Technical Report

TR-2012-007

A Parallel Implementation of the Modified Augmented Lagrangian Preconditioner for the Incompressible Navier-Stokes Equations

by

Michele Benzi, Zhen Wang

Mathematics and Computer Science

EMORY UNIVERSITY

A parallel implementation of the modified augmented Lagrangian preconditioner for the incompressible Navier–Stokes equations

Michele Benzi* Zhen Wang†

April 9, 2012

Abstract

We describe a parallel implementation of a block triangular preconditioner based on the modified augmented Lagrangian approach to the steady incompressible Navier–Stokes equations. The equations are linearized by Picard iteration and discretized with finite elements or finite differences on two- and three-dimensional domains. Results show good scalability of the parallel solver for up to 64 cores.

Keywords: preconditioning, saddle point problems, Oseen problem, Krylov subspace methods, multicores

1 Introduction

In this paper we consider the parallel solution of the incompressible steady-state Navier–Stokes equations describing viscous Newtonian fluids. Given an open bounded domain $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ with boundary $\partial\Omega$ and a given external force field $\mathbf{f}$ and (for example) Dirichlet boundary data $\mathbf{g}$, the goal is to find the velocity vector field $\mathbf{u} = \mathbf{u}(\mathbf{x})$ and pressure scalar field $p = p(\mathbf{x})$

*Department of Mathematics and Computer Science, Emory University, Atlanta, GA 30322, USA (benzi@mathcs.emory.edu). Work supported in part by a grant of the University Research Committee of Emory University.

†Scientific Computing Group, National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA (wangz@ornl.gov). Work supported in part by the Laney Graduate School of Arts and Science at Emory University.

satisfying the following system of partial differential equations:

$$-\nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in} \quad \Omega, \tag{1}$$

$$\operatorname{div}\mathbf{u} = 0 \quad \text{in} \quad \Omega, \tag{2}$$

$$\mathbf{u} = \mathbf{g} \quad \text{on} \quad \partial\Omega, \tag{3}$$

where $\nu > 0$ is the kinematic viscosity (inversely proportional to the Reynolds number), Δ is the vector Laplacian in $\mathbb{R}^d$, ∇ denotes the gradient and div is the divergence. Because of the convective term $(\mathbf{u}\cdot\nabla)\mathbf{u}$, the Navier–Stokes system is nonlinear. A widely used linearization method is Picard's fixed-point iteration. At each Picard iteration, a system of PDEs of the form

$$-\nu\Delta\mathbf{u} + (\mathbf{w}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in} \quad \Omega, \tag{4}$$

$$\operatorname{div}\mathbf{u} = 0 \quad \text{in} \quad \Omega, \tag{5}$$

$$\mathbf{u} = \mathbf{g} \quad \text{on} \quad \partial\Omega, \tag{6}$$

known as *Oseen problem*, is solved to obtain approximate solutions $(\mathbf{u}, p)$; see, e.g., [10, section 7.2.2]. In (4), the "wind" $\mathbf{w}$ is the velocity field obtained from the previous Picard step. The global convergence properties of Picard's iteration for the solution of the steady Navier–Stokes equations are discussed in detail in, e.g., [10] and [20]. In practice, convergence is often found to be quite rapid, except for very small values of the viscosity ν.

Discretization of the Oseen problem by finite elements, finite volumes, or finite differences leads to a generalized saddle point system [3], i.e., a large linear system of the form

$$\begin{pmatrix} A & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{7}$$

Here u and p represent the discrete velocity and pressure, respectively, $A \in \mathbb{R}^{n\times n}$ is the discretization of the diffusion and convection terms, $B^T \in \mathbb{R}^{n\times m}$ is the discrete gradient, B the (negative) discrete divergence, $C \in \mathbb{R}^{m\times m}$ is a stabilization matrix, and f and g contain forcing and boundary terms. If the discretization satisfies the Ladyzhenskaya-Babuška-Brezzi (LBB, or 'inf-sup') stability condition [10], no pressure stabilization is required and we can take $C = 0$. If the LBB condition is not satisfied, the stabilization matrix $C \neq 0$ is usually symmetric and positive semi-definite and the actual choice of C depends on the particular finite element pair being used; see, e.g., [10, Section 5.3.2]. In this paper we only present the results with stable discretization; results with stabilized discretization are quite similar.

