GMRES Method and its Parallel Application to Navier-Stokes Equations in Stability Assessment

Eero Vainikko

Tartu University Institute of Technology

joint work with:

Konstantin Skaburskas

Tartu University Institute of Technology

Ivan G. Graham, Alastair Spence

University of Bath, United Kingdom

Pedase, Arvutiteaduse teooriapäevad 2003

- Krylov subspace methods
- Preconditioning
- GMRES method
- Parallel implementation of GMRES
- Inner-outer GMRES method
- Stability Assessment for discretised PDEs
- Navier-Stokes Flows and DOUG

Suppose we are solving a linear system of equations

$$A\mathbf{x} = \mathbf{b}$$

with **large**, **sparse** $n \times n$ matrix A.

In Krylov subspace methods, the solution is designed as a linear combination of Krylov vectors (forming Krylov subspace)

$$\mathcal{K}^{(i)}(\mathbf{v}) = \{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, ..., A^{i-1}\mathbf{v}\}$$

where $\mathbf{v}$ is some initital guess to the solution. The approximate solution $\mathbf{x}$ is chosen such that it minimises the residual $\mathbf{r} = A\mathbf{x} - \mathbf{b}$. The examples of Krylov methods include CG, BiCGSTAB, MINRES and others. We are looking here at GMRES methods which are suitable for solving systems with **unsymmetric** matrices A.

Preconditioning. For better convergence, often some preconditioner M^{-1} is used, such that

$$M^{-1} \sim A^{-1}$$

but on contrary to A, the inverse of M is easy to compute. Here we are looking at Domain Decomposition preconditioners which is a natural way to parallelise the problem solution process. Depending on, weather left of right preconditioning is used, the underlying Krylov subspace is of the form:

$$\mathcal{K}^{(i)}_{\text{left}M}(\mathbf{v}) = \{\mathbf{v}, M^{-1}A\mathbf{v}, (M^{-1}A)^2\mathbf{v}, ..., (M^{-1}A)^{i-1}\mathbf{v}\}$$

$$\mathcal{K}^{(i)}_{\text{right}M}(\mathbf{v}) = \{\mathbf{v}, AM^{-1}\mathbf{v}, (AM^{-1})^2\mathbf{v}, ..., (AM^{-1})^{i-1}\mathbf{v}\}$$

Algorithm Left-preconditioned PGMRES(m) method:

Choose initial guess $\mathbf{x}^{(0)}$
for j=1,2,...
  Solve $\mathbf{r}$ from $M\mathbf{r} = \mathbf{b} - A\mathbf{x}^{(0)}$
  $\mathbf{v}^{(1)} = \mathbf{r} / \|\mathbf{r}\|_2$
  $\mathbf{s} := \|\mathbf{r}\|_2 \mathbf{e}_1$
  for i=1,2,...,m
    Solve $\mathbf{w}$ from $M\mathbf{w} = A\mathbf{v}^{(i)}$
    for k=1,...,i
      $h_{k,i} = (\mathbf{w}, \mathbf{v}^{(k)})$
      $\mathbf{w} = \mathbf{w} - h_{k,i}\mathbf{v}^{(k)}$
    end
    $h_{i+1,i} = \|\mathbf{w}\|_2$
    $\mathbf{v}^{(i+1)} = \mathbf{w} / h_{i+1,i}$

apply $J_1,...,J_{i-1}$ on $(h_{1,i},...,h_{i+1,i})$
construct J_i, acting on ith and $(i+1)$st component
of $h_{:,i}$, such that $(i+1)$st component of $J_i h_{:,i}$ is 0
$\mathbf{s} := J_i \mathbf{s}$
if $\mathbf{s}(i+1)$ is small enough then (UPDATE$(\tilde{\mathbf{x}}, i)$ and quit)
end
end
!*** In this scheme UPDATE$(\tilde{\mathbf{x}}, i)$is:
Compute $\mathbf{y}$ as the solution of $H\mathbf{y} = \tilde{\mathbf{s}}$, in which
the upper $i \times i$ triangular part of H has $h_{i,j}$
as its elements (in least squares sense if H
is singular),
$\tilde{\mathbf{s}}$ represents the first i components of $\mathbf{s}$
$\tilde{\mathbf{x}} = \mathbf{x}^{(0)} + y^{(1)}\mathbf{v}^{(1)} + y^{(2)}\mathbf{v}^{(2)} + ... + y^{(i)}\mathbf{v}^{(i)}$
$s^{(i+1)} = \|\mathbf{b} - A\tilde{\mathbf{x}}\|_2$
if $\tilde{\mathbf{x}}$ is an accurate enough approximation then quit
else $\mathbf{x}^{(0)} = \tilde{\mathbf{x}}$.

