Parallel Iterative Methods for Navier-Stokes Equations and Application to Stability Assessment *

26th October 2002

Ivan G. Graham, Alastair Spence Eero Vainikko

Abstract

We describe the construction of parallel iterative solvers for finite element approximations of the Navier-Stokes equations on unstructured grids using domain decomposition methods. The iterative method used is FGMRES, preconditioned by a parallel adaptation of a recent block preconditioner proposed by Kay, Loghin and Wathen. The parallelisation is achieved by adapting the technology of our domain decomposition solver `DOUG` (previously used for scalar problems) to block-systems. An application of the resultant linear solver to the stability assessment of flows is briefly indicated.

1 Introduction

When steady solutions of complex physical problems are computed numerically it is often crucial to compute their *stability* in order to, for example, check that the computed solution is "physical", or carry out a sensitivity analysis, or help understand complex nonlinear phenomena near a bifurcation point. An important example is in the computation of fluid flows governed by the steady-state Navier-Stokes equations. Suppose that a velocity field $\mathbf{w}$ has been computed for some particular parameter values. To assess its stability it is necessary to solve the PDE eigenvalue problem:

$$\left.\begin{array}{rcl} -\epsilon\Delta\mathbf{u} + \mathbf{w}\cdot\nabla\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{w} + \nabla p & = & \lambda\mathbf{u} \\ \nabla\cdot\mathbf{u} & = & 0\ , \end{array}\right\} \tag{1}$$

for some eigenvalue $\lambda \in \mathbb{C}$ and nontrivial eigenfunction $(\mathbf{u}, p)$, satisfying suitable homogeneous boundary conditions. Here the parameter ϵ is the viscosity, which is inversely proportional to the Reynolds number Re. The eigenfunction $(\mathbf{u}, p)$ consists of the velocity $\mathbf{u}$, a vector field in 2D (resp. 3D) and the pressure p (a scalar field) both defined on a 2D (resp. 3D) computational (usually finite) domain. (See, for example [8].) For ease of exposition we shall restrict here to the 2D case.

*Supported by UK Engineering and Physical Sciences Research Council Grant GR/M59075

The computation of the reference velocity $\mathbf{w}$ is, in itself, a large task, especially for 3D problems, or for 2D problems with multiple scales. The eigenvalue problem is generally accepted to be at least as hard as this, and in many cases considerably harder. Thus it is of great interest to find reliable and efficient algorithms for this task.

Restricting for convenience to 2D applications, discretisation of (1) with mixed finite elements yields the algebraic generalised eigenvalue problem:

$$A\mathbf{x} = \lambda M\mathbf{x} \tag{2}$$

where $\mathbf{x} = (\mathbf{U}_1^T, \mathbf{U}_2^T, \mathbf{P}^T)^T$ is a vector of n degrees of freedom approximating $(u_1, u_2, p)^T$, and the matrices A and M take the form:

$$A = \begin{bmatrix} F_{11} & F_{12} & B_1^T \\ F_{21} & F_{22} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix}, \; M = \begin{bmatrix} M_u & 0 & 0 \\ 0 & M_u & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3}$$

Here, for example, F_{11} is the finite element stiffness matrix corresponding to the operator $u_1 \to -\epsilon\Delta u_1 + \mathbf{w} \cdot \nabla u_1 + (\partial w_1/\partial x)u_1$ and M_u is the mass matrix corresponding to the identity operator operating on a typical velocity component. Overall, A is unsymmetric, M is positive semi-definite, both are sparse and in real applications n will be very large ($10^5 - 10^9$ is common). Full details of this type of discretisation can be found, for example, in [8].

In stability analysis only a small number of "dangerous" eigenvalues are required, often those (possibly complex) eigenvalues nearest the imaginary axis. In this context it is usually necessary to perform "shift-invert" iterations, which require repeated solution of systems of the form

$$(A - \sigma M)\mathbf{y} = M\mathbf{x}, \tag{4}$$

for some *shift* σ (which may be near a spectral point) and for various right-hand sides $\mathbf{x}$. Examples of algorithms which require solves of the form (4) include classical inverse iteration, its subspace variant, or Arnoldi's method, based on Krylov spaces.