A promising technique for preconditioning the linear system (7) is the modified augmented Lagrangian (AL)-based preconditioner introduced in [6] and analyzed in [5, 7]; see also [8, 14] for closely related work. The rate of convergence of preconditioned GMRES [18] with this preconditioner is independent of grid size and mildly dependent on viscosity for linear systems obtained by Picard and Newton linearization [7].

In this paper we evaluate the potential of the preconditioner for solving large problems in a parallel environment. Our parallel implementation is largely based on public domain software and utilities, mostly from the Trilinos suite [16]; details are given in section 3 below.

The remainder of the paper is organized as follows. In section 2, we briefly review the modified AL preconditioner. The details of the parallel implementation are given in section 3. Numerical results are presented in section 4, and final conclusions are drawn in section 5.

2 Review of the modified AL preconditioner

In this section, we recall the modified AL preconditioner for stable finite element pairs [6].

2.1 Problem formulation

Using an LBB-stable finite elements discretization method [10], the saddle point system is:

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \quad \text{or} \quad \mathcal{A}\,x = b. \tag{8}$$

The equivalent AL formulation [11] is

$$\begin{pmatrix} A_\gamma & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f_\gamma \\ g \end{pmatrix}, \quad \text{or} \quad \widehat{\mathcal{A}}\,x = \hat{b}, \tag{9}$$

where $A_\gamma := A + \gamma B^T W^{-1} B$, and $f_\gamma := f + \gamma B^T W^{-1} g$ with W symmetric positive definite. As shown in [4], a good choice for W is given by the pressure mass matrix M_p. In order to preserve sparsity one usually replaces M_p with its main diagonal $\widehat{M_p}$. The parameter $\gamma > 0$ can be optimized using Fourier analysis, at least for 2D problems; see [7].

2.2 An ideal AL-based preconditioner

The "ideal" AL-based preconditioner [4] for problem (9) is as follows:

$$\mathcal{P} = \begin{pmatrix} A_\gamma & B^T \\ 0 & \widehat{S} \end{pmatrix}, \tag{10}$$

where $\widehat{S}^{-1} := -\nu \widehat{M}_p^{-1} - \gamma W^{-1}$. We refer readers to [4, 5] for a detailed analysis of this preconditioner, for discussions on how to solve the linear system associated with A_γ and $\widehat{S}$, and for the choice of W and γ. Here the word "ideal" means that the whole block A_γ is used in the preconditioner. Solving auxiliary linear systems with this block can be challenging, especially on unstructured grids. This is the motivation for the modified AL preconditioner described below.

2.3 The modified AL-based preconditioner

In order to circumvent the difficulties associated with the solution of linear systems with coefficient matrix A_γ on unstructured grids and complicated geometries, the *modified* AL preconditioner was introduced in [6]. This is a simplified AL-based preconditioner which can be implemented using standard algebraic multilevel solvers for scalar elliptic PDEs. The modified AL preconditioner can be described as follows. Recall that in 2D we have $A = \mathrm{diag}(A_1, A_2)$, with each block A_i $(i = 1, 2)$ square and of order $n/2$, and $B = (B_1, B_2)$. Thus

$$\begin{aligned} A_\gamma &= A + \gamma B^T W^{-1} B \\ &= \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} + \gamma \begin{pmatrix} B_1^T \\ B_2^T \end{pmatrix} W^{-1} \begin{pmatrix} B_1 & B_2 \end{pmatrix} \\ &= \begin{pmatrix} A_1 + \gamma B_1^T W^{-1} B_1 & \gamma B_1^T W^{-1} B_2 \\ \gamma B_2^T W^{-1} B_1 & A_2 + \gamma B_2^T W^{-1} B_2 \end{pmatrix} \\ &=: \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}. \end{aligned}$$

Approximating A_γ with its block upper triangular part

$$\widetilde{A}_\gamma = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix},$$

we define the modified AL preconditioner to be the block triangular matrix

$$\widetilde{\mathcal{P}} = \begin{pmatrix} \widetilde{A}_\gamma & B^T \\ 0 & \widehat{S} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & B_1^T \\ 0 & A_{22} & B_2^T \\ 0 & 0 & \widehat{S} \end{pmatrix}. \tag{11}$$

We note in passing that although the original construction of the modified AL preconditioner relied on the block diagonal structure of A, this approximation can be generalized to cases where A does not have block diagonal structure, as in the case of Newton linearization — indeed, it is even more natural for such problems.

