The Augmented Block-Cimmino Distributed Method

Improvements to the Conjugate Gradient accelerated Block Cimmino Method

Mohamed Zenadi¹
Iain S. Duff²  Ronan Guivarch¹  Daniel Ruiz¹

IRIT - ENSEEIHT¹
CERFACS and Rutherford Appleton Laboratory²

September 13, 2013
Block Cimmino: The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]

Partitioning the system $Ax = b$

\[
\begin{pmatrix}
A_1 \\
A_2 \\
\vdots \\
A_p
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_p
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_p
\end{pmatrix}
\]

Partitions can be obtained by cutting uniformly the matrix (with no permutation) or using a any partitioner
Block Cimmino: The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]

The Block Cimmino Iteration

\[ \delta_i^{(k)} = A_i^+ b_i - P_{R(A_i^T)} x^{(k)} \]

\[ x^{(k+1)} = x^{(k)} + \nu \sum_{i=1}^{p} \delta_i^{(k)} \]

where:

\[ A_i^+ = A_i^T (A_i A_i^T)^{-1} \]

and

\[ P_{R(A_i^T)} = A_i^+ A_i \]
Block Cimmino: The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]

Acceleration
Apply CG to solve the SPD system

$$\sum_{i=1}^{p} A_i^+ A_i \mathbf{x} = \sum_{i=1}^{p} A_i^+ b_i$$
Block Cimmino : The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]
- CG acceleration: iteration matrix is
  \[ \sum_{i=1}^{P} A_i^+ A_i = \sum_{i=1}^{P} P_{\mathcal{R}(A_i^T)} \]

Projections: \( \delta_i^{(k)} = A_i^+ b_i - P_{\mathcal{R}(A_i^T)} x^{(k)} \)
Block Cimmino : The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]
- CG acceleration: iteration matrix is 
  \[
  \sum_{i=1}^{P} A_i^+ A_i = \sum_{i=1}^{P} P_{\mathcal{R}(A_i^T)}
  \]

Projections: 
\[
\delta_i^{(k)} = A_i^+ b_i - P_{\mathcal{R}(A_i^T)} x^{(k)}
\]

Solve independently using a direct solver the systems for each partition
\[
\begin{bmatrix}
  I & A_i^T \\
  A_i & 0 \\
\end{bmatrix}
\begin{bmatrix}
  u_i \\
  v_i \\
\end{bmatrix} = \begin{bmatrix}
  0 \\
  b_i - A_ix \\
\end{bmatrix}
\]

where: 
\[
u_i = A_i^+(b_i - A_ix) = \delta_i
\]
Block Cimmino: The basic facts

- Block row projection method [Elfving (Numer. Math. 1980)]

- CG acceleration: iteration matrix is
  \[ \sum_{i=1}^{P} A_i^{+} A_i = \sum_{i=1}^{P} P_{\mathcal{R}(A_i^T)} \]

- Can also exploit 2nd and 3rd levels of parallelism (sparsity structure, BLAS3 Kernels)

**Projections:**
\[ \delta_i^{(k)} = A_i^{+} b_i - P_{\mathcal{R}(A_i^T)} x^{(k)} \]

Solve independently using a direct solver the systems for each partition

\[
\begin{bmatrix}
  I & A_i^T \\
  A_i & 0
\end{bmatrix}
\begin{bmatrix}
  u_i \\
  v_i
\end{bmatrix}
= \begin{bmatrix}
  0 \\
  b_i - A_i x
\end{bmatrix}
\]

where: \( u_i = A_i^{+} (b_i - A_i x) = \delta_i \)
The Hybrid Idea: Computing projections

- Computes all the projections at once
- Build a block diagonal system of augmented system
- Analyse + Factorize then solve using a direct solver
- Exploits the Forest structure (if possible)
Distributed Scheme

- Distributed Conjugate Gradient Acceleration
- Multiple levels of parallelism
- Forest simulation

Important Notice
There is only a single Block-CG distributed over these processes.
Parallelism results: Distributed B-CG

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Nonzeros</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$: torso3</td>
<td>259,156</td>
<td>4,429,042</td>
<td>3D model of torso</td>
</tr>
<tr>
<td>$N_2$: CoupCons3D</td>
<td>416,800</td>
<td>17,277,420</td>
<td>structural problem</td>
</tr>
<tr>
<td>$N_3$: cage13</td>
<td>445,315</td>
<td>7,479,343</td>
<td>DNA electrophoresis</td>
</tr>
<tr>
<td>$N_4$: Hamrle3</td>
<td>1,447,360</td>
<td>5,514,242</td>
<td>Circuit Simulation</td>
</tr>
</tbody>
</table>

