HYMLS: A parallel solver for steady coupled fluid-transport equations

Fred Wubs*, Weiyan Song* and Jonas Thies†

* Johann Bernoulli institute for mathematics and computing science
  University of Groningen, the Netherlands
  f.w.wubs@rug.nl

† Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Simulations- und
  Softwaretechnik, Germany
  jonas.thies@dlr.de

PMAA14
Lugano, July 2, 2014
Problem Setting
**Objective**

- 3D CFD problems, academic flow problems and geophysical applications (ocean models) \( M \frac{du}{dt} = F(u, \mu) \) (possibly with noise)
- Compute branches of steady states: from \( F(u, \mu) = 0 \), compute \( u(\mu) \) as function of \( \mu \).
- Compute stability of solution: solve eigenvalue problem \( \lambda M v = F_u(u(\mu), \mu) v \)
- Identify bifurcation points: compute \( \mu \) for which \( \lambda = 0 \)
- If \( u(\mu) \equiv 0 \) is a steady solution, we can find all bifurcation points from this solution at once from the eigenvalue problem \( F_u(0, \mu) v = 0 \)

**Key challenge:** large sparse linear systems with the (shifted) Jacobian
Discretization

\[
\frac{\partial \vec{u}}{\partial t} + \mathcal{N}(\vec{u}, \vec{u}) + \frac{1}{Re} \mathcal{L} \vec{u} + \nabla p = 0 \\
\nabla \cdot \vec{u} = 0
\]

- Note that \( \mathcal{N}(\vec{u}, \hat{\vec{u}}) \) is a bilinear form → in operator form \( C(\vec{A}\vec{u}. \ast \vec{B}\hat{\vec{u}}) \).
- On closed domains it holds that \( \int_\Omega \vec{u}\mathcal{N}(\vec{u}, \hat{\vec{u}})d\Omega = 0 \) for any divergence free \( \hat{\vec{u}} \). **To be preserved in discretization**
- Discretize (here second-order symmetry-preserving finite volumes on C-grid) → no artificial diffusion
- Structure of resulting linear systems (Saddle-point matrix):

\[
\begin{pmatrix}
\frac{1}{Re} \mathcal{L} + \mathcal{N}(\vec{u}) & \text{Grad} \\
\text{Div} & 0
\end{pmatrix}
\begin{pmatrix}
\vec{u} \\
p
\end{pmatrix}
= \begin{pmatrix}
f_{\vec{u}} \\
f_{p}
\end{pmatrix}
\]

- Here \( \mathcal{N}(\vec{u}) = C \ast \text{diag}(\vec{A}\vec{u}) \ast \vec{B} + C \ast \text{diag}(\vec{B}\hat{\vec{u}}) \ast \vec{A} \rightarrow \text{store } \vec{A}, \vec{B} \) and \( C \) during initialization together with linear parts
Figure shows steady solutions at Reynolds numbers ranging from 100 to 2000 on $64^3$ grid.

- Picard iteration (Oseen equation) stagnates for $Re > 200 \rightarrow$ full Jacobian in linear system to be solved.
- Jacobian will eventually have positive eigenvalues.
- Big continuation steps should be possible (here 500).
Algorithmic overview

Parallel Data structure Trilinos: Epetra

Continuation program:

- Initialization
  - Partitioning + maps needed for parallelization
  - Set up templates for the matrix
  - Initialize solution

- Continuation using LOCA
  - Prediction
  - Correction using NOX.
    - Solve linear system using HYMLS
  - Eigenvalue computation using ANASAZI
Our workhorse: HYMLS
Dropping is safe for M-matrices only.
Dropping is safe for M-matrices only.
No: For SPD matrix one can drop by retaining just principal submatrices
Needs transformation to keep the relevant parts
Some statements

- Dropping is safe for M-matrices only.
  No: For SPD matrix one can drop by retaining just principal submatrices
  Needs transformation to keep the relevant parts

- Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
Some statements

- Dropping is safe for M-matrices only.
  No: For SPD matrix one can drop by retaining just principal submatrices
  Needs transformation to keep the relevant parts

- Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
  No: for certain types of discretizations one can iterate in the divergence free space.
Some statements

- Dropping is safe for M-matrices only.
  No: For SPD matrix one can drop by retaining just principal submatrices
  Needs transformation to keep the relevant parts

- Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
  No: for certain types of discretizations one can iterate in the divergence free space.

