EFFICIENCY IMPROVEMENT FOR PANEL CODES
LITERATURE REVIEW

Elisa Ang
30 Jan 2015, Friday
Project Overview

- Panel codes (or Boundary Element Method) are used for flow computations in MARIN
- Boundary Element Method generates dense linear system of equation
- The project aims to speed up the computation time required
Presentation Topics

- Review of what had been done
- Boundary Element Method
- Solver Methods comparison:
  - GMRES vs IDR(s)
- Preconditioning:
  - Block Jacobi vs Deflation
- Fast Multipole Method
- Subsequent plan
Current Strategy in MARIN:

Direct Solver or GMRES with incomplete LU preconditioner

<table>
<thead>
<tr>
<th>Matrix Name</th>
<th>Size</th>
<th>Real/Complex</th>
<th>Strategy</th>
<th>Solve time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steadycav</td>
<td>4620</td>
<td>Real</td>
<td>Direct</td>
<td>3.3 s (direct)</td>
</tr>
<tr>
<td>FATIMA_7894</td>
<td>7894</td>
<td>Complex</td>
<td>ILU</td>
<td>13.0 s</td>
</tr>
<tr>
<td>FATIMA_20493</td>
<td>20493</td>
<td>Complex</td>
<td>ILU</td>
<td>170.0 s</td>
</tr>
</tbody>
</table>
Review of what had been done

- Project was undertaken by Martijn de Jong in 2012
- The solution was GMRES with Block-Jacobi Preconditioner with OpenMP
- GPU was used to speed up the solver

<table>
<thead>
<tr>
<th>Matrix Name</th>
<th>Size</th>
<th>Real/Complex</th>
<th>Solve time (Old)</th>
<th>Solve time (Now)</th>
<th>Solve time (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steadycav</td>
<td>4620</td>
<td>Real</td>
<td>3.3 s (Direct)</td>
<td>0.5s</td>
<td>Not tested</td>
</tr>
<tr>
<td>FATIMA_7894</td>
<td>7894</td>
<td>Complex</td>
<td>13.0s (iLU)</td>
<td>4.5s</td>
<td>3.1s (Direct)</td>
</tr>
<tr>
<td>FATIMA_20493</td>
<td>20493</td>
<td>Complex</td>
<td>170.0 s (iLU)</td>
<td>34.3s</td>
<td>22.7s (Block-Jac)</td>
</tr>
</tbody>
</table>

All results and diagrams extracted from:
de Jong, M. 2012. Efficient Solvers For Panel Codes
6 Boundary Element Method

Kythe, K.P. 1995. An Introduction to Boundary Element Methods
Boundary Element Method (BEM)

- Numerical method to solve boundary value problems

\[-\nabla^2 u = 0\]

- FEM vs BEM
  - FEM solves this by discretizing entire domain, BEM only discretizes boundary
  - FEM results in sparse matrix, BEM, dense
Boundary Element Method (BEM)

**Equation**

- \(-\nabla^2 u = 0\)
  
  \(u = u_0 \text{ on } S_1\)
  
  \(q = \frac{\partial u}{\partial n} = q_0 \text{ on } S_2\)

- Weak Formulation

  \[-\iiint_V u^* \nabla^2 u \, dV = \iiint_V \nabla u^* \nabla u \, dV - \int_S u^* \nabla u \cdot \hat{n} \, ds = 0\]

  \[-\iiint_V u \nabla^2 u^* \, dV + \int_S u \nabla u^* \cdot \hat{n} \, ds - \int_S u^* \nabla u \cdot \hat{n} \, ds = 0\]

- \(u^* = \text{Fundamental Solution}\)

  \(\nabla^2 u^* = -\delta(x_i)\)

  \(u^* = \frac{1}{4\pi(x - x_i)}\)
Thus we have the Boundary Integral Equation (BIE)

\[ \nabla^2 u = 0 \]

\[ u = u_0 \text{ on } S_1 \]

\[ q = \frac{\partial u}{\partial n} = q_0 \text{ on } S_2 \]

\[ c(x_i)u(x_i) + \iint_S uq_i^* \, dS = \iint_S u^*_i q \, dS, \quad S = S_1 \cup S_2 \]

\[ c(x_i) = \begin{cases} 
1 & \text{if } x_i \text{ is inside } R \\
1 & \text{if } x_i \text{ is on a smooth portion of } S \\
\frac{1}{2} & \text{if } x_i \text{ is inside } R \\
\end{cases} \]
Boundary Element Method (BEM)

Discretization of BIE gives:

\[ c(x_i)u(x_i) + \int_S u_i^* dS = \int_S u_i^* q S, \]
Boundary Element Method (BEM)