- Runs made on Hyperion - CICT
- 3.2Ghz Intel Xeon Quad-Core CPUs (2 per node)
- 36GB per node
Parallelism results: Distributed B-CG

(a) Factorization speedup

<table>
<thead>
<tr>
<th>Problem</th>
<th>Partitions</th>
<th>Factorization at 8 Cores</th>
<th>B-CG at 8 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>16</td>
<td>12.11 s.</td>
<td>6.34 s.</td>
</tr>
<tr>
<td>$N_2$</td>
<td>32</td>
<td>14.41 s.</td>
<td>63.49 s.</td>
</tr>
<tr>
<td>$N_3$</td>
<td>256</td>
<td>28.15 s.</td>
<td>13.64 s.</td>
</tr>
<tr>
<td>$N_4$</td>
<td>64</td>
<td>6.67 s.</td>
<td>773 s.</td>
</tr>
</tbody>
</table>
Block Cimmino vs MUMPS

32 cores shared memory machine

<table>
<thead>
<tr>
<th>Problem</th>
<th>[MUMPS] Factorization</th>
<th>[BC] Factorization</th>
<th>B-CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>torso3</td>
<td>5.03 s.</td>
<td>3.21 s.</td>
<td>5.18 s.</td>
</tr>
<tr>
<td>Cage13</td>
<td>1452.44 s.</td>
<td>9.31 s.</td>
<td>3.18 s.</td>
</tr>
<tr>
<td>Hamrle3</td>
<td>413.21 s.</td>
<td>2.13 s.</td>
<td>282.90 s.</td>
</tr>
</tbody>
</table>

The reality: On most test cases, MUMPS did better

However:

- BC breaks the complexity down so that the factorization goes faster
- BC consumes less memory:
  - MUMPS + Cage13: MAX: 6.7GB / AVG: 4.0GB
A path to orthogonality

Issues with Block-Cimmino:

- Convergence is problem dependent
- Unpredictable convergence behaviour (usually plateaux based)
- Multiple solves requires a re-run of B-CG (too expensive)
A path to orthogonality

**Issues with Block-Cimmino:**

- Convergence is problem dependent
- Unpredictable convergence behaviour (usually plateaux based)
- Multiple solves requires a re-run of B-CG (too expensive)

**Proposed solution:**

- Enforce numerical orthogonality between partitions by adding extra variables and constraints
- Extract a condensed smaller subsystem (similar to Schur complement techniques) that can be reused for efficient further solves
A path to orthogonality

Issues with Block-Cimmino:

▶ Convergence is problem dependent
▶ Unpredictable convergence behaviour (usually plateaux based)
▶ Multiple solves requires a re-run of B-CG (too expensive)

Proposed solution:

▶ Enforce numerical orthogonality between partitions by adding extra variables and constraints
▶ Extract a condensed smaller subsystem (similar to Schur complement techniques) that can be reused for efficient further solves

⇒ Augmented Block Cimmino Distributed solver (ABCD solver)
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)

Illustrative example
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions \((j > i)\), expand with \(C_{i,j} = A_i A_j^T\),
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions \((j > i)\), expand with \(C_{i,j} = A_i A_j^T\), and enforce numerical orthogonality

Illustrative example
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions \((j > i)\), expand with \(C_{i,j} = A_i A_j^T\), and enforce numerical orthogonality to obtain \(\bar{A} = [A \ C]\)

Illustrative example
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions \( (j > i) \), expand with \( C_{i,j} = A_i A_j^T \), and enforce numerical orthogonality to obtain \( \bar{A} = [A \ C] \)
- Add extra constraints to build an equivalent linear system:

\[
\begin{bmatrix}
A & C \\
0 & I
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
b \\
0
\end{bmatrix},
\]

where \( y = 0 \) ensures the same solution \( x \).
The augmentation process

- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions \((j > i)\), expand with \(C_{i,j} = A_i A_j^T\), and enforce numerical orthogonality to obtain \(\bar{A} = [A \quad C]\)
- Add extra constraints to build an equivalent linear system:

\[
\begin{bmatrix}
A & C \\
0 & I
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
=
\begin{bmatrix}
b \\
0
\end{bmatrix},
\]

where \(y = 0\) ensures the same solution \(x\).