If σ is close to an eigenvalue of (2) then the condition number of $A - \sigma M$ will be large. Consequently one might think that if a Krylov-based iterative method were used to solve (4) convergence would be slow. In fact for the eigenvalue problem there are likely to be only a small number of isolated eigenvalues near zero and it is well known that in such cases the convergence of Krylov solvers is hardly affected. An alternative approach is the Jacobi-Davidson method (see [16]) which seeks to remove the near singularity in the linear solve. We have not chosen this approah.

Previous algorithms developed at Bath [1] were able to find the eigenvalues of (2) nearest the imaginary axis on the assumption that systems like (4) could be solved exactly. These algorithms form a basis of the stability assessment module in the commercial finite element code ENTWIFE [7]. However, requiring exact solutions of (4) is impracticable for large problems.

In large applications, systems (4) have to be solved iteratively. For the rest of this paper we concentrate on describing a parallel solver for (4) in the special case when A and M arise from Navier-Stokes equations, as in (3). At the end of the paper we describe some sample eigenvalue calculations using Arnoldi's method [14], each iteration of which requires a solve of the form (4).

2 Iterative Method

Our iterative method for (4) is the Flexible Generalised Minimal Residual algorithm (FGMRES), which requires the repeated computation of matrix-vector multiplication with $A-\sigma M$ and converges with a rate determined by the distribution of the spectrum of $A-\sigma M$. The rate of convergence can be accelerated by introducing a (right) preconditioner C and solving, instead of (4), the system

$$(A-\sigma M)C\tilde{\mathbf{y}} = M\mathbf{x} \ , \tag{5}$$

after which $\mathbf{y}$ is retrieved by calculating $\mathbf{y} = C\tilde{\mathbf{y}}$. For best performance, C should firstly be a good approximate inverse for $A-\sigma M$. When solving (5) we require repeated matrix-vector multiplication with $(A-\sigma M)C$ which means secondly that this multiplication operation also needs to be sufficiently fast.

Our experience suggests that even though the shift σ might be near an eigenvalue λ of (2), a good preconditioner for $A-\sigma M$ in the PDE context can be found by using a good preconditioner for A (essentially since A represents a higher order operator than M). Thus from now on we concentrate on describing our preconditioner for A.

Note that we choose to discuss the case of right preconditioning here because it fits in with the standard formulation of FGMRES [15].

3 Preconditioner for A

3.1 Block preconditioners for A

An ideal (right) preconditioner for A suggested by Elman and Silvester [6] is of the form

$$P = \begin{bmatrix} F_{11} & F_{12} & B_1^T \\ F_{21} & F_{22} & B_2^T \\ 0 & 0 & -X \end{bmatrix},$$

where

$$X = \mathbf{B}\mathbf{F}^{-1}\mathbf{B}^T, \ \ \mathbf{F} = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}, \ \ \mathbf{B} = \begin{bmatrix} B_1 & B_2 \end{bmatrix}.$$

This lead to a preconditioned matrix AP^{-1} with a minimal polynomial of degree two, thus guaranteeing convergence in two iterations. This "perfect" preconditioner requires assembling and inverting the Schur complement X and inverting the 2×2 block matrix $\mathbf{F}$ and is not a practical choice. Elman [5] suggested the more practical approximation for X^{-1}:

$$X_B^{-1} = (\mathbf{B}\mathbf{B}^T)^{-1}(\mathbf{B}\mathbf{F}\mathbf{B}^T)(\mathbf{B}\mathbf{B}^T)^{-1}. \tag{6}$$

To motivate the approximation X_B^{-1} note that since $\mathbf{B}$ is rectangular there is no simple formula for X in terms of a product of inverses of its factors. On the other hand $\mathbf{B}\mathbf{B}^T$ acts on the pressure space. X_B^{-1} is a plausible approximation of X^{-1} in that it combines (i) the action of F and (ii) the inverse of $\mathbf{B}\mathbf{B}^T$.

