{S} \end{bmatrix} = \begin{bmatrix} F & B^{\mathrm{T}} \\ 0 & -A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}} \end{bmatrix}.$$

The error matrix is

$$E_{\mathrm{PCD}} = A - \widetilde{A}_{\mathrm{PCD}} = \begin{bmatrix} 0 & 0 \\ \hat{B} & A_{\mathrm{p}}F_{\mathrm{p}}^{-1}Q_{\mathrm{p}} - C \end{bmatrix},$$

which shows that the momentum equation is unperturbed and only the pressure operator in the continuity equation is perturbed by this method, thus giving a "momentum preserving" strategy.

Considerable empirical evidence for two and three-dimensional problems indicates that this preconditioning strategy is effective, leading to convergence rates that are independent of mesh size and mildly dependent on Reynolds numbers for steady flow problems [9,12,17,29]. A proof that convergence rates are independent of the mesh is given in [19]. One drawback is the requirement that the matrix F_{p} be constructed. There might be situations where a developer of a solver does not have access to the code that would be needed to construct F_{p}. This issue is addressed in the following section.

3.2.2. The least squares commutator (LSC) preconditioner

The Least Squares Commutator method automatically generates an F_{p} matrix by solving the normal equations associated with a certain least squares problem derived from the commutator [10]. This approach leads to the following definition of F_{p}:

$$F_{\mathrm{p}} = Q_{\mathrm{p}}(\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}})^{-1}(\widehat{B}Q_{\mathrm{v}}^{-1}FQ_{\mathrm{v}}^{-1}B^{\mathrm{T}}). \tag{20}$$

Substitution of the operator into (19) generates an approximation to the Schur complement for div-stable finite element discretizations (i.e. $C = 0$):

$$\widehat{B}F^{-1}B^{\mathrm{T}} \approx (\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}})^{-1}(\widehat{B}Q_{\mathrm{v}}^{-1}FQ_{\mathrm{v}}^{-1}B^{\mathrm{T}})^{-1}(BQ_{\mathrm{v}}^{-1}B^{\mathrm{T}}). \tag{21}$$

For stabilized finite element discretizations, this can be modified to

$$C + \widehat{B}F^{-1}B^{\mathrm{T}} \approx (\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C)^{-1}(\widehat{B}Q_{\mathrm{v}}^{-1}FQ_{v}^{-1}B^{\mathrm{T}})(\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C)^{-1} + \alpha D^{-1} \tag{22}$$

where α and β are scaling factors, and D is the diagonal of $(\widehat{B}\mathrm{diag}(F)^{-1}B^{T} + C)$. For a further discussion of the merits of this method including heuristics for generating α and β, see [11].

In the taxonomy, the LSC operator is generated by grouping together the upper triangular and diagonal factors as in (12), choosing $H_2 = F^{-1}$ and $\hat{\mathrm{S}}$ as in (22). In matrix form this is

$$A_{\mathrm{LSC}} = \begin{bmatrix} F & FH_2B^{\mathrm{T}} \\ 0 & -\widehat{S} \end{bmatrix} = \begin{bmatrix} F & B^{\mathrm{T}} \\ 0 & (\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C)(\widehat{B}Q_{\mathrm{v}}^{-1}FQ_{v}^{-1}B^{\mathrm{T}})^{-1}(\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C) + \alpha D \end{bmatrix}.$$

The error matrix is

$$E_{\mathrm{LSC}} = A - \widetilde{A}_{\mathrm{LSC}} = \begin{bmatrix} 0 & 0 \\ \hat{B} & (\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C)(\widehat{B}Q_{\mathrm{v}}^{-1}FQ_{v}^{-1}B^{\mathrm{T}})^{-1}(\widehat{B}Q_{\mathrm{v}}^{-1}B^{\mathrm{T}} + \gamma C) + \alpha D - C \end{bmatrix},$$

so that the momentum equation is again unperturbed. Empirical evidence indicates that this strategy is effective, leading to convergence rates that are mildly dependent on Reynolds numbers for steady flow problems [12,29].

3.2.3. The approximate SIMPLE commutator preconditioner

In this section, we define an alternative strategy that uses the same factors as SIMPLE, together with the commutator used to derive the P-C–D and LSC factorizations. This results in a "mass preserving" strategy. In terms of the taxonomy, this method is generated by grouping together the lower triangular and diagonal factors, choosing $H_1 = F^{-1}$ and $\widehat{S} = C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1}$. Insertion of the choices into (12) leads to

$$A_{\mathrm{ASC}} = \begin{bmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{bmatrix} = \begin{bmatrix} F & 0 \\ \widehat{B}H_1F & -\widehat{S} \end{bmatrix}\begin{bmatrix} I & H_2B^{\mathrm{T}} \\ 0 & I \end{bmatrix} = \begin{bmatrix} F & 0 \\ \widehat{B} & -(C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1}) \end{bmatrix}\begin{bmatrix} I & H_2B^{\mathrm{T}} \\ 0 & I \end{bmatrix}.$$

We can approximate H_2B^{T} in the upper triangular factor by $\mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1}$. In matrix form this becomes

$$A_{\mathrm{ASC}} = \begin{bmatrix} F & B^{\mathrm{T}} \\ \widehat{B} & -C \end{bmatrix} = \begin{bmatrix} F & 0 \\ \widehat{B} & -(C + \widehat{B}\mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1}) \end{bmatrix}\begin{bmatrix} I & \mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1} \\ 0 & I \end{bmatrix}.$$

The error matrix is

$$E_{\mathrm{ASC}} = A - \widetilde{A}_{\mathrm{ASC}} = \begin{bmatrix} 0 & B^{\mathrm{T}} - F\mathrm{diag}(F)^{-1}B^{\mathrm{T}}F_{\mathrm{p}}^{-1} \\ 0 & 0 \end{bmatrix}.$$

Here, the continuity equation is unperturbed. This method performs well when the error in the (1, 2) block is small. More details on the method with a further discussion of how this method compares to SIMPLE can be found in [10].