- Discretization of BIE gives:

\[ c(x_i)u(x_i) + \sum_{j=1}^{N} u(x_j) \tilde{H}_{ij} = \sum_{j=1}^{N} q(x_j) G_{ij} \]

\[ \tilde{H}_{ij} = \int_{S_j} q^*_i \, dS \quad q^* = -\frac{1}{4\pi(x-x_i)^2} \]

\[ G_{ij} = \int_{S_j} u^*_i \, dS \quad u^* = \frac{1}{4\pi(x-x_i)} \]

- Rearranging -> Dense linear system \( Ax = b \)
**Boundary Element Method**

- **BEM test matrices from MARIN**

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Real/Complex</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steadycav1</td>
<td>4620</td>
<td>Real</td>
<td>PROCAL</td>
</tr>
<tr>
<td>Steadycav2</td>
<td>4620</td>
<td>Real</td>
<td>PROCAL</td>
</tr>
<tr>
<td>Steadycav3</td>
<td>4620</td>
<td>Real</td>
<td>PROCAL</td>
</tr>
<tr>
<td>Steadycav4</td>
<td>4649</td>
<td>Real</td>
<td>PROCAL</td>
</tr>
<tr>
<td>DIFFRAC_1828</td>
<td>1828</td>
<td>Complex</td>
<td>DIFFRAC</td>
</tr>
<tr>
<td>FATIMA_7894</td>
<td>7894</td>
<td>Complex</td>
<td>FATIMA</td>
</tr>
<tr>
<td>FATIMA_20493</td>
<td>20493</td>
<td>Complex</td>
<td>FATIMA</td>
</tr>
</tbody>
</table>
Solver

GMRES vs IDR(s)

What does GMRES stand for?

Generalized Minimum Residual Method

www.allacronyms.com

http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html
Solver Method

- Advantage of GMRES:
  - Optimality
  - Matrix vector multiplication required per iteration

- Advantage of IDR(s):
  - Short recurrence
  - Less matrix vector multiplication required as compared to bi-CG
Solver Method

- **GMRES**
  - Search for solution within increasing Krylov Subspace

- **IDR(s)**
  - Concept of nested subspace

\[ G_j \subset G_{j-1} \quad \text{with} \quad G_0 = \mathcal{K}(A, r_0) \]

\[ G_j = (I - \omega_j A)(G_{j-1} \cap S) \]

\[ G_j = \{0\}, \text{for some } j \leq N \]

Scalar value

left null space of some N x s matrix P
Solver Method

- **Numerical Results**

<table>
<thead>
<tr>
<th>Timing (s)</th>
<th>Matrix</th>
<th>GMRES</th>
<th>IDR(10)</th>
<th>Bi-CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steadycav1 N=4620</td>
<td>5.4</td>
<td>5.9</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Matrix</th>
<th>GMRES</th>
<th>IDR(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Steadycav1 N=4620</td>
<td>237</td>
<td>379</td>
</tr>
<tr>
<td></td>
<td></td>
<td>490</td>
<td>490</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timing for solving (s)</th>
<th>Matrix</th>
<th>GMRES with block jacobi preconditioned matrix</th>
<th>IDR(10) with block jacobi preconditioned matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATIMA_20493</td>
<td>1948</td>
<td>894.3</td>
<td></td>
</tr>
</tbody>
</table>

| Iteration | | |
|-----------| | |
|           | 96 | 115 |
Preconditioner
Preconditioner

- Martijn did a thorough comparison between ILU and Block Jacobi
- Can deflation further reduce the iterations required for convergence?
Preconditioner

- Short review of Deflation
  - Split solution space into 2 complementary subspaces through projection
    \[ x = (I - Q_D)x + Q_D x \]
  - Define the projectors
    \[ P_D = I - AZE^{-1}Y^T \]
    \[ Q_D = I - ZE^{-1}Y^T A \]
Consider

- \( Ax = (I - P_D)Ax + P_D Ax \)
- \( P_D Ax = P_D b \)
- \( x = ZE^{-1}Y^T b + Q_D \tilde{x} \)
Preconditioner

- Choice of deflation subspace $Z$ and $Y$
  - Space spanned by eigenvectors of $A$ corresponding to smallest eigenvalues
  - Effect is to shift the small eigenvalues to 0 while leaving the other eigenvalues unchanged

$$x = (I - Q_D)x + Q_Dx$$

$$P_D = I - AZE^{-1}Y^T$$

$$Q_D = I - ZE^{-1}Y^TA$$

$$P_DAx = P_Db$$
Preconditioner

- Choice of deflation subspace $Z$ and $Y$

- Subdomain decomposition

\[
x = (I - Q_D)x + Q_D x
\]
\[
P_D = I - AZE^{-1}YT
\]
\[
Q_D = I - ZE^{-1}YT A
\]
\[
P_D A x = P_D b
\]
## Numerical Results