There are 3 key issues concerning an implementation of the given algorithm:

- Minimising the communication cost
- Storage problem – how to choose m but still get fast convergence?
- Preconditioning issues

Minimising the communication cost. In the previous algorithm:

Modified Gram-Schmidt:

```
for k=1,...,i
```
$$h_{k,i} = (\mathbf{w}, \mathbf{v}^{(k)})$$
$$\mathbf{w} = \mathbf{w} - h_{k,i}\mathbf{v}^{(k)}$$
```
end
```

For // implementation – Classical Gram-Scmidt algorithm much better:

$$h_{(1:i),i} := (\mathbf{w}, \mathbf{v}^{(1:i)})$$
$$\mathbf{w} := \mathbf{w} - h^T_{(1:i),i}\{\mathbf{v}^{(1)}\mathbf{v}^{(2)}\ldots\mathbf{v}^{(i)}\}$$

Problem – loss of orthogonality. Therefore, **Iterated** Classical Gram-Schmidt orthogonalisation algorithm can be used:

$h_{(1:i),i} := \mathbf{0}$
`for j=1,2 !(3)`
  $h_{(1:i),i} := h_{(1:i),i} + (\mathbf{w}, \mathbf{v}^{(1:i)})$
  $\mathbf{w} = \mathbf{w} - h_{(1:i),i}{}^{T}\{\mathbf{v}^{(1)}\mathbf{v}^{(2)}...\mathbf{v}^{(i)}\}$
`end`

Benefits:

* Reduced number of dotproduct operations

* Possibility of using BLAS2 subroutines (`[DZ]GEMV()`)

* In parallel MPI implementation: `ALLREDUCE` of i values in a single call

In the PGMRES(m) method the preconditioner M^{-1} was fixed.

Even in the case when the system $M\mathbf{x} = \mathbf{y}$ is solved inexactly, with some iterative procedure, the actual preconditioner varies from iteration to iteration.

FGMRES (Flexible GMRES) method can be used:

The modifications to the PGMRES algorithm can be outlined as follows:

* Instead of using left preconditioning, right preconditioning is used:

$$\begin{aligned} AM^{-1}\mathbf{y} &= \mathbf{b} \\ \mathbf{x} &= M\mathbf{y} \end{aligned}$$

* In the algorithm, also the intermediate vectors $M^{-1}\mathbf{v}$ are stored as well.

* Use `UPDATE` $(M^{-1}\tilde{\mathbf{v}}, i)$ to compute the solution in the end.

What about the idea of preconditioning FGMRES method with some version of PGMRES itself?

Most often the inner GMRES method is Left-preconditioned PGMRES, with the preconditioner M^{-1}.

The result is called GMRES* or inner-outer GMRES method.

Benefits of the method:

* Better convergence behaviour than PGMRES(m) method

* On ith iteration, the unused allocated vectors $\mathbf{v}^{(i+1)}, \mathbf{v}^{(i+2)}, ..., \mathbf{v}^{(\mathbf{m})}$ of $V_{\text{outer}} = \{\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, ..., \mathbf{v}^{(\mathbf{m})}\}$ can be used to store V_{inner}.

* Possible variation – in the inner iteration method to orthogonalise also against $(V_{\text{outer}})_{(:,1}$ – sometimes giving benefit, (but not always for some unknown reason.)

Motivation: Stability Assessment for discretised PDEs

$$\frac{\partial w}{\partial t} = \mathcal{F}(\mathbf{w}, R), \text{+initial and boundary conditions}$$

Steady state $w = w(R), \quad R \in \mathbb{R}$. Stable?

- Solve eigenvalue problem $Aw = \lambda w$ for λ near Imaginary axis where $A = \mathcal{F}_x(x(R), R)$

Our particular case: **Navier-Stokes Flows**

Given a steady solution (w,q), **Eigenvalue problem:**

$$\begin{aligned} -\varepsilon\Delta\mathbf{u} + w\cdot\nabla\mathbf{u} + \mathbf{u}\cdot\nabla w + \nabla p &= \lambda\mathbf{u} \\ \nabla\cdot\mathbf{u} &= 0\,, \end{aligned}$$

+ Homogeneous boundary conditions.

Discretisation with mixed finite elements (e.g. in 2D):

$$Ax = \lambda Mx, \quad x = (\mathbf{U}_1^T, \mathbf{U}_2^T, \mathbf{P}^T)^T$$

$$A = \begin{bmatrix} F_{11} & F_{12} & B_1^T \\ F_{21} & F_{22} & B_2^T \\ B_1 & B_2 & 0 \end{bmatrix}, \quad M = \begin{bmatrix} \mathcal{M} & 0 & 0 \\ 0 & \mathcal{M} & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

e.g. $F_{11}\mathbf{U}_1 \approx -\varepsilon\Delta u_1 + w\cdot\nabla u_1 + (\partial w_1/\partial x)u_1$, $\mathcal{M}\mathbf{U}_1 \approx u_1$.

A is unsymmetric, M is positive semi-definite. $n \approx 10^5 \rightarrow$

Eigenvalue solvers for : $Ax = \lambda Mx$

For shift σ near an eigenvalue λ,

$$Ax = \lambda Mx \Longleftrightarrow (A - \sigma M)^{-1} Mx = (\lambda - \sigma)^{-1} x$$

Inverse Iteration, Subspace Iteration:

$$\begin{array}{lrcl} \text{Solve:} & (A - \sigma^i M) y^i & = & M x^i \quad (*) \\ \text{Normalise:} & x^{i+1} & = & y^i / \|y^i\| \end{array}$$

More generally: Arnoldi's method on $(A - \sigma^i M)^{-1} M$

In all cases: require solve of form (*).

Singular as $\sigma^i \rightarrow$ spectrum.

Large n?

Solve $(A - \sigma M^i)y^i = Mx^i \ (*)$ iteratively or with parallel multifrontal methods.