- Grid independent convergence with an ILU factorization is not possible.
Some statements

- Dropping is safe for M-matrices only.
  No: For SPD matrix one can drop by retaining just principal submatrices
  Needs transformation to keep the relevant parts

- Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
  No: for certain types of discretizations one can iterate in the divergence free space.

- Grid independent convergence with an ILU factorization is not possible.
  No: for instance NGICCG (vdPloeg,Botta,W, 1997) shows grid independent convergence. Number of iterations controlled by dropping parameter.
Some statements

- Dropping is safe for M-matrices only.
  No: For SPD matrix one can drop by retaining just principal submatrices
  Needs transformation to keep the relevant parts

- Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
  No: for certain types of discretizations one can iterate in the divergence free space.

- Grid independent convergence with an ILU factorization is not possible.
  No: for instance NGICCG (vdPloeg,Botta,W, 1997) shows grid independent convergence. Number of iterations controlled by dropping parameter.

- A Navier-Stokes Jacobian can be approximated arbitrary close by an ILU factorization that will lead to an $O(N)$ algorithm.
Dropping is safe for M-matrices only.
No: For SPD matrix one can drop by retaining just principal submatrices
Needs transformation to keep the relevant parts

Indefiniteness in (Navier)-Stokes matrix is always a big problem for the solution process.
No: for certain types of discretizations one can iterate in the divergence free space.

Grid independent convergence with an ILU factorization is not possible.
No: for instance NGICCG (vdPloeg,Botta,W, 1997) shows grid independent convergence. Number of iterations controlled by dropping parameter.

A Navier-Stokes Jacobian can be approximated arbitrary close by an ILU factorization that will lead to an $O(N)$ algorithm.
Yes, regarding the above we expect this
Ingredients

- Fill reducing ordering
- Local Fourier-like transformation
  - improves diagonal dominance
  - to get rid of unwanted couplings
- Drop by retaining principal submatrices
- For incompressible Navier Stokes equation, do not drop in divergence and gradient part
  - There is no increase of fill in this part (even not in direct method) on Arakawa A, B, and C-grids
A cartoon of the new algorithm

Stokes on a structured C-grid
A cartoon of the new algorithm, step 1

Domain decomposition
A cartoon of the new algorithm, step 2

Identify separators
Elimination yields ‘geometric’ Schur-complement

NB: All horizontal (vertical) velocities on a vertical (horizontal) separator are coupled to the pressure inside.

What happens if we eliminate a velocity that is coupled to two pressure unknowns?
Flux representation (‘coarse grid’)

Transformation is such that amount of mass flowing through the separator remains the same.
Remaining $V$-nodes: separators are decoupled, $V$-nodes and coarse grid ($= V_\Sigma$) nodes decoupled
Robust at high Reynolds numbers

- Can compute highly unstable steady states;
- Moderate increase in number of iterations;
- Conv. tol $10^{-8}$ here.
Numerical results
All results obtained on Millipede cluster: 12 cores per node (opteron cores), 24GB per node

One linear solve in process (8 digits gain)

"Eff" indicates the deviation from optimal algorithmic ($O(N)$ behaviour) and parallel scalability.