However, numerical experiments with X_B^{-1} indicate that this choice degrades considerably as $h\to 0$ and as the Reynolds number Re increases. Kay, Loghin and Wathen [12, 13] considered the Oseen problem (equivalent to deleting the term $\mathbf{u}\cdot\nabla\mathbf{w}$ in (1)) and for this they proposed instead

$$X_F^{-1} = M_p^{-1}F_pA_p^{-1}, \tag{7}$$

where F_p and A_p are discrete versions of $-\epsilon\Delta + \mathbf{w}\cdot\nabla$ and Δ respectively and M_p is the pressure mass matrix. With this choice the preconditioner is observed to be independent on h and has only a tiny dependence on Re. But the drawback is that (7) requires rediscretisation of the convection-diffusion operator $-\epsilon\Delta+\mathbf{w}\cdot\nabla$ and Laplacian Δ on the pressure space. Our aim is to develop a preconditioner corresponding to (7) which does not need rediscretisation of any of the operators and uses only submatrices of A. We restrict here to the case of the $Q_2 - Q_1$ finite element discretisation (see, for example [8]) and use a suitable restriction of F_{11} to the pressure space instead of explicitly forming F_p. Since $Q_1 \subset Q_2$ we can define a linear interpolation operator Π from pressure to velocity freedoms by simple injection and we use the following approximation of X:

$$X_\Pi^{-1} = \Pi^T F_{11} \Pi (\mathbf{B}\mathbf{B}^T)^{-1}. \tag{8}$$

This is another plausible approximation of X^{-1} based on the same discussion as that following equation (6). Action of the whole block preconditioning step $(\mathbf{u}, p)^T = P^{-1}(\mathbf{w}, r)^T$ is achieved with the following algorithm:

(i) Solve $(\mathbf{B}\mathbf{B}^T)s = r$,
(ii) apply $p = -\Pi^T F_{11} \Pi s$,
(iii) apply $\mathbf{v} = \mathbf{w} - \mathbf{B}^T p$,
(iv) solve $\mathbf{F}\mathbf{u} = \mathbf{v}$.

Note that we assemble the matrix $\mathbf{B}\mathbf{B}^T$ by a sparse matrix multiplication algorithm and the product remains sparse. In our implementation we replace $\mathbf{F}$ in (iv) with a block diagonal approximation $\mathrm{diag}(F_{11}, F_{22})$.

This "block" preconditioner calls for the fast inversion of F_{ii} and $\mathbf{B}\mathbf{B}^T$. These are essentially discrete scalar PDEs and for this we use the DOUG technology [4].

3.2 The DOUG package

The general parallel iterative solver DOUG (Domain Decomposition on Unstructured Grids, see [4, 10]) for linear symmetric and unsymmetric systems in 2D or 3D has been developed at Bath University since 1996. The current version of the code can be used for the efficient solution of scalar elliptic finite element problems defined in terms of element stiffness matrices on parallel computers. (See, for example [2].) The next version of the code will handle also systems with assembled matrices and block systems (like in the case of linearised Navier Stokes equations discussed here). Krylov subspace methods like CG, MINRES, BiCGSTAB and (F)GMRES are available with additive Schwarz type preconditioners with overlapping subdomains.

For parallel scalability in the presence of unstructured grids a nonmatching coarse grid technique is used. The coarse grid (if required) is computed automatically and adaptively so that it reflects the qualitative features of the fine grid.