Our Choice: **Iterative methods using Domain Decomposition.**

Fast Parallel inner solvers:

DOmain Decomposition on Unstructured Grids

DOUG Graham, Haggers, Stals, Vainikko, 1997 - 2003

- solves systems of steady state PDEs
- User-defined discretisation on unstructured grids
- automatic parallelisation and load-balancing
- Portable

- 2D and 3D
- 1 and 2-level Additive Schwarz method
- two-level mesh partitioning
- Automatic Domain Decomposition and coarse grid generation
- Adaptive coarse grid refinement
- Elemental form and assembled form input of stiffness matrices
- WWW-interface

Parallel implementation based on:

- Message Passing Interface (**MPI**) - **LAM** and **MPICH** implementations
- **UMFPACK2** - current underlying solvers
- **METIS** - graph partitionining software
- **BLAS**

Non-blocking communication where at all possible

Preconditioned iterative methods: Following operations required:

Vector update: $\mathbf{z} = \mathbf{x} + \mathbf{y}$

Matrix-vector multiply: $\mathbf{y} = A\mathbf{x}$

Dot products: $(\mathbf{x}, \mathbf{y})$

Solution of systems: $P\mathbf{z} = \mathbf{r}$ for some preconditioner P.

PCG, MINRES, BICGSTAB, Inner-outer **PGMRES** with right or left preconditioning

Navier-Stokes Preconditioner

An ideal preconditioner for A (Elman and Silvester 96)

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

where

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

Our strategy:

$$\begin{bmatrix} F_{11} & 0 & B_1^T \\ 0 & F_{22} & B_2^T \\ 0 & 0 & -X \end{bmatrix}^{-1}$$

Choice of X :

a) $X_M^{-1} = M_p^{-1}/\varepsilon$ (Elman & Silvester, 1996)

b) $X_F = M_p^{-1} F L_p^{-1}$ (Kay & Loghin, 1999)

c) $X_B^{-1} = (BB^T)^{-1}(BFB^T)(BB^T)^{-1}$ (Elman, 1999)

d) $X_\Pi^{-1} = \Pi F_{11}^T \Pi (\mathbf{B}\mathbf{B}^T)^{-1}$, where Π – lin. interpolation operator form pressure to velocity freedoms.

$X_{\Pi}^{-1} = \Pi F_{11}^{T} \Pi (\mathbf{B}\mathbf{B}^{T})^{-1}$ for $(Q_2 P_0)$ elements

Action of the whole block preconditioning step $(\mathbf{u}, p)^T = P^{-1}(w, r)^T$ is achieved with the following algorithm:

(i) Solve $(\mathbf{B}\mathbf{B}^T)s = r$, ($\mathbf{B}\mathbf{B}^T$ formed with sparse matrix mult.)

(ii) apply $p = -\Pi^T F_{11} \Pi s$,

(iii) apply $\mathbf{v} = w - \mathbf{B}^T p$,

(iv) solve $\mathbf{F}\mathbf{u} = \mathbf{v}$.

Whole preconditioner combined with $X_B^{-1} = (BB^T)^{-1}(BFB^T)(BB^T)^{-1}$ (multiplicatively) for $(Q_2 P_{-1})$ elements

The method: **2-level Additive Schwarz method with minimal overlap**

Global $N \times N$ stiffness matrix

$$A = \begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} = \begin{pmatrix} F^{11} & F^{12} & (B^1)^T \\ F^{21} & F^{22} & (B^2)^T \\ B^1 & B^2 & 0 \end{pmatrix} = \sum_{e \in E} A_e$$

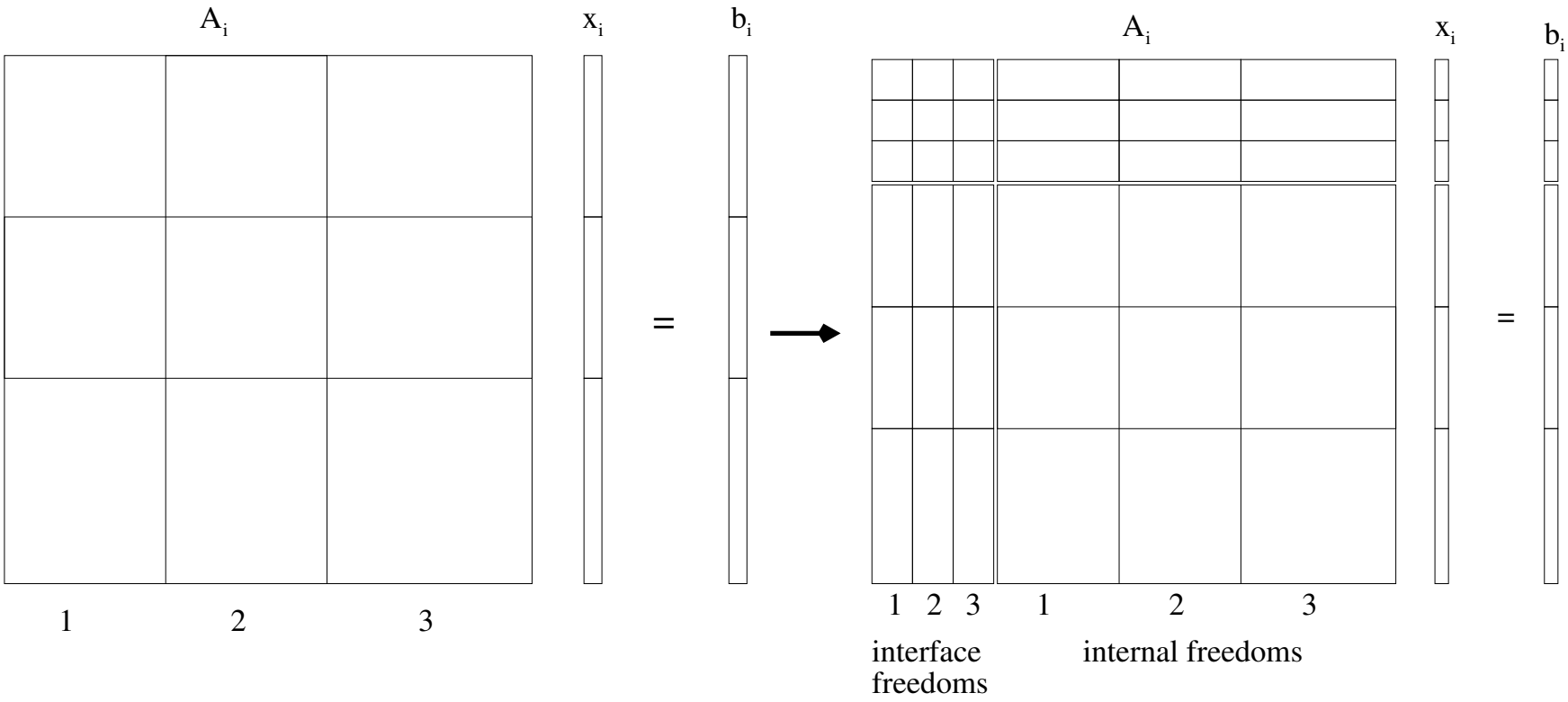
Set of elements E is partitioned into subsets $E_i \; i = 1, ..., N_p$