<table>
<thead>
<tr>
<th>nx</th>
<th>sx</th>
<th>nl</th>
<th>np</th>
<th>setup</th>
<th>eff</th>
<th>solve</th>
<th>eff</th>
<th>iters</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>16.4</td>
<td>1.00</td>
<td>5.8</td>
<td>1.00</td>
<td>76</td>
</tr>
<tr>
<td>48</td>
<td>4</td>
<td>3</td>
<td>32</td>
<td>11.5</td>
<td>0.83</td>
<td>4.6</td>
<td>0.94</td>
<td>78</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>3</td>
<td>32</td>
<td>37</td>
<td>1.13</td>
<td>12.5</td>
<td>1.07</td>
<td>81</td>
</tr>
<tr>
<td>81</td>
<td>3</td>
<td>4</td>
<td>27</td>
<td>75</td>
<td>0.95</td>
<td>34</td>
<td>1.22</td>
<td>102</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>4</td>
<td>64</td>
<td>171</td>
<td>1.30</td>
<td>100</td>
<td>2.15</td>
<td>115</td>
</tr>
</tbody>
</table>

Number of levels influences computation time.
The amount of parallel work for solution part is relative low.
Eigenvalue results

Eigenvalues at Re=1900 and 2000 on $128^3$ grid.

<table>
<thead>
<tr>
<th>Re</th>
<th>$\lambda_{32}$</th>
<th>$\lambda_{64}$</th>
<th>$\lambda_{128}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1900</td>
<td>$-53.05 + 468i$</td>
<td>$-22.28 + 550i$</td>
<td>$-6.762 + 588i$</td>
</tr>
<tr>
<td>2000</td>
<td>$-42.35 + 463i$</td>
<td>$-10.73 + 545i$</td>
<td>$3.901 + 571i$</td>
</tr>
</tbody>
</table>

| $Re_c$ | 2395 | 2093 | 1963 |

Eigenvalues in thousands on sequence of grids.
Our conclusion: Based on second order extrapolation $Re_c$ is about 1930.
Rayleigh Benard problem

Matrix from stability study at Ra=2000 (singular at Ra=2600) 3D computation

<table>
<thead>
<tr>
<th>nx</th>
<th>sx</th>
<th>nl</th>
<th>np</th>
<th>nnz S</th>
<th>setup</th>
<th>eff</th>
<th>solve</th>
<th>eff</th>
<th>iters</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>3361</td>
<td>11</td>
<td>1.00</td>
<td>6</td>
<td>1.00</td>
<td>85</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>4</td>
<td>32</td>
<td>1</td>
<td>64</td>
<td>2.91</td>
<td>31</td>
<td>2.58</td>
<td>109</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>4</td>
<td>64</td>
<td>10969</td>
<td>250</td>
<td>2.84</td>
<td>109</td>
<td>2.27</td>
<td>130</td>
</tr>
</tbody>
</table>

To study scalability we performed 2D computations

<table>
<thead>
<tr>
<th>nx</th>
<th>sx</th>
<th>nl</th>
<th>np</th>
<th>nnz S</th>
<th>setup</th>
<th>eff</th>
<th>solve</th>
<th>eff</th>
<th>iters</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>4</td>
<td>3</td>
<td>16</td>
<td>646</td>
<td>0.318</td>
<td>1.00</td>
<td>0.20</td>
<td>1.00</td>
<td>65</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
<td>3</td>
<td>32</td>
<td>3046</td>
<td>0.723</td>
<td>1.14</td>
<td>0.475</td>
<td>1.19</td>
<td>72</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>3</td>
<td>32</td>
<td>13126</td>
<td>2.85</td>
<td>1.12</td>
<td>2.58</td>
<td>1.61</td>
<td>76</td>
</tr>
<tr>
<td>512</td>
<td>4</td>
<td>3</td>
<td>32</td>
<td>54406</td>
<td>38.4</td>
<td>3.77</td>
<td>12.3</td>
<td>1.92</td>
<td>77</td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
<td>3</td>
<td>128</td>
<td>221446</td>
<td>728</td>
<td>71.5</td>
<td>54.4</td>
<td>8.50</td>
<td>80</td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
<td>4</td>
<td>128</td>
<td>13126</td>
<td>40</td>
<td>3.9</td>
<td>33</td>
<td>5.15</td>
<td>145</td>
</tr>
<tr>
<td>2048</td>
<td>4</td>
<td>4</td>
<td>128</td>
<td>54406</td>
<td>56</td>
<td>1.37</td>
<td>104</td>
<td>4.06</td>
<td>146</td>
</tr>
</tbody>
</table>
3D Laplace, gain 10 digits