DOUG is written in Fortran77, the parallelisation is based on MPI (Message Passing Interface, see for example [9]) and is written using a master-slave setup. The master node handles input and output and is responsible for the coarse problem solutions. The slaves are responsible for the subdomain solves. With N slaves the computational domain is split into N subdomains using the mesh partitioning software METIS [11]. To minimise computation time and to achieve efficient parallelisation, DOUG chooses automatically a suitable size of actual

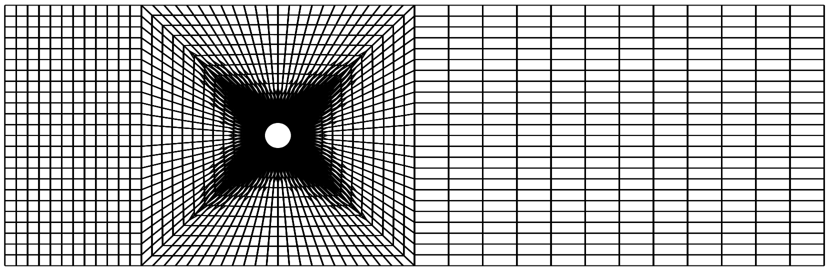

Figure 1: Typical discretisation grid of the flow past a cylinder.

subdomains. For this purpose, the slaves may perform further mesh partitioning to achieve the desired subdomain size. Subdomain solves are performed exactly using the direct multifrontal solver UMFPACK [3]. The LU-factors are computed once at the beginning of computations and stored for each subproblem so that, during the course of the iterative solution, only the corresponding back-substitution phase needs to be called.

For efficient parallel implementation, computations are desired to overlap with communication. With this in mind, DOUG reorders the freedoms suitably for matrix-vector operations so that freedoms at overlaps always come first in the freedom arrays. In the block preconditioner case, freedoms are additionally grouped together according to their block-rank. Thus, our parallel matrix-vector multiplication computations start with overlapping freedoms, after which simultaneously overlapped communication and multiplication operations on inner freedoms are performed. For a more detailed overview of the parallel implementation please refer to [10].

3.3 Test results: a typical solve with A

The test problem of our choice is the problem of flow past a cylinder. A typical discretisation is given in Fig. 1. This mesh has 1728 elements. When the Navier Stokes equations are approximated using $Q_2 - Q_1$ elements on this mesh, the resulting system has 15286 freedoms. The applied velocity is $(1,0)^T$ on the top, bottom and left hand end of the domain, $(0,0)^T$ on the cylinder and $(\mathbf{u},p)^T$ satisfies the outflow conditions $\epsilon\frac{\partial u_1}{\partial x} + p = 0 = \frac{\partial u_2}{\partial x}$ at the right hand end. We compare the number of FGMRES iterations for solving (4) with $\sigma = 0$ with preconditioner

$$P = \begin{bmatrix} F_{11} & 0 & B_1^T \\ 0 & F_{22} & B_2^T \\ 0 & 0 & -X \end{bmatrix}$$

with X^{-1} approximated using either (6) or (8). We employ the stopping criteria that the initial residual norm should be reduced by six orders of magnitude.

In the implementation of each solve with P it is required to implement (subproblem) solves with F_{ii} $i = 1,2$ and $\mathbf{BB}^T$(the latter is done twice in the case of $X^{-1} \approx X_B^{-1}$). We have tested our preconditioner with exact and inexact subproblem solves. In the exact case all the solves with matrices $\mathbf{BB}^T$, F_{11} and F_{22} are done directly. In the case of inexact subsolvers we apply K iterations of an inner FGMRES solver with two level additive Schwarz preconditioner.

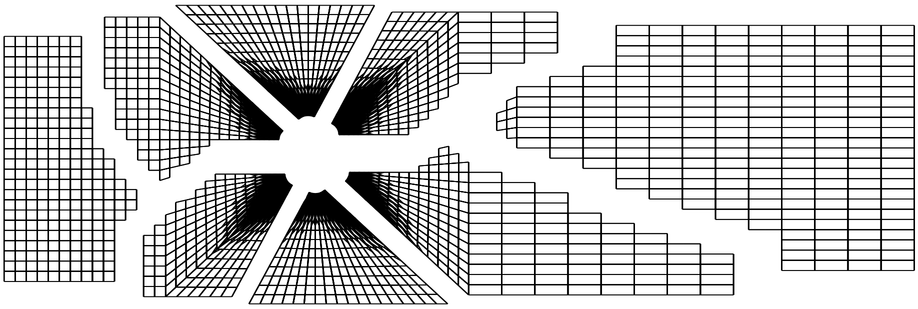

Figure 2: Flow past a cylinder: the grid partitioned into eight subdomains.