For each i the contribution to the global stiffness matrix:

$$A_i = \sum_{e \in E_i} A_e$$

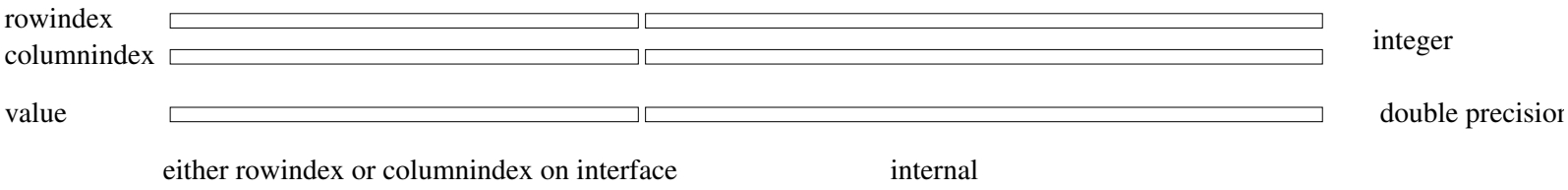
Reordering of the system freedoms.

For each slave *i* we make a reordering:



Note that:

- further reordering available inside each block
- A_i stored in sparse triple format



Matrix-vector multiply $y = Ax$ operations. On each slave i:

* Calculate $y = A_i x$ on the interfaces

* Start nonblocking sends/receives to/from the corresponding neighbours

* Calculate $y = A_i x$ on internal freedoms

* Add on the interfaces after each receive has ended.

Partitioning

Using **METIS** on master

Connected graph, element stiffness matrices as graph nodes; graph edges where two elements share an edge (2D) or a face (3D)

Subpartitioning on slaves

- to obtain optimal size of subproblems (default 1400 DoF)

For each subpartition j and each diagonal block k define restriction $\widetilde{A_j^k}$:

$$(\widetilde{A_j^k})_{pq} = \sum_{e \in E} (A_e^{kk})_{pq}, \quad \text{for } p,q \in \Phi_j^k$$
$$(\widetilde{A_j^k})_{pq} = 0 \qquad\qquad \text{otherwise}$$

$$\widetilde{A_j^k} = R_j^k A^{kk} (R_j^k)^T \text{, where } A^{kk} = \begin{cases} F^{kk}, & k = 1,2 \\ BB^T \text{ or } M_p, & k = 3 \end{cases} .$$

A_0^k - approx. of F^{kk} $(k=1,2)$ or BB^T $(k=3)$ on the coarse mesh

R_0^T $(=(R^{Hh})^T)$ – linear interpolation from the coarse to the fine mesh

2-level Additive Schwarz preconditioner:

$$M_{\mathrm{ASC}}^{-1} = R_0^T (A_0^k)^{-1} R_0 + \sum_{i=1}^{p} R_i^T \widetilde{A_i^k}^{-1} R_i$$

1-level Additive Schwarz preconditioner:

$$M_{\mathrm{AS}}^{-1} = \sum_{i=1}^{p} R_i^T \widetilde{A_i}^{-1} R_i$$

Implementation of the algorithm

Master-slave setup

master initialisation; coarse grid poblem solves

slaves subdomain solves

Vector updates $\mathbf{z} = \mathbf{x} + \mathbf{y}$ **–** in parallel implementation no communication needed

Dot Products: Produce unique sets of freedoms:

$$\begin{aligned}\overline{\Phi}_1 &= \Phi_1 \\ \overline{\Phi}_i &= \Phi_i \setminus \{\overline{\Phi}_1 \bigcup ... \bigcup \overline{\Phi}_{i-1}\}, \quad i = 2, ..., N_p \end{aligned}\ ,$$

where N_p is the number of slaves.