For 3D problems, $A = \mathrm{diag}(A_1, A_2, A_3)$ and $B = (B_1, B_2, B_3)$. Therefore, the coefficient matrix of the equivalent augmented Lagrangian formulation is

$$\begin{aligned} A_\gamma &= A + \gamma B^T W^{-1} B \\ &= \begin{pmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ 0 & 0 & A_3 \end{pmatrix} + \gamma \begin{pmatrix} B_1^T \\ B_2^T \\ B_3^T \end{pmatrix} W^{-1} \begin{pmatrix} B_1 & B_2 & B_3 \end{pmatrix} \\ &= \begin{pmatrix} A_1 + \gamma B_1^T W^{-1} B_1 & \gamma B_1^T W^{-1} B_2 & \gamma B_1^T W^{-1} B_3 \\ \gamma B_2^T W^{-1} B_1 & A_2 + \gamma B_2^T W^{-1} B_2 & \gamma B_2^T W^{-1} B_3 \\ \gamma B_3^T W^{-1} B_1 & \gamma B_3^T W^{-1} B_2 & A_3 + \gamma B_3^T W^{-1} B_3 \end{pmatrix} \\ &=: \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}. \end{aligned}$$

The ideal AL preconditioner is again given by (10). In the modified variant we replace A_γ with the block triangular approximation

$$\begin{aligned} \tilde{A}_\gamma &= \begin{pmatrix} A_1 + \gamma B_1^T W^{-1} B_1 & \gamma B_1^T W^{-1} B_2 & \gamma B_1^T W^{-1} B_3 \\ 0 & A_2 + \gamma B_2^T W^{-1} B_2 & \gamma B_2^T W^{-1} B_3 \\ 0 & 0 & A_3 + \gamma B_3^T W^{-1} B_3 \end{pmatrix} \\ &= \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix}. \end{aligned}$$

Note that in the 3D case three blocks are being dropped: the (2,1), (3,1) and (3,2) blocks of A_γ. Nevertheless, the performance is usually just as good as in the 2D case. Again, each diagonal block A_{ii} represents a discrete scalar convection-diffusion operator. Eigenvalue analysis and how to choose the parameter γ have been discussed in [6, 7].

Next, we consider the parallel implementation of this preconditioner.

3 Parallel implementation

The parallel implementation is built upon the Trilinos framework [16] and is written in C++. We illustrate the strategy used to parallelize the solver

in the 3D case; 2D problems are handled similarly.

Here we briefly describe the Trilinos packages and classes we use in our code.

- `Epetra`. Fundamental linear algebra package. It contains definitions of matrices, vectors, etc.
- `EpetraExt`. An extension to `Epetra`. In our implementation, it provides functions to read matrices and vectors in Matrix Market [1] format, and perform matrix-matrix multiplication and addition.
- `AztecOO`. Linear solvers, including GMRES and other Krylov subspace methods.
- `ML`. Multilevel preconditioner. This is the preconditioner used when computing approximate solutions to auxiliary linear systems associated with the diagonal blocks A_{ii}.
- `Teko`. Block preconditioners. In our implementation it is only used to extract blocks from a big matrix.
- `Epetra_Operator`. A base class for defining operations on `Epetra_MultiVector`.
- `Epetra_CrsMatrix`. A class for constructing and using sparse matrices in compressed row storage format. This is derived from `Epetra_Operator`.
- `Epetra_MultiVector`. A class enabling the construction and use of dense vectors and multi-vectors (defined as collections of vectors having the same length and distribution).
- `Epetra_Import`. A class that transfers data between different processes.