Problem
the extra partition \(Y = [0 \quad I]\), linked to the constraints equations, is not orthogonal to the previous partitions in \(\bar{A} = [A \quad C]\).
The augmentation process

To enforce this orthogonality, we project the column vectors $Y^T$ onto the null space of $\tilde{A} = [A \ C]$ (orthogonal complement of $\mathcal{R}(\tilde{A}^T)$):

$$W^T = (I - P) Y^T,$$

where (as a result of the enforced orthogonality):

$$P = P_{\mathcal{R}(\tilde{A}^T)} = P_{\bigoplus_{i=1}^{p} \mathcal{R}(\tilde{A}_i^T)} = \sum_{i=1}^{p} P_{\mathcal{R}(\tilde{A}_i^T)}$$

We finally obtain $[A \ C \ B \ S]$, where $[B \ S] = W$, an augmented matrix with mutually numerically orthogonal partitions.
The augmentation process

Illustrative example
The augmentation process

To keep the consistency within the solution of the new system:

\[
\begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix}
\]

we compute the right hand side \( f \) as:

\[
f = \begin{bmatrix} B & S \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = Y(I - P) \begin{bmatrix} x \\ 0 \end{bmatrix}
\]

\[= -YP \begin{bmatrix} x \\ 0 \end{bmatrix} \quad \text{(since } Y = \begin{bmatrix} 0 & I \end{bmatrix} \text{)} \]

\[= -Y \tilde{A}^+ \tilde{A} \begin{bmatrix} x \\ 0 \end{bmatrix} \]

\[f = -Y \tilde{A}^+ b\]
Implicit Direct Solver

Since all the partitions in the new equivalent linear system

\[
\begin{bmatrix}
A & C
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
= 
\begin{bmatrix}
b \\
f
\end{bmatrix}
\]

are mutually numerically orthogonal, the Cimmino iteration matrix becomes the Identity matrix, and the solution can be directly obtained as :

\[
\begin{bmatrix}
x \\
y
\end{bmatrix}
= \tilde{A}^+ b + W^+ f
\]

\[
= \tilde{A}^+ b - W^+ Y \tilde{A}^+ b
\]

\[
= \sum_{i=1}^{p} \tilde{A}_i^+ b_i - W^+ Y \sum_{i=1}^{p} \tilde{A}_i^+ b_i
\]
Computational Ingredients

Knowing that $W = [B \ S] = Y (I - P)$, with $Y = [0 \ 1]$, we have:

\[
WW^T = Y (I - P) (I - P)^T Y^T
\]
\[
= Y (I - P)^2 Y^T
\]
\[
= Y (I - P) Y^T
\]
\[
= [B \ S] Y^T
\]
\[
= S
\]

Therefore $S = Y (I - P) Y^T$ and is SPD.

And the pseudo inverse $W^+ = W^T (WW^T)^{-1}$ is given by

\[
W^+ = W^T S^{-1}
\]
\[
W^+ = (I - P) Y^T S^{-1}
\]
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \tilde{A}^+ b + W^+ f \\
= \tilde{A}^+ b - (I - P) Y^T S^{-1} Y \tilde{A}^+ b
\]
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \tilde{A}^+ b + W^+ f \\
= \tilde{A}^+ b - (I - P) Y^T S^{-1} Y \tilde{A}^+ b
\]

which can be computed through the 4 following steps:
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \bar{A}^+ b + W^+ f
= \bar{A}^+ b - (I - P) Y^T S^{-1} Y \bar{A}^+ b
\]

which can be computed through the 4 following steps:

- Build \( w = \bar{A}^+ b \) and then by simple restriction set \( f = - Yw \)
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \bar{A}^+ b + W^+ f
\]

\[
= \bar{A}^+ b - (I - P) Y^T S^{-1} Y \bar{A}^+ b
\]

which can be computed through the 4 following steps:

- **Build** \( w = \bar{A}^+ b \) and then by simple restriction set \( f = -Yw \)
- **Solve** \( Sz = f \) (\( S \) should be small enough)
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
y
\end{bmatrix} = \tilde{A}^+ b + W^+ f
\]

\[
= \tilde{A}^+ b - (I - P) Y^T S^{-1} Y \tilde{A}^+ b
\]

which can be computed through the 4 following steps:

▶ Build \( w = \tilde{A}^+ b \) and then by simple restriction set \( f = -Yw \)
▶ Solve \( Sz = f \) (\( S \) should be small enough)
▶ Expand \( z \) and then project it onto the null space of \( \tilde{A} \) viz.
  \[ u = (I - P) Y^T z \]
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = \bar{A}^+ b + W^+ f
\]

\[
= \bar{A}^+ b - (I - P) Y^T S^{-1} Y \bar{A}^+ b
\]

which can be computed through the 4 following steps:

- **Build** \( w = \bar{A}^+ b \) and then by simple restriction set \( f = -Yw \)
- **Solve** \( Sz = f \) (\( S \) should be small enough)
- **Expand** \( z \) and then project it onto the null space of \( \bar{A} \) viz. \( u = (I - P) Y^T z \)
- **Then sum** \( w + u \) to obtain the solution \( \begin{bmatrix} x \\ y \end{bmatrix} \) (where \( y = 0 \))
Computational Ingredients

The solution is thus given by:

\[
\begin{bmatrix} x \\ y \end{bmatrix} = \tilde{A}^+ b + W^+ f = \tilde{A}^+ b - (I - P) Y^T S^{-1} Y \tilde{A}^+ b
\]

which can be computed through the 4 following steps:

▶ Build \( w = \tilde{A}^+ b \) and then by simple restriction set \( f = -Yw \)
▶ Solve \( Sz = f \) (\( S \) should be small enough)
▶ Expand \( z \) and then project it onto the null space of \( \tilde{A} \) viz. \( u = (I - P) Y^T z \)
▶ Then sum \( w + u \) to obtain the solution \( \begin{bmatrix} x \\ y \end{bmatrix} \) (where \( y = 0 \))

Note that we don’t need to build \( B \), only \( S \) is used
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

R6 (132k), non-symmetric, 16 partitions on 32 cores

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 16)</th>
<th>ABCD (size S = 8536)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>0.2 s.</td>
<td>0.2s.</td>
</tr>
<tr>
<td>CG</td>
<td>(521itr) 107.6 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>0.16s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>5.5s.</td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>1.2s.</td>
</tr>
</tbody>
</table>
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

bmw3_2 (227k) 16 partitions on 32 cores

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 1)</th>
<th>ABCD (size S = 16695)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>1.7 s.</td>
<td>1.97s.</td>
</tr>
<tr>
<td>CG</td>
<td>(Failed) 176.5 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>0.6s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>40.0s.</td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>18.0s.</td>
</tr>
</tbody>
</table>

ES stands for Exploit-Sparsity a feature available in the future release of MUMPS
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

Hamrle3 (1.447M), non-symmetric, 64 partitions on 32 cores

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 4)</th>
<th>ABCD (size S = 54608)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>2.13 s.</td>
<td>3.4s.</td>
</tr>
<tr>
<td>CG</td>
<td>(615itr) 282.90 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>4.6s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>145.4s.</td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>49.1s.</td>
</tr>
</tbody>
</table>
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

Hamrle3 (1.447M) 64 partitions on 32 cores - MUMPS_TRUNK

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 4)</th>
<th>ABCD (size $S = 54608$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>2.13 s.</td>
<td>2.7s.</td>
</tr>
<tr>
<td>CG</td>
<td>(615itr) 282.90 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>4.6s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>97.0s.</td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>47.1s.</td>
</tr>
</tbody>
</table>
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

Hamrle3 (1.447M) 64 partitions on 32 cores - MUMPS_TRUNK+ES

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 4)</th>
<th>ABCD (size S = 54608)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>2.13 s.</td>
<td>2.7s.</td>
</tr>
<tr>
<td>CG</td>
<td>(615itr) 282.90 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>4.6s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>58.4s.</td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>43.1s.</td>
</tr>
</tbody>
</table>
ABCD results

32 cores shared memory machine (MUMPS 4.10 as direct solver)

Hamrle3 (1.447M) 64 partitions on 32 cores - MUMPS_TRUNK+ES

<table>
<thead>
<tr>
<th></th>
<th>BC (blk. size = 4)</th>
<th>ABCD (size $S = 54608$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fact.</td>
<td>2.13 s.</td>
<td>2.7s.</td>
</tr>
<tr>
<td>CG</td>
<td>(615itr) 282.90 s.</td>
<td>-</td>
</tr>
<tr>
<td>Augmentation</td>
<td>-</td>
<td>4.6s.</td>
</tr>
<tr>
<td>Build S</td>
<td>-</td>
<td>145s. → <strong>58.4s.</strong></td>
</tr>
<tr>
<td>Fact S</td>
<td>-</td>
<td>43.1s.</td>
</tr>
</tbody>
</table>

ES stands for **Exploit-Sparsity** a feature available in the future release of MUMPS
Thoughts and possible orientations

Current situation: Size of $S$

- sparsity structure (preprocessing, permutations...)
- number and size of partitions
- interconnections between partitions