Dot product operation is given by:

$$(\mathbf{x}, \mathbf{y}) = \sum_i \sum_{p \in \overline{\Phi}_i} (\mathbf{x}_i)_p (\mathbf{y}_i)_p .$$

MPI_ALLREDUCE with MPI_SUM flag.

Preconditioner solve

Needed:

$$\mathbf{z} = \sum_i \widetilde{A^k}_i^{-1} \mathbf{x}$$

Parallel implementation:

- $\widehat{\mathbf{z}}_i = \widetilde{A_i^k}^{-1} \mathbf{x}_i$

- Nearest-neighbour communication like in matrix-vector multiply

Coarse mesh implementation

* The coarse mesh covers all of the fine mesh

* No coarse mesh element lies completely outside the fine mesh

* **Prolongation and restriction** – in matrix representation

* Coarse matrix calculation (computed in parallel)

$$A_0^k = R_0 F^{kk} R_0^T = R_0 \left(\sum_{e \in E} F_e^{kk} \right) R_0^T, \; k = 1, 2$$

$$A_0^3 = R_0 B B^T R_0^T$$

Automatic coarse grid generation - 2 conflicting aims:

- adequate representation of the PDE
- complexity which does not adversely affect the overall parallel performance of the algorithm

* **Choice of coarse mesh**

- The base coarse mesh + Adaptive refinement technique:

2 main parameters:

- max # of fine grid freedoms per coarse cell
- max # coarse nodes

Parallel Preconditioner algorithm (in solution with F^{kk} $(k = 1, 2)$ or with BB^T :)

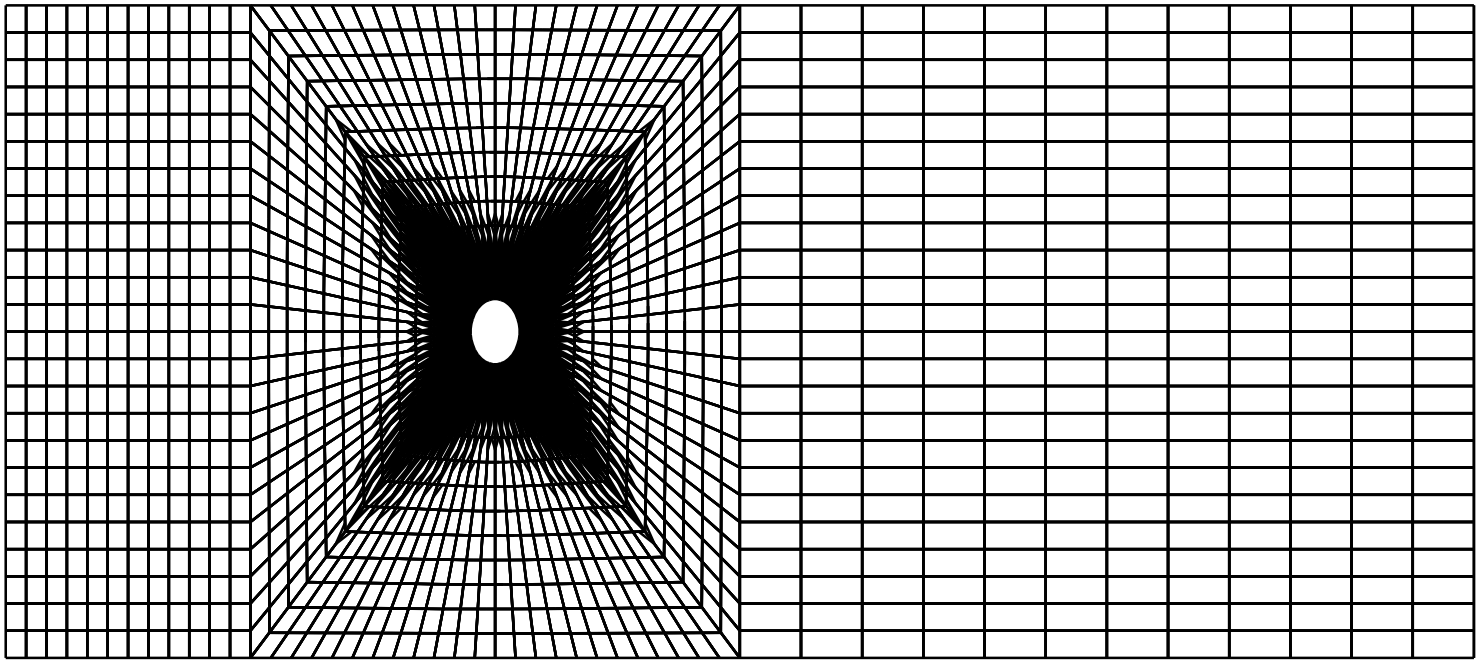
1. Restriction operation $R_0 \mathbf{x}_i$