The most important parts, which also require most effort to use efficiently, are defining the matrix vector multiplication and preconditioner application in the modified AL setting. Consider first the (non-augmented) linear system written in the following block form:

$$\begin{pmatrix} A_1 & 0 & 0 & B_1^T \\ 0 & A_2 & 0 & B_2^T \\ 0 & 0 & A_3 & B_3^T \\ B_1 & B_2 & B_3 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}.$$

The first key issue in the implementation is the extraction of the submatrices A_1, A_2, A_3, B_1, B_2 and B_3 from A and B, and of the subvectors x_1, x_2, x_3, and x_4 from the vector x (and corresponding components from b). How this is done will depend on the code used to generate the test matrices. In this paper, two types of problems are solved: one is generated by MATLAB, and the other from the C++ finite elements library LifeV [2]. For the former, it is natural and convenient to partition A and B and to save the blocks in separate disk files using Matrix Market format in MATLAB, and then to read these files using the package `EpetraExt` into the `Epetra_CrsMatrix` format. As a result, the blocks A_1, A_2, A_3, B_1, B_2, and B_3 are distributed among all processes. For the latter, using the `Teko` package, we are able to partition the `Epetra_CrsMatrix` matrices A and B into smaller `Epetra_CrsMatrix` matrices A_1, A_2, A_3, B_1, B_2, and B_3, which are then distributed among all processes in the same fashion. When dealing with the global vectors x and b, no matter how the problem is generated, we partition the `Epetra_MultiVector` x in our code using functionalities in `Epetra`. Specifically, once a global vector has been generated (or read from a disk file, as is the case when MATLAB is used), `Epetra_Import`'s are created and used for transferring the data corresponding to the part of x comprising x_i to form the blocks x_i, as well as for forming the whole vector x from the blocks x_i. Note that these `Epetra_Import`'s are only created once and can be used hereafter. After this step, all matrices and vectors are partitioned as required, and one only operate on the blocks from now on.

With all the blocks available, we create two new `Epetra_Operator`'s, one implementing the matrix vector multiplication $\widehat{\mathcal{A}}v$ for a given vector v, and the other applying the modified AL preconditioner $\widetilde{\mathcal{P}}$ to a vector v, i.e., computing the action of $\widetilde{\mathcal{P}}^{-1}$ on v . The first `Epetra_Operator` is straightforward, since all the necessary blocks are already available. All we do is partition the vector v using the `Epetra_Import`'s already created, perform matrix vector products with the blocks, and form the new vector. In the `Epetra_Operator` for the preconditioner, explicit construction of the matrices $A_{ii} = A_i + \gamma B_i^T W^{-1} B_i$ is required, since the preconditioners for these blocks are given by the algebraic multilevel preconditioner `ML`, which requires explicit definition of them. Using the functionalities provided by `EpetraExt`, the necessary matrix multiplications and additions are performed. In contrast, the off-diagonal blocks A_{12}, A_{13} and A_{23} of the preconditioner need not be explicitky formed, since they are only used in matrix-vector products and these can be expressed in terms of A_1, A_2, A_3, B_1, B_2, B_3, and the diagonal matrix W^{-1}. Note that the preconditioner is block triangular and is applied in a block backward substitution fashion, as follows. First,

Table 1: GMRES(50) iterations and timings with modified AL preconditioner (cavity, Q2-Q1, uniform 256×256 grid, $\nu = 0.01$)

No. of cores	2	4	8	16	32	64
Iterations	16	15	15	14	15	15
Setup time	5.72	2.91	1.57	0.90	0.77	0.44
Iter time	6.07	2.95	1.55	0.78	0.64	0.31
Total time	11.79	5.86	3.12	1.68	1.41	0.75

a diagonal scaling with the approximate Schur complement $\widehat{S}$ is performed. The solution is used to construct the right-hand side for a system with coefficient matrix A_{33}. Next, this system is solved, and its solution is used in the construction of the right-hand side for a system with coefficient matrix A_{22}. This system is then solved, and its solution is used in the construction of the right-hand side of a linear system with coefficient matrix A_{11}, which is then solved. All solves involving the diagonal blocks A_{ii} are performed inexactly (with rather low accuracy needed, see next section). The entire procedure is repeated at each application of the block triangular preconditioner $\widetilde{\mathcal{P}}$, except for the construction of the preconditioners for the diagonal blocks, which can be reused within each Picard step. Note that all processors are involved in each of the solution steps described above, so that there are no idle processes in the course of a preconditioner application.

4 Numerical results

In this section we show the results of a few numerical experiments on a computer cluster. The cluster consists of 32 nodes and 128 processor cores. Each node has 2 dual-core AMD 2.2 GHz Opteron CPUs and 4 GB RAM. The program is compiled and run with Open MPI.