<table>
<thead>
<tr>
<th>Matrix</th>
<th>IDR(10) with block jacobi</th>
<th>IDR(10) with subdomain deflation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SteadyCav1 N=4620</td>
<td>0.65</td>
<td>0.69 BJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.13 D</td>
</tr>
<tr>
<td>SteadyCav1 N=4620</td>
<td>0.49</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>1.13</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>41</td>
</tr>
<tr>
<td>FATIMA 7894 N=7894</td>
<td>7.7</td>
<td>8.0 BJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.1 D</td>
</tr>
<tr>
<td>FATIMA 7894 N=7894</td>
<td>10.1</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>17.8</td>
<td>19.3</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>86</td>
</tr>
</tbody>
</table>

No Improvement
Further analysis of why deflation do not provide any improvement
Further analysis of why deflation do not provide any improvement
## Numerical Results

<table>
<thead>
<tr>
<th></th>
<th>Matrix</th>
<th>IDR(10) with block jacobi</th>
<th>IDR(10) with subdomain deflation</th>
<th>IDR(10) with eigenvectors deflation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing for</td>
<td>Steadycav1</td>
<td>0.65</td>
<td>0.69 BJ</td>
<td>0.69 BJ</td>
</tr>
<tr>
<td>preconditioning (s)</td>
<td>N=4620</td>
<td></td>
<td>0.13 D</td>
<td>2.25 D</td>
</tr>
<tr>
<td>Timing for</td>
<td>FATIMA 7894</td>
<td>7.7</td>
<td>8.0 BJ</td>
<td>8.08 BJ</td>
</tr>
<tr>
<td>solving (s)</td>
<td>N=7894</td>
<td></td>
<td>1.1 D</td>
<td>52.5 D</td>
</tr>
<tr>
<td>Total time (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steadycav1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N=4620</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing for</td>
<td></td>
<td>0.49</td>
<td>0.44</td>
<td>0.29</td>
</tr>
<tr>
<td>preconditioning (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing for</td>
<td></td>
<td>1.13</td>
<td>1.26</td>
<td>3.23</td>
</tr>
<tr>
<td>solving (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total time (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FATIMA 7894</td>
<td></td>
<td>10.1</td>
<td>10.2</td>
<td>7.8</td>
</tr>
<tr>
<td>N=7894</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing for</td>
<td></td>
<td>17.8</td>
<td>19.3</td>
<td>68.3</td>
</tr>
<tr>
<td>preconditioning (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing for</td>
<td></td>
<td>84</td>
<td>86</td>
<td>66</td>
</tr>
<tr>
<td>solving (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total time (s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But there’s a need to improve the time to construct the deflation space.

Reduction!
Fast Multipole Method (FMM)

Leslie Greengard  Vladimir Rokhlin
New York University  Yale University

1996
Fast Multipole Method (FMM)

- Speed up matrix-vector multiplication from $O(N^2)$ to $O(N \log N)$ or $O(N)$
- Exploit the hierarchical structure of the matrix
- What is the hierarchical structure?
Hierarchical Structure

First, this is an example of the hierarchical splitting of a matrix $A$

\[ A = \sum_{l=2}^{n} (M_l + N_l) \]

**Represents a admissible block:**
An admissible block has low rank

**Represents previously admissible block**
Hierarchical Structure

- All test matrices have the same hierarchical structure

Steadycav1

Steadycav2
Fast Multipole Method (FMM)
Fast Multipole Method (FMM)

- Why is the blocks of low rank?
- How can we use the fact that the blocks are low rank?
Recall that each element of our matrix $A$ is:

$$a_{ij} = \int_{S_j} q^*(x_i, x_j) dS \text{ or } \int_{S_j} u^*(x_i, x_j) dS$$

$$q^* = -\frac{1}{4\pi(x-x_i)^2}$$

$$u^* = \frac{1}{4\pi(x-x_i)}$$
Let $K(x, y) = a(x_i, x_j)$

And apply Taylor Expansion, centred around $(c_\sigma, c_\tau)$

$$K(x, y) = \sum_{l=0}^{p-1} \frac{1}{l!} [(x - c_\sigma) \partial_x + (y - c_\tau) \partial_y]^l K(c_\sigma - c_\tau) + R_p(x, y)$$

$$\approx 0 \text{ if } \left| \frac{(x - c_\sigma) + (y - c_\tau)}{|x - y|} \right| < 1$$
Applying binomial expansion and simplifying:

\[ K(x, y) = \sum_{m=0}^{l+m} \sum_{l=-m}^{p-1-m} \frac{1}{m!l!} \partial_x^l \partial_y^m K(c_\sigma - c_\tau) (x - c_\sigma)^l (y - c_\tau)^m \]