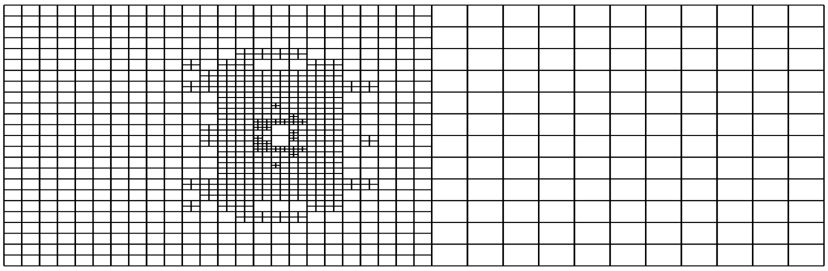

Figure 3: Flow past a cylinder: an adaptively refined coarse grid.

$K = 1$ corresponds to simply replacing the exact matrix to be inverted with one application of its additive Schwarz approximate inverse. In general, the larger the value of K the more precisely the subproblems are solved inside the preconditioner.

An example of the mesh partitioned into subdomains is given in Fig. 2 and an example of an adaptively refined coarse grid is given in Fig. 3. The results with $Re = 1.0$ and $Re = 25.0$ are presented in Table 1 and in Table 2 respectively. Note that the iteration counts in the tables reflect also the exact number of matrix vector operations in FGMRES (since we use a zero initial guess). Generally, the method with X_Π^{-1} converges better. Particularly, with exact subsolves we observe that the preconditioner with X_B^{-1} has some dependence on the meshsize but for X_Π^{-1} with $Re = 1.0$ such dependence is very small and with $Re = 25.0$ does not appear at all. Similar observations

Table 1: Number of iterations for exact and inexact subproblem solvers with $Re = 1.0$.

$Re = 1.0$	X_B^{-1}						X_Π^{-1}				
#dof \ K	exact	16	8	4	2	1	exact	8	4	2	1
6734	54	55	60	130	170	133	50	50	53	66	114
27294	87	104	350	>800	>800	234	63	65	78	259	171
61678	116	188	791	-	-	311	73	78	173	781	205
109886	139	352	-	-	-	382	81	91	281	>800	250

Table 2: Number of iterations for exact and inexact solvers with $Re = 25.0$.

$Re = 25.0$	X_B^{-1}						X_Π^{-1}				
#dof \ K	exact	16	8	4	2	1	exact	8	4	2	1
6734	123	126	136	170	338	244	144	150	191	273	295
27294	194	235	529	>800	-	335	133	146	174	275	384
61678	258	346	>800	-	-	475	132	162	173	261	407
109886	309	621	>800	-	-	593	127	153	183	272	459

Table 3: Relative speedups for solving a moderate size problem on a Linux cluster (for $K = 1$).

#slaves	time (s)	relative speedup
1	452.2	-
2	241.1	1.9
4	125.5	3.6
8	82.03	5.5

appear in [12, 13] for preconditioner (7). Similarly, the preconditioner with X_B^{-1} is much more demanding on the accuracy of the subsystem solves – with the number of degrees of freedom increasing, more subsolve iterations K are needed to improve the iteration counts obtained with $K = 1$. Too few inner solves can damage the convergence. (For the smallest problem $K \geq 4$ is needed to improve on the result with $K = 1$ and at least $K \simeq 16$ inner iterations are needed for the largest problem.) For X_Π^{-1} with $Re = 1.0$ the same behaviour is observed but is less significant. Finally, in the case of the inexact subsolves, X_Π^{-1} with $Re = 25.0$ gives essentially no increase in iteration counts with mesh refinement for each $K > 1$. This is a particularly good feature for stability calculations as in many problems computations over a large range of Re values are required.