The first test problem is a strong scalability study for the steady 2D Oseen equations for the lid driven cavity problem generated by IFISS [9, 19]. The value of the viscosity is $\nu = 0.01$, and γ is chosen by Fourier analysis (see [7]). Using Q2-Q1 finite elements on the uniform 256×256 grid results in a $148,739 \times 148,739$ saddle point matrix with $16,836,552$ nonzero entries. In the modified AL preconditioner, the linear systems associated with the diagonal blocks A_{ii} are solved by one iteration of `ML`. The parameters for `ML` are based on the default parameters for non-symmetric smoothed aggregation [12] with some modifications. The linear solver is GMRES implemented

Table 2: GMRES(50) iterations and timings with modified AL preconditioner (3D Oseen, MAC, $\nu = 0.01$)

No. of cores	2	4	8	16	32	64
$64 \times 64 \times 64$	19	19	19	19	19	27
Setup time	8.68	5.03	3.82	1.94	1.66	1.60
Iter time	22.91	12.74	7.19	4.14	2.22	1.66
Total time	31.59	17.77	11.01	6.08	3.88	3.26
$128 \times 128 \times 128$	19	19	19	19	19	21
Setup time	78.70	44.96	23.65	14.30	9.23	8.25
Iter time	217.11	141.78	64.73	36.73	18.90	12.03
Total time	295.81	186.74	88.38	51.03	28.13	20.28

as in the Trilinos package `AztecOO` [15]. GMRES iteration counts and timings using $2, 4, 8, 16, 32$ and 64 cores are shown in Table 1. The 'Setup time' includes matrix multiplication and addition for constructing A_{11} and A_{22} as well as `ML` setup time for them. 'Iter time' is the iterative phase of preconditioned GMRES. The 'Total time' is the sum of the previous two. The iteration counts are practically constant as the number of cores grows, and the timings show fairly good scalability for up to 64 cores.

Next, we show the parallel results for a 3D Oseen problem discretized by the stable Marker-and-Cell [13] finite difference method with viscosity $\nu = 0.01$. GMRES iteration counts and timings are shown in Table 2 when the problem is discretized on the $64 \times 64 \times 64$ and $128 \times 128 \times 128$ grids. The augmentation parameter $\gamma = 0.06$ is determined experimentally to minimize the GMRES iteration counts.

On the $64 \times 64 \times 64$ grid, the coefficient matrix is $1,036,288 \times 1,036,288$ with $8,442,624$ nonzero elements. One can observe that the iteration counts do not typically increase; the case of the $64 \times 64 \times 64$ grid problem when using 64 cores is somewhat anomalous. The timings indicate good scalability. For the larger $128 \times 128 \times 128$ grid, the coefficient matrix is now $8,339,456 \times 8,339,456$ with $99,290,112$ nonzero entries. Very good scalability is still achieved up to 32 cores. When using 64 cores, total timings are still being reduced but some degradation of parallel performance is evident.

Next, in Table 3 we present GMRES iteration counts and timings for a linearized steady 3D lid driven cavity problem discretized by P2-P1 finite elements in the finite element library LifeV [2]. The viscosity ν is 0.05. Here we retain the full tensor $(\nabla \mathbf{u} + \nabla \mathbf{u}^T)/2$ in the Navier–Stokes equations. This leads to an additional so-called grad-div stabilization term [17], which

Table 3: GMRES(50) iterations and timings with modified AL preconditioner (3D cavity, P2-P1, $\nu = 0.05$)

No. of cores	2	4	8	16	32	64
$16 \times 16 \times 16$	25	25	24	25	23	23
Setup time	1.85	1.57	1.68	1.16	0.89	0.72
Iter time	5.35	3.68	3.06	1.94	0.95	0.67
Total time	7.20	5.25	4.74	3.10	1.84	1.39
$24 \times 24 \times 24$	24	23	23	23	23	22
Setup time	7.53	5.98	6.34	3.82	2.83	1.82
Iter time	18.80	11.61	10.32	6.14	4.11	2.04
Total time	25.33	17.59	16.66	9.96	6.94	3.86
$32 \times 32 \times 32$	22	21	20	20	20	19
Setup time	37.71	16.53	18.09	9.38	6.27	4.00
Iter time	67.88	29.24	25.41	12.88	8.11	4.33
Total time	105.59	45.77	43.50	22.26	14.38	8.33