4. Implementation and testing environment

We have tested the methods discussed above using MPSalsa [26], a code developed at Sandia National Laboratory, that models chemically reactive, incompressible fluids. The discretization of the Navier–Stokes equations provided by MPSalsa is a pressure stabilized, streamline upwinded Petrov Galerkin finite element scheme

[30] with Q_1–Q_1 elements. One advantage of equal order interpolants is that the velocity and pressure degrees of freedom are defined at the same grid points, so the same interpolants for both velocity and pressure are used.

4.1. Problem and preconditioner structure

The nonlinear system is solved by Newton's method where the structure of a two-dimensional steady version of F is a 2×2 block matrix consisting of a discrete version of the operator

$$\begin{pmatrix} -\nu\Delta + u^{(n-1)} \cdot \nabla + (u_1^{(n-1)})_x & (u_1^{(n-1)})_y \\ (u_2^{(n-1)})_x & -\nu\Delta + u^{(n-1)} \cdot \nabla + (u_2^{(n-1)})_y \end{pmatrix}. \tag{23}$$

For the pressure convection–diffusion preconditioning strategy, we need to specify the operators F_p, A_p, and Q_p. These operators are generated using the application code, MPSalsa. For the A_p operator required by this strategy, we choose it by taking $1/\nu$ times the symmetric part of F_p. This generates a Laplacian type operator suitable for the use in this preconditioning strategy. For Q_p, we use a lumped version of the pressure mass matrix. For problems with inflow boundary conditions, we specify Dirichlet boundary conditions on the inflow boundary for all of the preconditioning operators [8]. For singular operators found in problems with enclosed flow, the hydrostatic pressure makes B^T and the Jacobian system rank-deficient by one. Since we are given a Jacobian matrix from MPSalsa that is "pinned", i.e. a row and column that is causing the rank deficiency is removed, we pin all of the operators in the preconditioner (F_p, A_p,Q_p) as the Jacobian matrix is pinned. The other methods (i.e. SIMPLE, LSC) in this study were built as described in Section 3.

One aspect of the block preconditioners discussed here is that they require two subsidiary scalar computations, solutions for the Schur complement approximation and convection–diffusion-like subproblem. Both of these computations are amenable to multigrid methods. We employ smoothed aggregation algebraic multigrid (AMG) for these computations because AMG does not require mesh or geometric information, and thus is attractive for problems posed on complex domains or unstructured meshes. More details on AMG can be found in [31,33].

4.2. Software

Our implementation of the preconditioned Krylov subspace solution algorithm uses *Trilinos* [16], a software environment developed at Sandia National Laboratories for implementing parallel solution algorithms using a collection of object-oriented software packages for large-scale, parallel multiphysics simulations. One advantage of using Trilinos is its capability to seamlessly use component packages for core operations. We use the following components of Trilinos:

1. *Meros* – This package provides scalable block preconditioning for problems with coupled simultaneous solution variables. Both the pressure convection–diffusion and SIMPLE preconditioner studied here are implemented in this package. Meros uses the Epetra package for basic linear algebra functions.
2. *Epetra* – This package provides the fundamental routines and operations needed for serial and parallel linear algebra libraries. Epetra also facilitates matrix construction on parallel distributed machines. Each processor constructs the subset of matrix rows assigned to it via the static domain decomposition partitioning generated by a stand-alone library, CHACO [15], and a local matrix–vector product is defined. Epetra handles all the distributed parallel matrix details (e.g. local indices versus global indices, communication for matrix–vector products, etc.). Once the matrices F, B, $\widehat{B}$, and C are defined, a global matrix–vector product for (7) is defined using the matrix–vector products for the individual systems. Construction of the preconditioner follows in a similar fashion.
3. *AztecOO* – This package is a massively parallel iterative solver library for sparse linear systems. It supplies all of the Krylov methods used in solving (7), the F, and Schur complement approximation subsystems.
4. *ML* – This is a multi-level algebraic multigrid preconditioning package. We use this package with AztecOO to solve the F and Schur complement approximation subsystems.

5. *NOX* – This is a package for solving nonlinear systems of equations. We use NOX for the inexact nonlinear Newton solver.

4.3. Operations required

Once all of the matrices and matrix–vector products are defined, we can use Trilinos to solve the incompressible Navier–Stokes equations using our block preconditioner with specific choices of linear solvers for the Jacobian system and the convection–diffusion and Schur complement approximation subproblems.