We have tested the parallel efficiency of our code in the case of solving a system with 61678 freedoms. We have measured the total time for the solution of the whole problem (including initialisation and data distribution phases.) The same problem is solved in the case of one up to eight slaves. Since the DOUG code optimises the choice of subdivisions even for one processor, we can think of the time for one processor as "best" in this computation. The timings for a nine-node 1.3 GHz AMD Athlon Linux cluster are presented in Table 3. We conclude that the algorithm has a good parallel efficiency. Better efficiency may be obtained for finer discretisations.

4 Test results: The Eigenvalue Problem

To compute the dangerous eigenvalues of the Navier-Stokes system (1), in the stability analysis we use PARPACK – the parallel version of the ARPACK package – which is based on the Implicitly Restarted Arnoldi Method [14] and is suitable for large scale eigenvalue problems. The software provides the option to compute a few (k) eigenvalues of a general $n \times n$ matrix with user specified features. For example, one can ask that eigenvalues of largest real part or largest

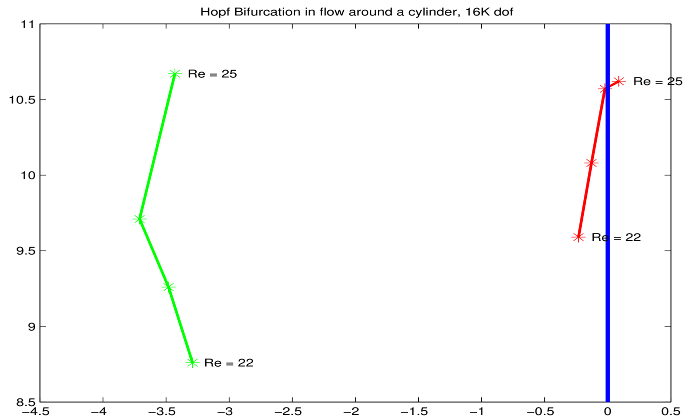

Figure 4: The paths of a few eigenvalues as Re increases, 16K dof.

magnitude be computed. The storage requirements of the algorithm are of the order of $n * k$ locations. From (4) it follows that for our purposes the matrix under consideration is $(A - \sigma M)^{-1}M$, and we call PARPACK in the shift-invert mode. (P)ARPACK uses a reverse communication interface which in this case requires calling one of the following three operations on each outer iteration step (depending on a PARPACK subroutine output parameter value):

a) $\mathbf{y} = (A - \sigma M)^{-1}M\mathbf{x}$,
b) $\mathbf{y} = (A - \sigma M)^{-1}\mathbf{x}$,
c) $\mathbf{y} = M\mathbf{x}$.

After one of these calls control is looped back to the PARPACK subroutine. When convergence is achieved another PARPACK subroutine is called to extract the corresponding computed eigenvalues.

We used the PARPACK computational mode 'LI' – as we know from some preliminary investigations that the desired eigenvalues have large imaginary part. In our case $k = 16$ was chosen which was enough for the eigenvalue nearest to the imaginary axis (using the zero shift $\sigma = 0$) to be found. The eigenvalue tolerance in PARPACK was fixed at 10^{-6}. The paths of a few dangerous eigenvalues as Re increases are depicted in Figs. 4 and 5 for two different discretisations. The figures illustrate the necessity to solve the eigenvalue problem with a large number of degrees of freedom. A Hopf bifurcation for this system occurs at $Re < 24$. An incorrect approximation would result if one only examined the eigenvalues of the small system (Fig. 4). The number of calls to the linear solver requested by PARPACK was ≈ 130 for each Re.