2. Start the non-blocking receive for the result from the master

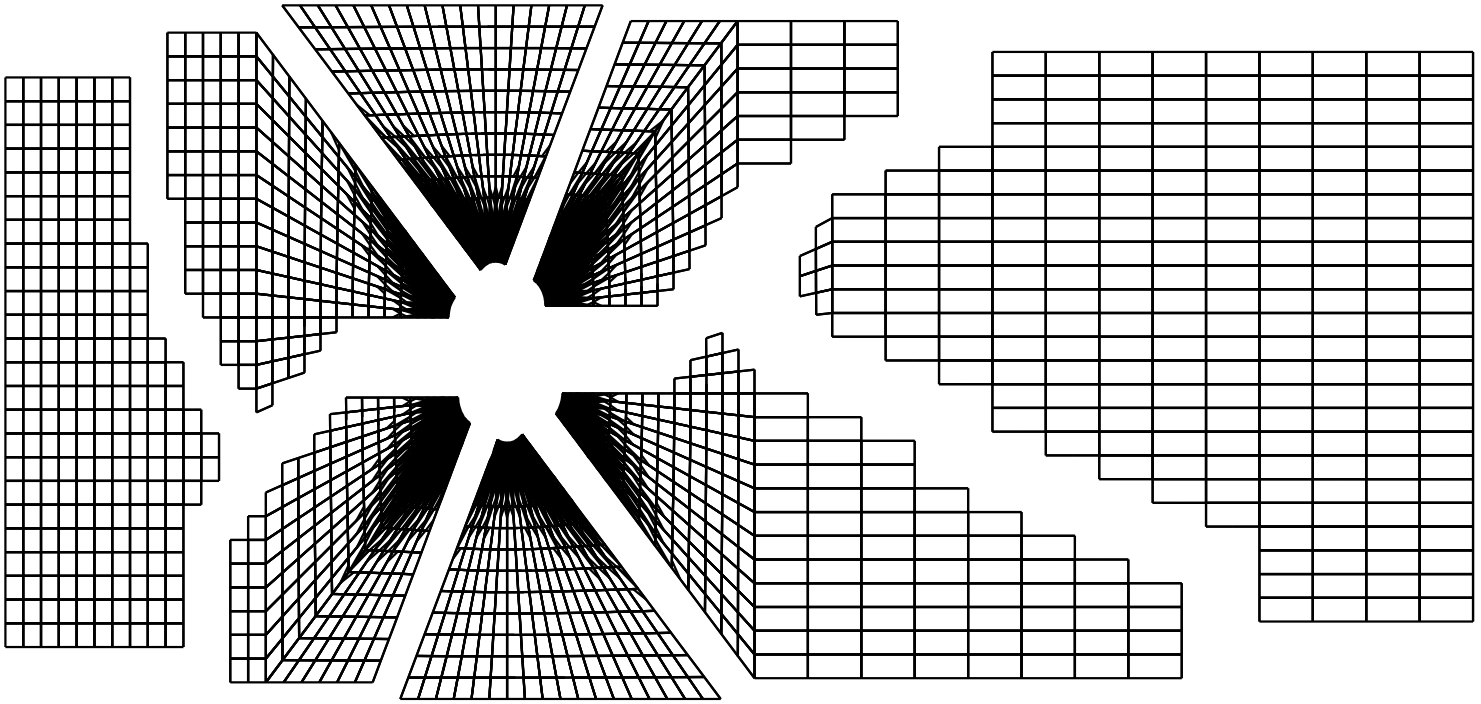
3. Compute the local subdomain solve(s)

4. Send updates on shared entries to the other slaves

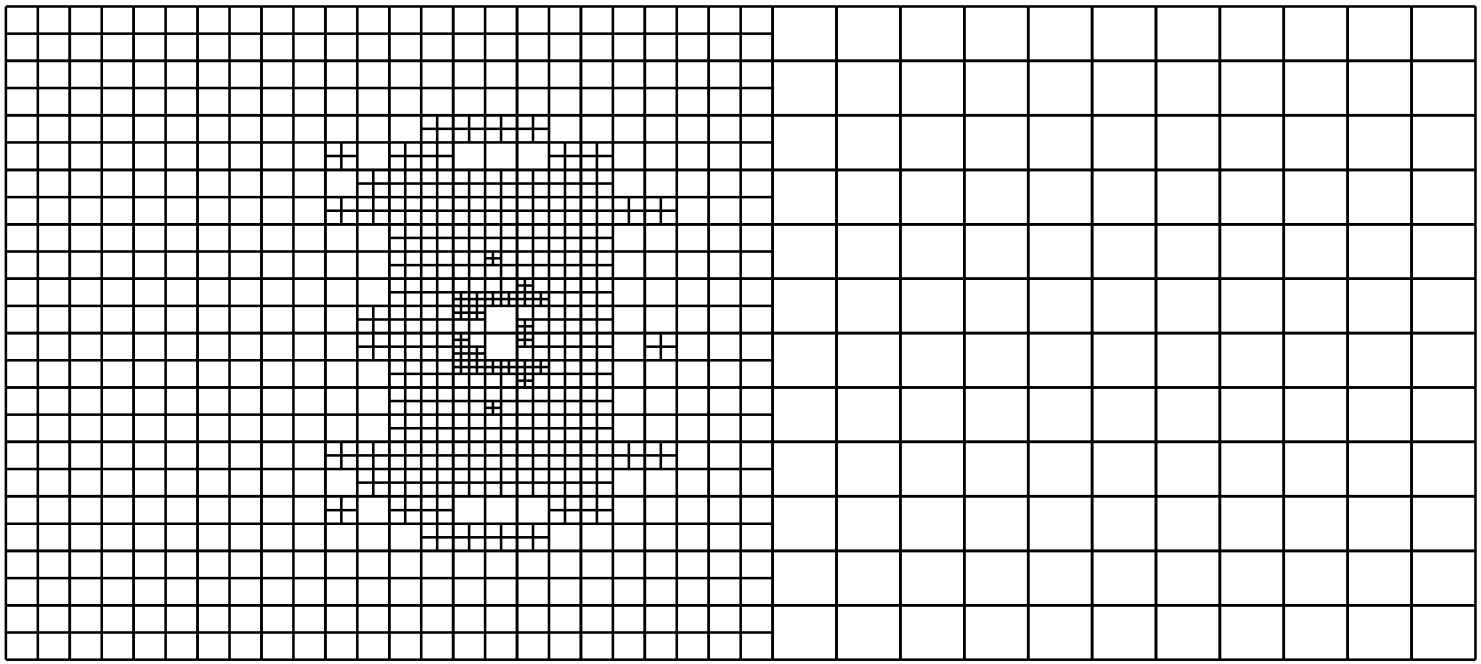
5. Wait for the shared entry receives *or* the result from the master. If the result from the master has arrived then immidiately prolong it and add the result to the local vector.