For solving the system with coefficient matrix F we use the generalized minimal residual method (GMRES) preconditioned with four levels of algebraic multigrid, and for the pressure Poisson problem, we use the conjugate gradient method (CG) preconditioned with four levels of algebraic multigrid. For the convection–diffusion problem, a block Gauss Seidel (GS) smoother is used and for the pressure Poisson problem, a multi-level smoother polynomial is used for the smoothing operations [1]. The block GS smoother is a domain-based Gauss Seidel smoother where the diagonal blocks of the matrix (the velocity components) correspond to subdomains, and a traditional point GS sweep occurs in the smoothing step. The local Gauss–Seidel procedure includes a communication step (which updates ghost values around each subdomain's internal boundary) followed by a traditional Gauss–Seidel sweep within the subdomain. For the coarsest level in the multigrid scheme, a direct LU solve was employed. We used smoothed aggregation multigrid solvers available in Trilinos. To solve the linear problem associated with each Newton iteration, we use GMRESR, a variation on GMRES proposed by van der Vorst and Vuik [32] allowing the preconditioner to vary at each iteration. GMRESR is required because we use a preconditioned Krylov subspace method to generate approximate solutions in the subsidiary computations (pressure Poisson and convection–diffusion-like) of the preconditioner, so the preconditioner is not a fixed linear operator.

In our experiments, we compare methods from pressure correction (SIMPLEC) and approximate commutator (PCD) with a one-level Schwarz domain decomposition preconditioner [27]. This preconditioner does not vary from iteration to iteration (as the block preconditioners do), so GMRES can be used as the outer solver. Domain decomposition methods are based upon computing approximate solutions on subdomains. Robustness can be improved by increasing the coupling between processors, thus expanding the original subdomains to include unknowns outside of the processor's assigned nodes. Again, the original Jacobian system matrix is partitioned into subdomains using CHACO, whereas AztecOO is used to implement the one-level Schwarz method and automatically construct the overlapping submatrices. Instead of solving the submatrix systems exactly we use an incomplete factorization technique on each subdomain (processor). For our experiments, we used an ILU with a fill-in of 1.0 and a drop tolerance of 0.0. Therefore, the ILU factors have the same number of nonzeros as the original matrix with no entries dropped. A 2-level or 3-level Schwarz scheme might perform better. However, there are some issues with directly applying a coarsening scheme to the entire Jacobian-system due to the indefinite nature of the system [27].

5. Numerical results

5.1. Benchmark problems

For our computational study, we have focused our efforts on steady solutions of two benchmark problems, the lid driven cavity problem and flow over an obstruction, each posed in both two and three spatial dimensions

5.1.1. Driven cavity problem

For the two-dimensional driven cavity, we consider a square region with unit length sides. Velocities are zero on all edges except the top (the lid), which has a driving horizontal velocity of one. For the three-dimensional driven cavity, the domain is a cube with unit length sides. Velocities are zero on all faces of the cube, except the top (lid), which has a driving velocity of one. Each of these problems is then discretized on a uniform mesh of width h. In two dimensions, we have approximately $3/h^2$ unknowns, i.e. $1/h^2$ pressure and $2/h^2$ velocity unknowns. In three dimensions, we have approximately $4/h^3$ unknowns.

The lid driven cavity is a well-known benchmark for fluids problems because it contains many features of harder flows, such as recirculations. The lid driven cavity poses challenges to both linear and nonlinear solvers and exhibits unsteady solutions and multiple solutions at high Reynolds numbers. In two dimensions, unsteady solutions appear around Reynolds number 8000 [14]. In three dimensions, unsteady solutions appear around Reynolds number 100 [28]. Fig. 1 shows the velocity field and pressure field for an example solution to a two-dimensional lid driven cavity problem with $h = 1/128$.

5.1.2. Flow over an obstruction

For the two-dimensional flow over a diamond obstruction, we consider a rectangular region with width of unit length and a channel length of seven units, where the fluid flows in one side of a channel, then around the obstruction and out the other end of the channel. Velocities are zero along the top and bottom of the channel and along the obstruction. The flow is set with a parabolic inflow condition, i.e. $\mathbf{u}_x = 1 - y^2$, $\mathbf{u}_y = 0$ and a natural outflow condition, i.e. $\frac{\partial u_x}{\partial x} = p$ and $\frac{\partial u_x}{\partial x} = 0$.

For the three-dimensional flow over a cube, we consider a rectangular region with a width of one and a half units, a height of three units, and a channel length of five units. The fluid flows in one side of the channel, then around the cube, and out the other end of the channel. Velocities are zero along the top and bottom of the channel, and along the obstruction. The flow is set with a parabolic inflow condition similar to the two-dimensional case and with a natural outflow condition.

The flow over an obstruction also poses many difficulties for both linear and nonlinear solvers. This problem contains an unstructured mesh with inflow and outflow conditions which generates a more realistic, yet difficult problem than the driven cavity. In two dimensions, unsteady solutions appear around Reynolds number 50 [13]. Figs. 2 and 3 shows the velocity field and unstructured mesh for an example solution to a two-dimensional flow over a diamond obstruction for *Re* 25. Fig. 4 shows the velocity field for an example solution to a three-dimensional flow over a cube obstruction for *Re* 50.