5 Conclusions

In this paper we have discussed some possibilities of building up parallel preconditioners for the linearised Navier-Stokes problem. We investigated two types of preconditioners based on Schur complement techniques. The preconditioners are built up using only existing submatrices of the original problem stiffness matrix and are relatively cheap to construct. For parallelisation the Domain

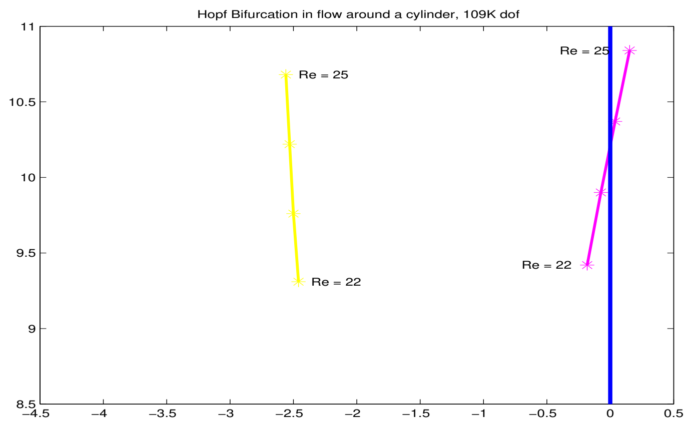

Figure 5: The paths of a few eigenvalues as Re increases, 109K dof.

Decomposition technique is used and for the implementation the DOUG package is extended to block systems. The parallel preconditioner gives relatively good speedup results. We made a sample stability analysis of a Navier-Stokes problem problem using PARPACK together with our parallel solver to calculate a few eigenvalues nearest to the imaginary axis. Although with PARPACK the number of calls to the solver is relatively small, we think there is a room for improvement by using some other techniques. The design of such methods is subject to further research. Further investigation is also needed to extend the given technique to other types of finite element discretisations.

References

[1] K. A. Cliffe, T. J. Garrett and A. Spence, Eigenvalues of the discretized Navier-Stokes equations with application to the detection of Hopf bifurcations. Advances in Computational Mathematics **1**, (1993), pp. 337-356.

[2] K. A. Cliffe, I.G. Graham, R. Scheichl and L. Stals, Parallel computation of flow in heterogeneous media modelled by mixed finite elements, Journal of Computational Physics **164** (2000), pp. 258 -282.

[3] T.A. Davis and I.S. Duff, A combined unifrontal/multifrontal method for unsymmetric sparse matrices. ACM Transactions on Mathematical Software, vol. **25**, no. 1, pp. 1-19, 1999.

[4] DOUG: http://www.maths.bath.ac.uk/~parsoft.

[5] H. C. Elman, Preconditioning for the steady-state Navier-Stokes equations with low viscosity, SIAM J. Sci. Comp. **20** (1999), pp. 1299-1316.

[6] H. C. Elman and D. J. Silvester, Fast nonsymmetric iterations and preconditioning for Navier-Stokes equations, SIAM J. Sci. Comp. **17** (1996), pp. 33-46.

[7] ENTWIFE: http://www.entwife.com.

[8] P. M. Gresho and R. L. Sani, Incompressible Flow and the Finite Element Method. Advection-diffusion and isothermal laminer flow. John Wiley & Sons, 1998.

[9] W. Gropp, E. Lusk and A. Skjellum, Using MPI - Portable Parallel Programming with the Message Passing Interface. MIT Press, 2nd Edition, 1999.

[10] M. J. Hagger, Automatic domain decomposition on unstructured grids DOUG, Advances in Computational Mathematics **9** (1998), pp. 281 - 310.

[11] G. Karypis and V. Kumar, METIS: Unstructured graph partitioning and sparse matrix ordering system, Department of Computer Science, University of Minnesota Report, Minneapolis, Aug. 1995.

[12] D. Kay and D. Loghin, A Green's function preconditioner for the steady-state Navier-Stokes equations. Oxford Numerical Analysis Group Research Report NA-99/06, 1999.

[13] D. Kay, D. Loghin and A. Wathen. A preconditioner for the steady-state Navier-Stokes equations. SIAM J. Sci. Computing, to appear.

[14] R. B. Lehoucq, D. C. Sorensen and C. Yang, ARPACK user's guide: Solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods. http://www.caam.rice.edu/software/ARPACK.

[15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. SIAM J. Sci. Comput., **14** (1993), pp. 461-469.

[16] G. Sleijpen and H. van der Vorst. Jacobi-Davidson Methods (Section 4.7 pp 88 - 105) In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia, 2000.