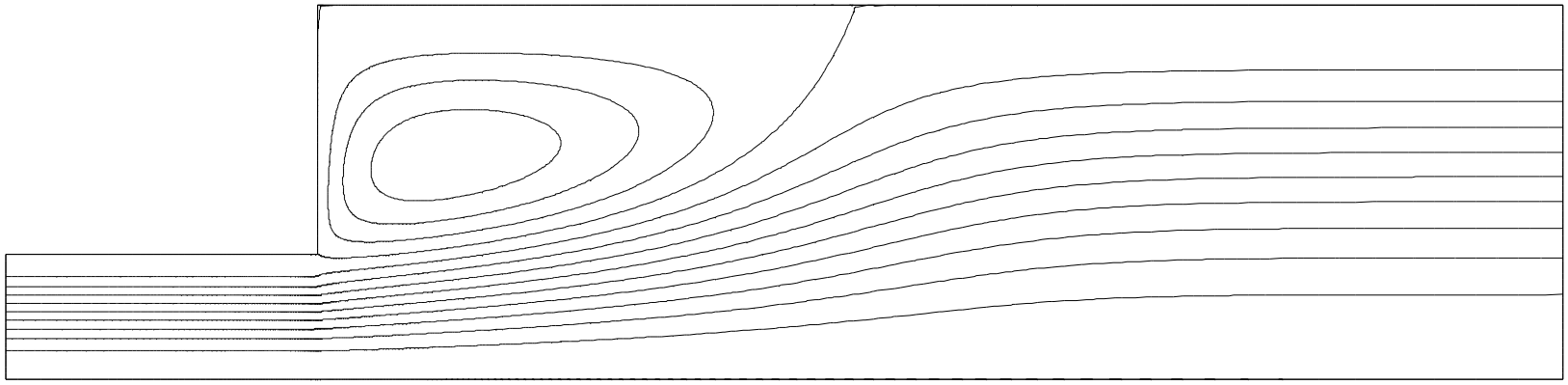
Typical discretisation grid of the flow past a cylinder.



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



Flow past a cylinder: an adaptively refined coarse grid.



Flow in an expanding pipe: $Re = 100$

Timings and relative speedups for the flow past a cylinder and the expanding pipe problem.

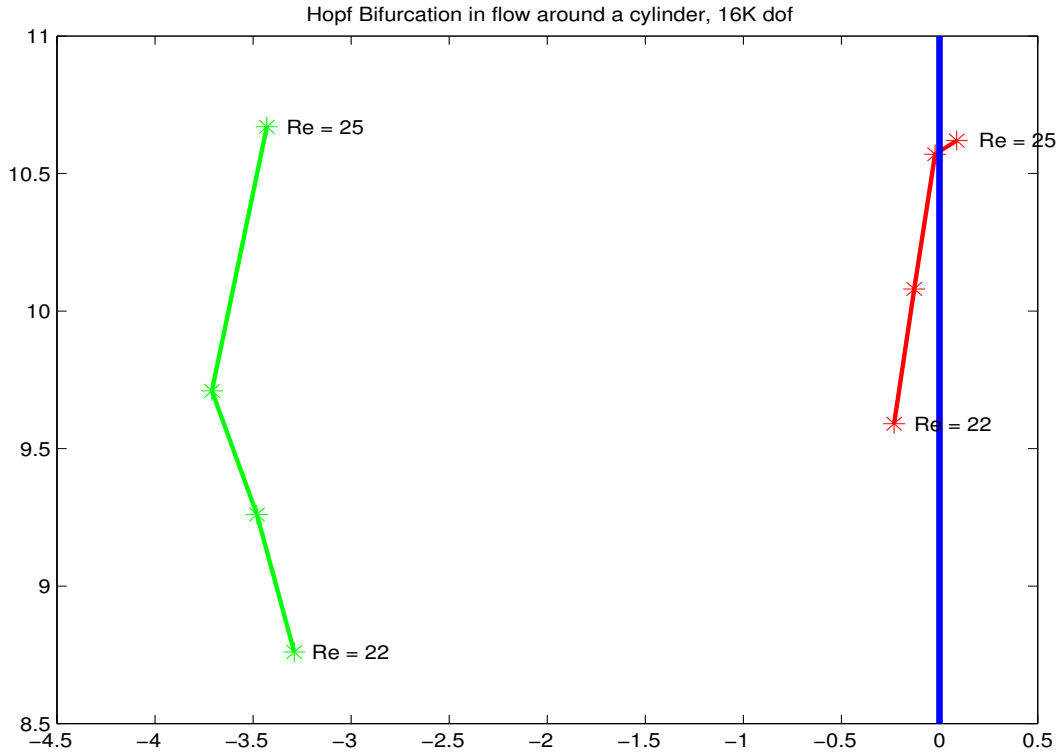
	Flow around a Cylinder			Flow in an expanding pipe		
	Block prec., 52158 DoF			Whole prec., 96400 DoF		
#Slaves	#it.	time(s)	rel.s/o	#it.	time(s)	rel.s/o
1	1100	1507	-	3200	7237.7	-
2	1000	712.8	2.11	3400	4051.0	1.79
4	1100	379.2	3.97	3400	2291.3	3.16
8	900	193.6	7.78	2800	761.7	9.50
12	1000	174.1	8.66	4411	946.3	7.65
16	1000	175.5	8.59	3600	728.8	9.93
20	900	147.8	10.20	3800	774.5	9.35