5.2. Numerical results

We terminate the nonlinear iteration when the relative error in the residual is 10^{-4}, i.e.

$$\left\| \mathbf{f} - \begin{pmatrix} F(\mathbf{u})\mathbf{u} + B^{\mathrm{T}}p) \\ g - (\widehat{B}\mathbf{u} - Cp) \end{pmatrix} \right\| \leqslant 10^{-4} \left\| \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix} \right\|. \tag{24}$$

The tolerance η_k for (6), the solve with the Jacobian system, is fixed at 10^{-5} with zero initial guess. For all of the problems with the pressure convection–diffusion preconditioner, we employ inexact solves on the subsidiary pressure Poisson type and convection–diffusion subproblems. For solving the system with coefficient matrix A_{p}, we use six iterations of algebraic multigrid preconditioned CG and for the convection–diffusion-like subproblem, with coefficient matrix F, we fix a tolerance of 10^{-2}, i.e. this iteration is terminated when

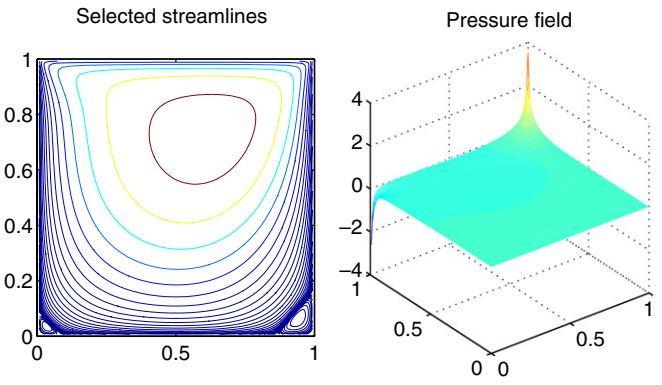

Fig. 1. Sample velocity field and pressure field from 2D lid driven cavity. $h = 1/128$, $Re = 100$.

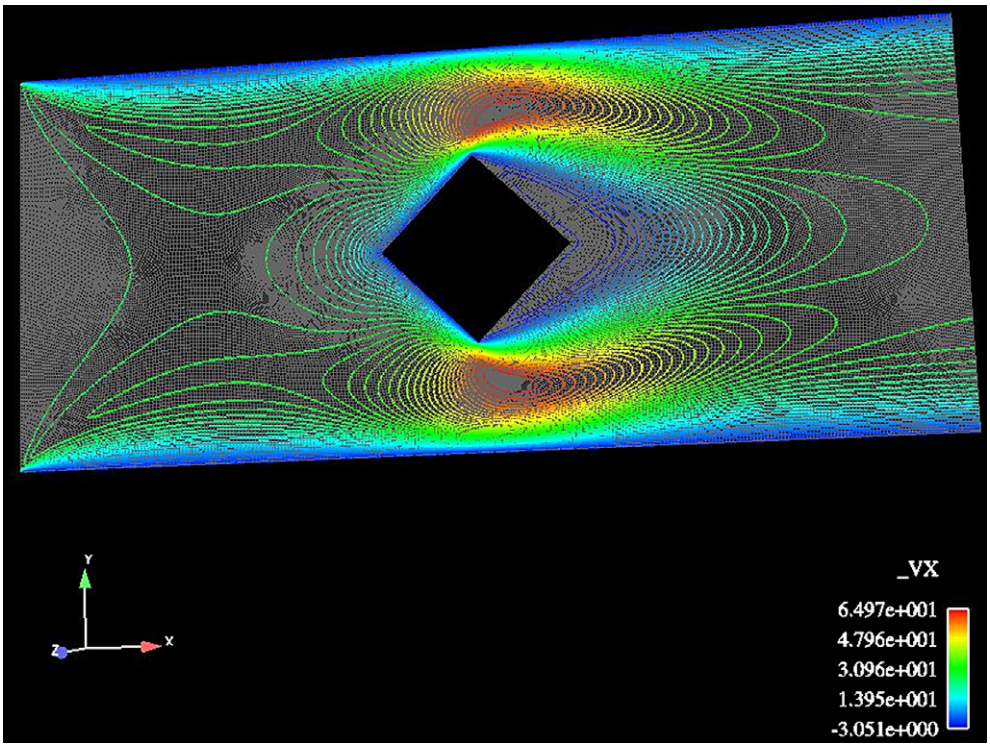

Fig. 2. Sample velocity field from 2D flow over a diamond obstruction. 62K unknowns, $Re = 25$.

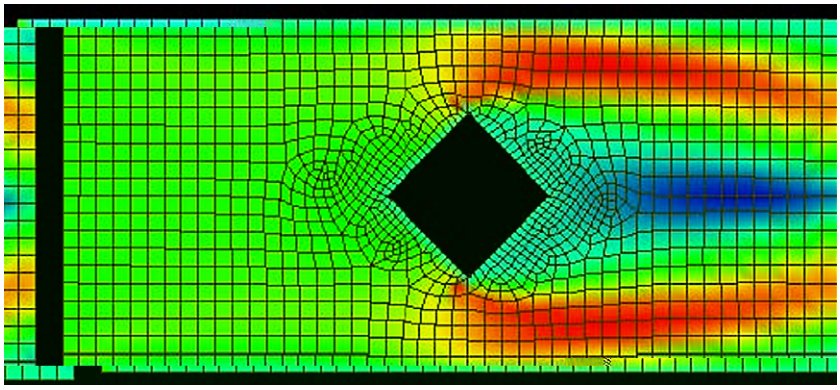

Fig. 3. Sample velocity field and unstructured mesh from 2D flow over a diamond obstruction.