is very similar to the algebraic augmentation $\gamma B^T W^{-1} B$; therefore, the modified AL preconditioner (with $\gamma = 1$) is a natural choice here. Moreover, the explicit construction of A_{ii} $(i = 1, 2, 3)$ can be circumvented since this is done during the discretization phase, saving a considerable amount of time; see also [8, 14]. Firstly, we observe that the iteration counts remain essentially unchanged as the number of cores increase or the size of the mesh becomes larger, demonstrating the effectiveness of the modified AL preconditioner and the `ML` preconditioner for subproblems. In fact, larger grids provide better approximations to the continuous problem, and the number of iterations actually decreases slightly. Secondly, good scalability is achieved for the $32 \times 32 \times 32$ grid, but not for smaller grids when using less than 8 cores. Finally, we observe the nearly perfect scaling with respect to problem size.

5 Conclusions

In this paper we evaluated the parallel performance of GMRES with a modified AL preconditioner for the solution of linear systems arising from various discretizations of the steady Oseen equations in 2D and 3D. Parallelization of the code was effected by means of Trilinos utilities and Open MPI. The results show good scalability of the preconditioned Krylov iteration approach

with respect to the number of cores and problem size. Ongoing work includes the implementation of the modified AL preconditioner in the framework of `Teko` within the Trilinos code suite.

References

[1] Matrix Market. `http://math.nist.gov/MatrixMarket/`.

[2] The LifeV project. `http://www.lifev.org`.

[3] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 2005.

[4] M. Benzi and M. A. Olshanskii. An augmented Lagrangian-based approach to the Oseen problem. *SIAM J. Sci. Comput.*, 28(6):2095–2113, 2006.

[5] M. Benzi and M. A. Olshanskii. Field-of-values convergence analysis of augmented Lagrangian preconditioners for the linearized Navier–Stokes problem. *SIAM J. Numer. Anal.*, 49:770–788, 2011.

[6] M. Benzi, M. A. Olshanskii, and Z. Wang. Modified augmented Lagrangian preconditioners for the incompressible Navier–Stokes equations. *Internat. J. Numer. Methods Fluids*, 66(4):486–508, 2011.

[7] M. Benzi and Z. Wang. Analysis of augmented Lagrangian-based preconditioners for the steady incompressible Navier–Stokes equations. *SIAM J. Sci. Comput.*, 33:2761–2784, 2011.

[8] S. Börm and S. Le Borne. $\mathcal{H}$-LU factorization in preconditioners for augmented Lagrangian and grad-div stabilized saddle point systems, Internat. J. Numer. Methods Fluids, 68:83–98, 2012.

[9] H. C. Elman, A. Ramage, and D. J. Silvester. Algorithm 886: IFISS, a Matlab toolbox for modelling incompressible flow. *ACM Trans. Math. Software*, 33(2):2–14, 2007.

[10] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2005.

[11] M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary Value Problems*, volume 15 of *Studies in Mathematics and its Applications*. North-Holland Publishing Co., Amsterdam, 1983.

[12] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. Sala. ML 5.0 Smoothed Aggregation User's Guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[13] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8(12):2182–2189, 1965.

[14] T. Heister and G. Rapin. Efficient augmented Lagrangian-type preconditioning for the Oseen problem using Grad-Div stabilization. Internat. J. Numer. Methods Fluids, DOI: 10.1002/fld.3654, 2012.

[15] M. A. Heroux. AztecOO user's guide. Technical Report SAND2004-3796, Sandia National Laboratories, 2004.

[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, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Software*, 31(3):397–423, 2005.

[17] M. A. Olshanskii and A. Reusken. Grad-div stabilization for Stokes equations. *Math. Comp.*, 73(248):1699–1718 (electronic), 2004.

[18] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[19] D. J. Silvester, H. C. Elman, and A. Ramage. IFISS: Incompressible Flow Iterative Solution Software, January 2011. `http://www.manchester.ac.uk/ifiss/`.

[20] R. Temam. *Navier–Stokes Equations. Fourth Edition*. AMS Chelsea Publishing, Providence, RI, 2001.