Fast Multipole Method (FMM)

All elements within each block has the same \((c_\sigma, c_\tau)\)
Fast Multipole Method (FMM)

- Applying binomial expansion and simplifying:

\[
K(x, y) = \sum_{m=0}^{l+m} \sum_{l=-m}^{p-1-m} \frac{1}{m! l!} \partial_x^l \partial_y^m K(c_\sigma - c_\tau)(x - c_\sigma)^l(y - c_\tau)^m
\]

Thus for each block, we can define upper triangular matrix \( S^{\sigma,\tau} \in \mathbb{C}^{p \times p} \)

\[
s_{l,m} = \begin{cases} 
\frac{1}{l! m!} \partial_x^l \partial_y^m K(c_\sigma, c_\tau) & \text{if } 0 \leq l + m \leq p - 1 \\
0 & \text{else}
\end{cases}
\]
Applying binomial expansion and simplifying:

\[
K(x, y) = \sum_{m=0}^{l+m} \sum_{l=-m}^{p-1-m} \frac{1}{m! l!} \partial_x^l \partial_y^m K(c_\sigma - c_\tau)(x - c_\sigma)^l(y - c_\tau)^m
\]

Then we define 2 matrices, \( \Psi^\tau \in \mathbb{C}^{b_y \times p} \), \( \Psi^\sigma \in \mathbb{C}^{b_x \times p} \)

\[
\psi^\sigma_{ixl} = (x - c_\sigma)^l, \quad x = X(i), \quad \psi^\tau_{jxm} = (y - c_\tau)^m, \quad y = X(j),
\]

where \( \frac{\sigma N}{2^l} \leq i \leq \frac{\sigma N}{2^l} + b \)

and \( \frac{\sigma N}{2^l} \leq j \leq \frac{\sigma N}{2^l} + b \)
Fast Multipole Method (FMM)

Each admissible block can be written as:

\[ M_{\sigma,\tau}(l) \approx \tilde{M}_{\sigma,\tau}(l) = (\Psi^\sigma)S^\sigma,\tau(\Psi^\tau)^T \]

Each block in the same row has the same \( \Psi^\sigma \), and in the same col with the same \( \Psi^\tau \).

\[ M_l \approx \sum \tilde{M}_{\sigma,\tau}(l) \]

\[ = \text{blockdiag}(\Psi^\sigma)_{\sigma=0,1,\ldots,2^l}S(l)\text{blockdiag}(\Psi^\tau)^T_{\tau=0,1,\ldots,2^l} \]

\[ \begin{align*}
N \times p2^l & \quad \text{and} \quad p2^l \times p2^l \quad \text{and} \quad N \times p2^l
\end{align*} \]
Fast Multipole Method (FMM)

Consider now the matrix vector multiplication $Ax$

$$Ax \approx \sum_{l=2}^{n} \tilde{M}_l x + N_n x$$

$$M_l x = \text{blockdiag}(\Psi^{\sigma})_{\sigma=0,1,\ldots,2^l} S(l) \text{blockdiag}(\Psi^{\tau})^T_{\tau=0,1,\ldots,2^l} x$$

$$N \times p^{2^l}$$

$$p^{2^l} \times p^{2^l}$$

$$O(N)$$
Consider now the matrix vector multiplication $Ax$

$$Ax \approx \sum_{l=2}^{n} \tilde{M}_l x + N_n x$$

It can be shown that $N_n$ is a sparse matrix with at most $c \times N$ non-zero elements

$O(N)$
Consider now the matrix vector multiplication $Ax$

$$Ax \approx \sum_{l=2}^{n} \tilde{M}_l x + N_n x$$

$Ax$ is now a $O(N)$ operation
Other ways to obtain low rank approximation

\[ M_{\sigma,\tau}(l) \approx \tilde{M}_{\sigma,\tau}(l) = (\Psi^\sigma) S^{\sigma,\tau} (\Psi^{\tau})^T \]

Without domain & kernel information, we can use lanzcos bidiagonalization to check for admissibility and obtain low rank approximation

\[ M_{\sigma,\tau}(l) \approx \tilde{M}_{\sigma,\tau}(l) = U_{\sigma,\tau} B_{\sigma,\tau} V_{\sigma,\tau}^H \]
But we can’t form block diag matrix for $M(l)$