Parallel Efficiency in solving the eigenvalue problem

Arnoldi (**PARPACK**), 20 eigenvalues, 27K freedoms

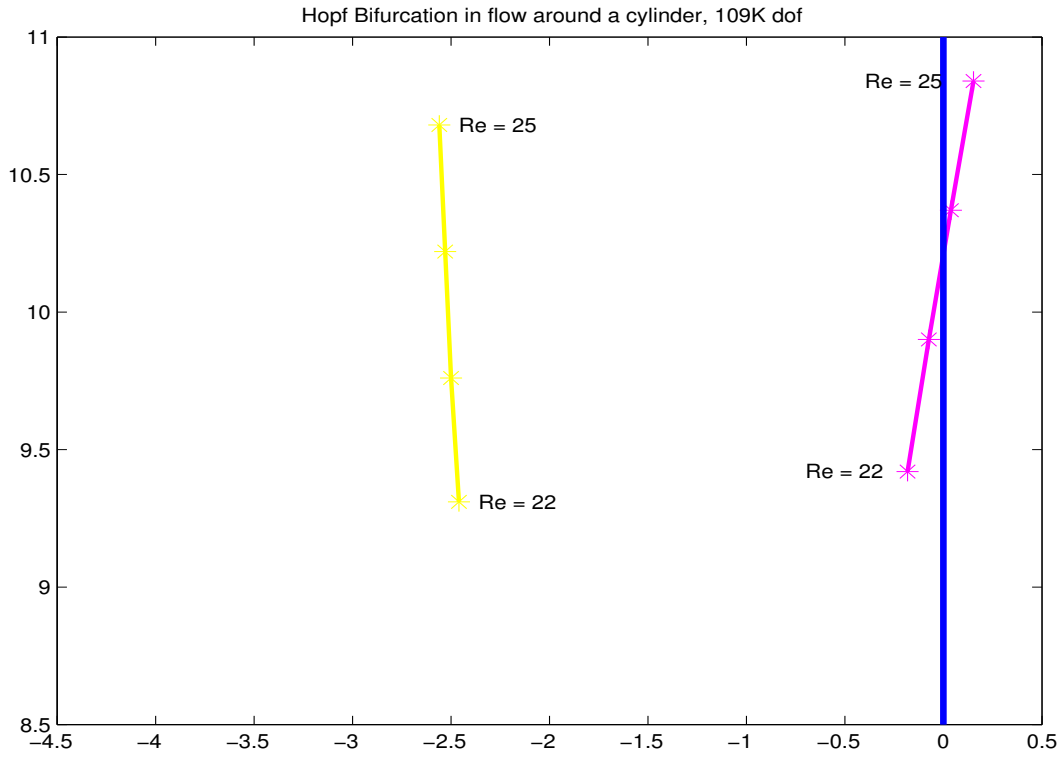
Processors	time (s)	relative speedup
1	50938	-
2	22673	2.25
4	11763	4.33
8	6573	7.75

Hopf bifurcation in flow around a cylinder



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

Hopf bifurcation in flow around a cylinder



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

Computing eigenvalues with PARPACK and the combined method of refining rough PARPACK eigenvalues by the inverse iteration methods

Strategy	1.PARPACK	2.PARPACK combined with inverse iteratio		
#DoF	Total time	PARPACK time	Inverse it. time	Total time
Flow past a cylinder				
33278	2270.6	471.1	53.4	524.5
52158	3732.4	714.9	260.6	975.5
75262	5282.6	114.7	261.0	1375.7
Expanding pipe problem				
96400	71952.0	13566.8	2773.6	16340.4

Ongoing and future work

* Releasing the new version of DOUG code (DOUG 2)

* Reimplementation in an object oriented environment.
Fortran95.

* Fault tolerance and parallel programming
Problem: MPI standard says – FT is to be taken care by the user
A prototype communication model for DOUG has been implemented – based on LAM MPI implementation.

* Research in the direction of possibility of using the framework of multiagent systems for designing parallel adaptive computational environments.

* Adapting the DOUG code to the GRID environment.