$$\|(\mathbf{y} - F\mathbf{u})\| \leqslant 10^{-2}\|\mathbf{y}\|. \tag{25}$$

We compare this method to a one-level overlapping Schwarz domain decomposition preconditioner that uses GMRES to solve the Jacobian system at each step [27]. In order to minimize the CPU time and thus reduce the number of outer iterations, we have found that for the SIMPLEC preconditioner, we could not perform the Schur complement approximation solve and the solve with F as loosely as we did with the pressure convection–diffusion preconditioner. For SIMPLEC, we fix a tolerance of 10^{-5} for the solve with coefficient matrix F in (25) and the solve with the Schur complement approximation. For the pressure convection–diffusion and SIMPLEC preconditioners, we use a Krylov subspace size of 300 and a maximum number of iterations of 900. For the 2D domain decomposition preconditioner, we use a Krylov subspace of 600 and a maximum number of iterations of 1800. For the 3D domain decomposition preconditioner, we use a Krylov subspace of 400 and

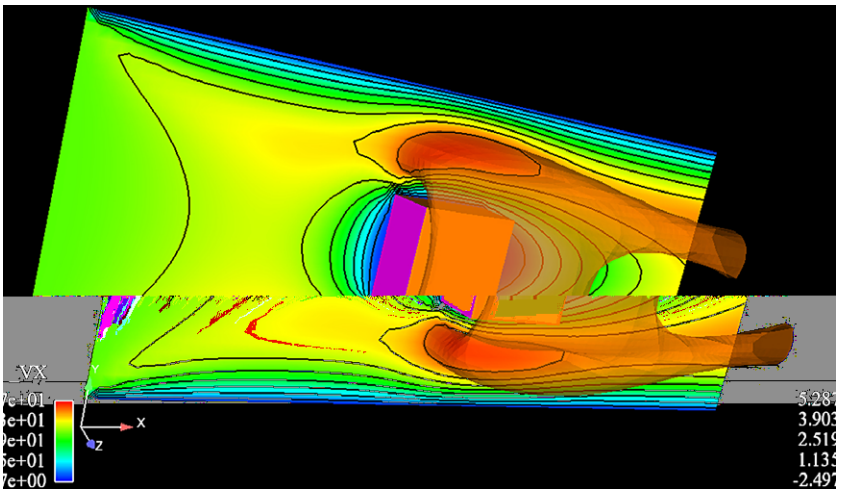

Fig. 4. Sample velocity field from 3D flow over a cube obstruction, $Re = 50$.

a maximum number of iterations of 1200. All of these values are chosen to limit the number of restarts needed for the solver, while balancing the memory on the compute node. The results were obtained in parallel on Sandia's Institutional Computing Cluster (ICC). Each of this cluster's compute nodes are dual Intel 3.6 GHz Xenon processors with 2 GB of RAM.

5.2.1. Lid driven cavity problem

We first compare the performance of the pressure convection–diffusion preconditioner to that of the domain decomposition preconditioner and SIMPLEC on the lid driven cavity problem generated by MPSalsa. In the first column of Table 1, we list the Reynolds number followed by four mesh sizes in column two. In columns three, four, and five, we list the total CPU time and the average number of outer linear iterations per Newton step for the pressure convection–diffusion, domain decomposition, and SIMPLEC preconditioners, respectively.

The trends are as follows. The pressure convection–diffusion method displays iteration counts that are largely independent of the mesh size. The domain decomposition preconditioner does not display mesh independent convergence behavior as the mesh is refined. However, there is much less computational effort involved in one iteration of preconditioning with domain decomposition than in one iteration of preconditioning with pressure convection–diffusion. For the fine meshes, the CPU time for the pressure convection–diffusion preconditioner is four times smaller than domain decomposition (when the latter method was convergent). The SIMPLEC method also does not display mesh independent convergence behavior, but it provides solutions in fewer iterations and in less CPU time for finer meshes than the domain decomposition preconditioner. For large Re, SIMPLEC is sensitive to the damping parameter on the pressure update. For the results shown, the damping factor was 0.01; for larger values of α the method stagnated. We found SIMPLE to be less effective than SIMPLEC and do not report results for SIMPLE.

For the 3D driven cavity problems (Table 2), we find that the pressure convection–diffusion method again is faster on larger meshes than the one-level domain decomposition method. The pressure convection–diffusion method again displays essentially mesh-independent iteration counts and a slight dependence on the Reynolds number. The SIMPLEC method produces iteration counts that are less dependent on the Reynolds number than domain decomposition, but it is competitive and in many cases faster than domain decomposition in terms of CPU time.

5.2.2. Flow over an obstruction

The pressure convection–diffusion preconditioner, SIMPLEC, and the domain decomposition preconditioner are compared for the two-dimensional diamond obstruction problem in Table 3. The trends are sim-

Table 1
Comparison of the iteration counts and CPU time for the pressure convection–diffusion, SIMPLEC, and domain decomposition preconditioners for the 2D lid driven cavity problem