<table>
<thead>
<tr>
<th>nx</th>
<th>sx</th>
<th>nl</th>
<th>np</th>
<th>nnz</th>
<th>S</th>
<th>setup</th>
<th>eff</th>
<th>solve</th>
<th>eff</th>
<th>iters</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>631</td>
<td>11</td>
<td>1.00</td>
<td>16.1</td>
<td>1.00</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>5</td>
<td>128</td>
<td>1</td>
<td>16</td>
<td>1.45</td>
<td>32</td>
<td>1.98</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>4</td>
<td>5</td>
<td>512</td>
<td>631</td>
<td>52</td>
<td>2.36</td>
<td>177</td>
<td>5.50</td>
<td>87</td>
<td>0.62TB</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
<td>5</td>
<td>2048</td>
<td>7839</td>
<td>535</td>
<td>12.2</td>
<td>777</td>
<td>12.1</td>
<td>105</td>
<td>4.2 TB</td>
<td></td>
</tr>
</tbody>
</table>

**ML**

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>64</td>
<td>23</td>
<td>1.00</td>
<td>48.9</td>
<td>1.00</td>
<td>11</td>
<td>71GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>128</td>
<td>586</td>
<td>6.37</td>
<td>211</td>
<td>1.08</td>
<td>10</td>
<td>291GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Swapping occurs in $1024^3$ case
There is room for improvement, 10 times slower than ML
Outlook and Conclusions
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.

ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.

HYMLS makes it possible to do real steady state computations. Considerable effort to develop HYMLS, but it has high potential.

Easy extension with more physics, temperature, salt etc., avoids inner iterations.

Can be used as approximate Jacobian in Jacobi-Davidson for eigenvalue problems.

It benefits directly from improvements in Epetra.

It gives control over the communication between processors.

Grid independent convergence is possible with ILU factorization.

Indefiniteness in (Navier)-Stokes matrix for standard A,B,C-grid can be treated exactly; method is provably robust for Stokes for every size of subdomain. Never had stagnation in Navier-Stokes.
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.

- ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.
- ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.
- HYMLS makes it possible to do real steady state computations.
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.
- ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.
- HYMLS makes it possible to do real steady state computations.
- Considerable effort to develop HYMLS, but is has high potential.
  - Easy extension with more physics, temperature, salt etc., avoids inner iterations.
  - Can be used as approximate Jacobian in Jac. Davidson for eigenvalue problems.
  - It benefits directly from improvements in Epetra.
  - It gives control over the communication between processors.
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.
- ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.
- HYMLS makes it possible to do real steady state computations.
- Considerable effort to develop HYMLS, but it has high potential.
  - Easy extension with more physics, temperature, salt etc., avoids inner iterations.
  - Can be used as approximate Jacobian in Jac. Davidson for eigenvalue problems.
  - It benefits directly from improvements in Epetra
  - It gives control over the communication between processors
- Grid independent convergence is possible with ILU factorization.
Concluding remarks

- Trilinos is a very pleasant framework to work with. Epetra will be extended to Tepetra. This allows also complex arithmetic. Handy for eigenvalue computations.
- ANASAZI works but dominates the computation time. Reuse of already computed basis and better targeting (using complex arithmetic). Preference for Jacobi-Davidson solver.
- HYMLS makes it possible to do real steady state computations.
- Considerable effort to develop HYMLS, but it has high potential.
  - Easy extension with more physics, temperature, salt etc., avoids inner iterations.
  - Can be used as approximate Jacobian in Jac. Davidson for eigenvalue problems.
  - It benefits directly from improvements in Epetra
  - It gives control over the communication between processors
- Grid independent convergence is possible with ILU factorization.
- Indefiniteness in (Navier)-Stokes matrix for standard A,B,C-grid can be treated exactly; method is provably robust for Stokes for every size of subdomain. Never had stagnation in Navier-Stokes.
References