Matrix vector multiplication has to be done like this:

\[
\tilde{M}_l x = \begin{bmatrix}
(\tilde{M}_l x)_1 \\
\vdots \\
(\tilde{M}_l x)_{2^l}
\end{bmatrix} = \begin{bmatrix}
\sum_{\tau=1}^{2^l} \tilde{U}_{1,\tau} B_{1,\tau} \tilde{V}_{1,\tau}^H x_{\tau} \\
\vdots \\
\sum_{\tau=1}^{2^l} \tilde{U}_{2^l,\tau} B_{2^l,\tau} \tilde{V}_{2^l,\tau}^H x_{\tau}
\end{bmatrix}
\]

$O(N \log N)$
## Numerical Results

<table>
<thead>
<tr>
<th>b</th>
<th>p</th>
<th>$t_{m\text{-full}}$</th>
<th>$t_{m\text{-hie}}$</th>
<th>% reduction</th>
<th>$t_{split}$</th>
<th>$t_{m\text{-full}}$</th>
<th>$t_{m\text{-hie}}$</th>
<th>% reduction</th>
<th>$t_{split}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>10</td>
<td>0.043</td>
<td>0.034</td>
<td>20.93%</td>
<td>1.35</td>
<td>0.3</td>
<td>0.14</td>
<td>53.33%</td>
<td>11.03</td>
</tr>
<tr>
<td>20</td>
<td>0.028</td>
<td>34.88%</td>
<td>2.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0.024</td>
<td>44.19%</td>
<td>3.79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.024</td>
<td>44.19%</td>
<td>5.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>0.034</td>
<td>0.032</td>
<td>5.88%</td>
<td>0.91</td>
<td>0.15</td>
<td>40.00%</td>
<td>9.4</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0286</td>
<td>15.88%</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0.02</td>
<td>41.18%</td>
<td>2.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.018</td>
<td>47.06%</td>
<td>3.32</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fast Multipole Method (FMM)

**Numerical Results – Storage Requirement**

<table>
<thead>
<tr>
<th></th>
<th>Steadycav1</th>
<th>Steadycav2</th>
<th>Steadycav3</th>
<th>Steadycav4</th>
<th>FATIMA_7894</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage requirement</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for full matrix</td>
<td>0.17 GB</td>
<td>0.17 GB</td>
<td>0.17 GB</td>
<td>0.17 GB</td>
<td>0.99 GB</td>
</tr>
<tr>
<td><strong>Storage requirement</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in hierarchical form (b=100, p=50)</td>
<td>N: 0.086 GB</td>
<td>N: 0.085 GB</td>
<td>N: 0.084 GB</td>
<td>N: 0.08 GB</td>
<td>N: 0.56 GB</td>
</tr>
<tr>
<td></td>
<td>B: 3.71e-4 GB</td>
<td>B: 3.74e-4 GB</td>
<td>B: 3.81e-4 GB</td>
<td>B: 3.82e-4 GB</td>
<td>B: 5.28e-4 GB</td>
</tr>
<tr>
<td></td>
<td>U: 0.038 GB</td>
<td>U: 0.038 GB</td>
<td>U: 0.038 GB</td>
<td>U: 0.04 GB</td>
<td>U: 0.078 GB</td>
</tr>
<tr>
<td></td>
<td>V: 0.037 GB</td>
<td>V: 0.037 GB</td>
<td>V: 0.038 GB</td>
<td>V: 0.039 GB</td>
<td>V: 0.077 GB</td>
</tr>
<tr>
<td></td>
<td><strong>Total: 0.16 GB</strong></td>
<td><strong>Total: 0.16 GB</strong></td>
<td><strong>Total: 0.16 GB</strong></td>
<td><strong>Total: 0.16 GB</strong></td>
<td><strong>Total: 0.72 GB</strong></td>
</tr>
<tr>
<td><strong>Decrease in storage required</strong></td>
<td>5.88%</td>
<td>5.88%</td>
<td>5.88%</td>
<td>5.88%</td>
<td>27.3%</td>
</tr>
</tbody>
</table>
"We're under a lot of time-pressure here, so we'll need to jump to conclusions."
Conclusion & Subsequent Plan

- Implement IDR(s) in place of GMRES
- Explore efficient implementation of deflation with eigenvectors
- Explore use of GPU
- Fast Multipole Method with Lanczos bidiagonalization
- Fast Multipole Method with domain and kernel information
Thank you!