Re number	Mesh size	Pressure C–D		SIMPLEC		DD one-level		Procs
		Iters	Time	Iters	Time	Iters	Time	
$Re = 10$	64×64	19.4	17.2	41.8	32.9	79.4	19.4	1
	128×128	21.2	28.4	66.0	78.9	220.6	79.8	4
	256×256	23.0	69.3	104.3	229.2	467.2	619.4	16
	512×512	23.2	257.2	164.0	619.4	1356.8	2901.9	64
$Re = 100$	64×64	35.0	28.7	52.0	50.8	86.5	26.4	1
	128×128	35.9	59.5	71.8	87.9	300.3	130.2	4
	256×256	41.3	102.1	109.8	410.5	528.8	593.1	16
	512×512	41.0	345.7	169.4	941.2	NC	NC	64
$Re = 500$	64×64	73.0	200.5	73.9	206.7	89.7	44.4	1
	128×128	79.1	385.6	107.5	401.2	334.9	215.9	4
	256×256	84.3	607.4	177.6	1600.6	896.1	1592.5	16
	512×512	90.2	1811.1	204.3	4109.2	NC	NC	64
$Re = 1000$	64×64	NC	NC	NC	NC	NC	NC	1
	128×128	126.4	570.9	142.0	1220.4	352.5	275.8	4
	256×256	126.6	1207.6	251.6	3494.2	839.5	2009.6	16
	512×512	143.2	2563.2	401.2	7598.2	NC	NC	64

NC stands for no covergence. For the 512×512 DD solver results, we could not converge to a solution for a Krylov subspace size of 900 and 4500 max iterations.

Table 2
Comparison of the iteration counts and CPU time for the pressure convection–diffusion, SIMPLEC, and domain decomposition preconditioners for the 3D lid driven cavity problem

Re number	Mesh size	Pressure C–D		SIMPLEC		One-level DD		Procs
		Iters	Time	Iters	Time	Iters	Time	
$Re = 10$	$32 \times 32 \times 32$	28.0	803.2	30.5	1205.6	67.0	634.6	1
	$64 \times 64 \times 64$	28.4	865.2	50.8	2034.1	159.8	1507.5	8
	$128 \times 128 \times 128$	31.1	1249.0	280.8	12490.5	356.2	4529.3	64
$Re = 50$	$32 \times 32 \times 32$	40.2	946.9	33.3	1302.6	62.2	615.5	1
	$64 \times 64 \times 64$	47.8	1061.6	52.5	2457.6	162.6	1533.2	8
	$128 \times 128 \times 128$	50.1	2101.2	291.2	14987.2	385.5	6460.9	64
$Re = 100$	$32 \times 32 \times 32$	56.0	1232.7	40.8	1884.4	61.7	730.7	1
	$64 \times 64 \times 64$	62.1	1697.8	61.6	3184.4	168.5	2131.6	8
	$128 \times 128 \times 128$	64.2	3019.2	299.1	17184.2	404.6	6953.9	64

ilar to the results from the driven cavity problem; in particular, iteration counts for PCD are largely independent of mesh size for a given Reynolds number. The domain decomposition preconditioner and SIMPLEC do not display mesh independent convergence behavior as the mesh is refined. For *Re* 10 and *Re* 25, the pressure convection–diffusion preconditioner was faster in all cases. For *Re* 40, it was faster for all meshes except for the small problems with 62,000 unknowns on one processor. Note that the GMRES solver preconditioned with domain decomposition stagnated before a solution was found for the problems with 4 million unknowns. The pressure convection–diffusion preconditioner converged without difficulty on this problem. On modest sized problems (those with more than 256 K unknowns) where both methods converged, the pressure convection–diffusion preconditioner ranged from 4 to 14 times faster than domain decomposition.

Results for the three-dimensional flow over a cube in Table 4. Once again the trends are similar; we omit a detailed discussion.

Table 3
Comparison of the iteration counts and CPU time for the pressure convection–diffusion, SIMPLEC and domain decomposition preconditioners for the 2D flow over a diamond obstruction

Re number	Unknowns	Pressure C–D		SIMPLEC		DD one-level		Procs
		Iters	Time	Iters	Time	Iters	Time	
$Re = 10$	62K	21.7	138.8	52.8	502.2	110.8	186.6	1
	256K	22.6	192.7	83.6	1203.9	282.6	1054.9	4
	1M	25.6	252.3	130.8	1845.3	890.2	6187.4	16
	4M	29.7	397.5	212.6	5834.6	NC	NC	64
$Re = 25$	62K	34.9	248.0	66.5	760.5	101.7	198.8	1
	256K	40.4	384.6	104.7	1920.3	273.8	1118.6	4
	1M	43.6	445.9	160.8	2985.2	864.5	6226.0	16
	4M	49.1	736.6	402.1	8241.3	NC	NC	64
$Re = 40$	62K	64.6	565.8	74.8	1278.7	70.4	267.2	1
	256K	68.9	975.2	113.6	2718.9	203.9	1269.3	4
	1M	72.7	1039.2	260.9	7535.0	770.0	6933.5	16
	4M	78.3	1528.6	410.1	11992.2	NC	NC	64

NC stands for no convergence.

Table 4
Comparison of the iteration counts and CPU time for the pressure convection–diffusion and domain decomposition preconditioners for the flow over a 3D cube

Re number	Unknowns	Pressure C–D		SIMPLEC		DD one-level		Procs
		Iters	Time	Iters	Time	Iters	Time	
$Re = 10$	270K	20.7	997.7	45.2	1897.1	67.2	859.8	1
	2.1M	21.7	1507.5	79.3	4593.2	151.2	2004.0	8
	16.8M	24.7	1997.7	118.7	19907.1	667.2	20908.0	64
$Re = 50$	270K	35.9	1209.7	49.2	2109.2	69.4	889.2	1
	2.1M	38.7	1797.7	84.9	3201.3	132.4	2676.1	8
	16.8M	44.7	2397.7	140.2	28156.1	637.2	18646.0	64

NC stands for no convergence.

Table 5
Comparison of the iteration counts and CPU time for the inexact pressure convection–diffusion, exact pressure convection–diffusion and domain decomposition preconditioners for the 2D flow over a diamond obstruction

Re number	Unknowns	Inexact Pressure C–D		Exact P-C–D		DD one-level		Procs
		Iters	Time	Iters	Time	Iters	Time	
$Re = 10$	62K	21.7	138.8	18.7	194.8	110.8	186.6	1
	256K	22.6	192.7	16.8	294.0	282.6	1054.9	4
	1M	25.6	252.3	16.1	406.4	890.2	6187.4	16
	4M	29.7	397.5	14.8	655.8	NC	NC	64
$Re = 25$	62K	34.9	248.0	32.8	695.2	101.7	198.8	1
	256K	40.4	384.6	31.6	621.4	273.8	1118.6	4
	1M	43.6	445.9	28.6	778.8	864.5	6226.0	16
	4M	49.1	736.6	25.3	1312.8	NC	NC	64
$Re = 40$	62K	64.6	565.8	44.4	781.3	70.4	267.2	1
	256K	68.9	975.2	39.2	1116.7	203.9	1269.3	4
	1M	72.7	1039.2	38.7	1352.7	770.0	6933.5	16
	4M	78.3	1528.6	35.2	2280.3	NC	NC	64

NC stands for no convergence.

Finally, in Table 5, we compare the impact of inexact solves of the subsidiary systems required for the pressure convection–diffusion preconditioner. In particular, we look at the "exact" pressure convection–diffusion preconditioner, where we solved the subsidiary systems to a tolerance of 10^{-5}. The exact PCD preconditioner shows iteration counts that are mesh independent and reduce as the mesh is refined, but with increasing CPU cost. However, the exact method is still considerably faster than domain decomposition for this problem. For a user of these methods, we recommend the inexact variant because the iteration counts are nearly independent and require less CPU time.

5.2.3. Additional discussion

We comment on some additional points concerning costs and scalability of the PCD preconditioner. In most of the examples with this preconditioner, as the mesh is refined we do notice an increase in the computational time for a given Reynolds number. A representative example is from Table 3, *Re* 25, where the CPU times are 445.9 and 736.6 in the cases of 1M and 4M unknowns, respectively. There are two causes for this. One is iteration counts: both the outer iterations needed to satisfy the stopping criterion (6) and the inner iterations needed for (25) in the approximate convection–diffusion solve show some increase as the mesh is refined.[2] The convection–diffusion solve is the dominant computation of the outer iteration, and this leads to an increase in CPU time even though the number of unknowns per processor is constant. In the example cited from Table 3, the average inner iteration counts increased from 10 (for 1M unknowns) to 13, and the average outer iteration counts increased from 43.6 to 49.1. If we use the factor $(\frac{43.6}{49.1})(\frac{10}{13}) = .68$ to adjust the CPU time in the case of 4M unknowns, we obtain an adjusted CPU time of 503.1, which is 13% higher than the time for 1M unknowns.

The second main cause of performance sensitivity to mesh size is the increasing cost of the coarsest level solve in the multi-level method, which in these tests was done with a serial sparse direct solver. One can control this cost by adding additional levels to the multi-level method or by using either a parallel solver or an iterative method for the coarse direct solve. However, we have found that this computation is not responsible for significant overhead (about 13% in the example cited above) and we have not explored this further.

6. Conclusions

We have described a taxonomy for preconditioning techniques for the incompressible Navier–Stokes equations. We have included traditional methods of pressure projection and pressure correction type along with newer approximate commutator methods derived from an approximation of the Schur complement. This taxonomy is based upon a block factorization of the Jacobian matrix in the Newton nonlinear iteration where methods are determined by making choices on the grouping of the block upper, lower, and diagonal factors along with approximations to the action of the inverse of certain operators and the Schur complement. All the methods require solutions of discrete scalar systems of convection diffusion and pressure Poisson-type that are significantly easier to solve than the entire coupled system.

In experiments with these methods using benchmark problems from MPSalsa we have demonstrated that the pressure convection–diffusion method gives superior iteration counts and CPU times for 2D and 3D problems with the one-level additive Schwarz domain decomposition method. For the approximate commutator methods we have demonstrated asymptotic convergence behavior that is essentially mesh independent in 2D and 3D for problems generated by an application code, MPSalsa, over a range of Reynolds numbers and problems discretized on structured and unstructured meshes with inflow and outflow conditions. For the steady-state problems explored, the iteration counts show only a slight degradation for increasing Reynolds number. In the future, we intend to further expand this technique to time dependent problems and problems posed on more complex domains.

[2] We expect the former count to tend to a constant as the mesh is refined, but smoothed aggregation multigrid can display some mild mesh dependence [31,33]. The convection–diffusion solve also has an impact on costs as the Reynolds number is increased. Solving nonsymmetric problems with algebraic multigrid is an active research topic; if a more effective scalable solver did exist for this subproblem, then the CPU would be considerably lower and more scalable [2].

References

[1] M. Adams, M. Brezina, J. Hu, R. Tuminaro, Parallel multigrid smoothing: polynomial versus Gauss-Seidel, Journal of Computational Physics 188 (2003) 593–610.
[2] M. Benzi, Preconditioning techniques for large linear systems: a survey, Journal of Computational Physics 182 (2002) 418–477.
[3] M. Benzi, G.H. Golub, A preconditioner for generalized saddle point problems, SIAM Journal on Matrix Analysis and its Applications 26 (2004) 20–41.
[4] M. Benzi, G.H. Golub, J. Liesen, Numerical solution of saddle point problems, Acta Numerica 14 (2005) 1–137.
[5] A.N. Brooks, T. Hughes, Streamline upwind/Petrov–Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations, Computational Methods in Applied Mechanics and Engineering 32 (1982) 199–259.
[6] E. Chow, Y. Saad, Approximate inverse techniques for block-partitioned matrices, SIAM Journal on Scientific Computing 18 (1997) 1657–1675.
[7] S.C. Eisenstat, H.F. Walker, Choosing the forcing terms in an inexact Newton method, SIAM Journal on Scientific Computing 17 (1996) 16–32.
[8] H. Elman, D. Silvester, A. Wathen, Finite Elements and Fast Iterative Solvers, Oxford University Press, Oxford, UK, 2005.
[9] H.C. Elman, Preconditioning for the steady-state Navier–Stokes equations with low viscosity, SIAM Journal on Scientific Computing 20 (1999) 1299–1316.
[10] H.C. Elman, V.E. Howle, J. Shadid, R. Shuttleworth, R. Tuminaro, Block preconditioners based on approximate commutators, SIAM Journal on Scientific Computing 27 (2006) 1651–1668.
[11] H.C. Elman, V.E. Howle, J. Shadid, D. Silvester, R. Tuminaro, Least squares preconditioners for stabilized discretizations of the Navier–Stokes equations, Technical Report, University of Maryland, College Park, SIAM Journal on Scientific Computing, in press.
[12] H.C. Elman, D.J. Silvester, A.J. Wathen, Performance and analysis of saddle point preconditioners for the discrete steady-state Navier–Stokes equations, Numerische Mathematik 90 (2002) 665–688.
[13] B. Fornberg, Computing incompressible flows past blunt bodies – a historical overview, Numerical Methods for Fluid Dynamics IV (1993).
[14] A. Fortin, M. Jardark, J. Gervais, R. Pierre, Localization of Hopf bifurcations in fluid flow problems, International Journal for Numerical Methods in Fluids 24 (1997) 1185–1210.
[15] B. Hendrickson, R. Leland, A Users Guide to Chaco, Version 1.0., Technical Report SAND93-2339, Sandia National Laboratories, 1993.
[16] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, K.S. Stanley, An overview of the Trilinos project, ACM Transactions on Mathematical Software 31 (2005) 397–423.
[17] D. Kay, D. Loghin, A.J. Wathen, A preconditioner for the steady-state Navier–Stokes equations, SIAM Journal on Scientific Computing 24 (2002) 237–256.
[18] D.A. Knoll, D.E. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, Journal of Computational Physics 193 (2004) 357–397.
[19] D. Loghin, A. Wathen, H. Elman, Preconditioning techniques for Newton's method for the incompressible Navier–Stokes equations, BIT 43 (2003) 961–974.
[20] M.F. Murphy, G.H. Golub, A.J. Wathen, A note on preconditioning for indefinite linear systems, SIAM Journal on Scientific Computing 21 (2000) 1969–1972.
[21] S.V. Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, New York, 1980.
[22] S.V. Patankar, D.A. Spalding, A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows, International Journal on Heat and Mass Transfer 15 (1972) 1787–1806.
[23] M. Pernice, M.D. Tocci, A multigrid-preconditioned Newton–Krylov method for the incompressible Navier–Stokes equations, SIAM Journal on Scientific Computing 123 (2001) 398–418.
[24] J.B. Perot, An analysis of the fractional step method, Journal of Computational Physics 108 (1993) 51–58.
[25] A. Quarteroni, F. Saleri, A. Veneziani, Factorization methods for the numerical approximation of Navier–Stokes equations, Computational Methods in Applied Mechanical Engineering 188 (2000) 505–526.
[26] J. Shadid, A. Salinger, R. Schmidt, T. Smith, S. Hutchinson, G. Hennigan, K. Devine, H. Moffat, MP- Salsa Version 1.5: A Finite Element Computer Program for Reacting Flow Problems, Technical Report, Sandia National Laboratories, 1998.
[27] J. Shadid, R. Tuminaro, K. Devine, G. Hennigan, P. Lin, Performance of fully-coupled domain decomposition preconditioners for finite element transport/reaction simulations, Journal of Computational Physics 205 (2005) 24–47.
[28] P.N. Shankar, M.D. Deshpande, Fluid mechanics in the driven cavity, Annual Review of Fluid Mechanics 32 (2000) 93–136.
[29] D. Silvester, H. Elman, D. Kay, A. Wathen, Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow, Journal on Computational and Applied Mathematics 128 (2001) 261–279.
[30] T.E. Tezduyar, Stabilized finite element formulations for incompressible flow computations, Advances in Applied Mechanics 28 (1991) 1–44.
[31] R. Tuminaro, C. Tong, Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines, in: J. Donnelley (Ed.), SuperComputing 2000 Proceedings, 2000.

[32] H.A. van der Vorst, C. Vuik, GMRESR: a family of nested GMRES methods, Numerical Linear Algebra with Applications 1 (1994) 369–386.
[33] P. Vanek, M. Brezina, J. Mandel, Convergence of algebraic multigrid based on smoothed aggregation, Numerische Mathematik 88 (2001) 559–579.