<table>
<thead>
<tr>
<th>Matrix</th>
<th>N</th>
<th>Pts</th>
<th>Size of $S$</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hamrle3</td>
<td>1,447,360</td>
<td>64</td>
<td>54,608</td>
<td>3.8%</td>
</tr>
<tr>
<td>R6</td>
<td>132,106</td>
<td>16</td>
<td>8,536</td>
<td>6.5%</td>
</tr>
<tr>
<td>ohne2</td>
<td>181,343</td>
<td>16</td>
<td>48,920</td>
<td>27%</td>
</tr>
</tbody>
</table>
Thoughts and possible orientations

Current situation: Size of $S$

▷ sparsity structure (preprocessing, permutations...)
▷ number and size of partitions
▷ interconnections between partitions

Possible solutions

▷ Relax the augmentation process by reducing the number of columns in $C$ and therefore reduce the size of $S$
▷ Avoid building $S$ by using implicitly (MV products) in an iterative process (CG, $S$ is SPD)
Relaxation of the augmentation process

Target:
- A reduced size of $S$ with respect to the size of $A$: better control of memory requirements

Issues:
- The augmented partitions $\tilde{A}_i$ lose "partly" their mutual numerical orthogonality
- $(I - P)$ is no longer explicitly available, and must be recovered via an iterative process
Relaxation of the augmentation process

**Target:**
- A reduced size of $S$ with respect to the size of $A$ : better control of memory requirements

**Issues:**
- The augmented partitions $\tilde{A}_i$ lose "partly" their mutual numerical orthogonality
- $(I - P)$ is no longer explicitly available, and must be recovered via an iterative process

**Results on bayer01**

<table>
<thead>
<tr>
<th>Drop threshold</th>
<th>0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of $S$</td>
<td>752</td>
<td>270</td>
<td>77</td>
<td>46</td>
<td>18</td>
<td>-</td>
</tr>
<tr>
<td>$w = \tilde{A}^+ b$</td>
<td>1</td>
<td>103</td>
<td>282</td>
<td>466</td>
<td>1183</td>
<td>1700</td>
</tr>
<tr>
<td>AVG. iter per column</td>
<td>1</td>
<td>17</td>
<td>45</td>
<td>64</td>
<td>340</td>
<td>-</td>
</tr>
<tr>
<td>Total iterations to build $S$</td>
<td>752</td>
<td>4590</td>
<td>3465</td>
<td>2944</td>
<td>6130</td>
<td>-</td>
</tr>
</tbody>
</table>
Relaxation of the augmentation process

**Target:**
- A reduced size of $S$ with respect to the size of $A$ : better control of memory requirements

**Issues:**
- The augmented partitions $\bar{A}_i$ lose "partly" their mutual numerical orthogonality
- $(I - P)$ is no longer explicitly available, and must be recovered via an iterative process

**In general:**
- Build a reduced size $S$
- Outperforms regular Block-Cimmino after a few successive solves (in the bayer01’s case after 15 solves with a drop of 0.3)
- Slower to build (less parallel advantages + iterations)
Iterative solution of $Sz = f$

Recall that $S = Y(I - P)Y^T$ is SPD, therefore the system $Sz = f$ can be solved using CG.
Iterative solution of $Sz = f$

Recall that $S = Y(I - P)Y^T$ is SPD, therefore the system $Sz = f$ can be solved using CG. In the CG iteration, $S$ is used implicitly in the instruction:

$$\alpha_k = \left( r_k^T r_k \right) / \left( p_k^T S p_k \right)$$
Iterative solution of $Sz = f$

Recall that $S = Y(I - P)Y^T$ is SPD, therefore the system $Sz = f$ can be solved using CG. In the CG iteration, $S$ is used implicitly in the instruction:

$$\alpha_k = \left( r_k^T r_k \right) / \left( p_k^T Sp_k \right)$$

The matrix-vector product can be written as:

$$Sp_k = Y(I - P)Y^T p_k$$
$$= p_k - YPY^T p_k$$
Iterative solution of $Sz = f$

Recall that $S = Y(I - P)Y^T$ is SPD, therefore the system $Sz = f$ can be solved using CG. In the CG iteration, $S$ is used implicitly in the instruction:

$$\alpha_k = \left( r_k^T r_k \right) / \left( p_k^T Sp_k \right)$$

The matrix-vector product can be written as:

$$Sp_k = Y(I - P)Y^T p_k$$

$$= p_k - YPY^T p_k$$

Where $Y^T p_k = \begin{bmatrix} 0 \\ p_k \end{bmatrix}$ is to be projected by solving augmented systems using MUMPS.
Iterative solution of $S z = f$ : Test

- Conjugate Gradient acceleration with stopping criteria $1 \times 10^{-8}$.
- A testing (rudimentary) preconditioner (partial build of $S$)
Iterative solution of $Sz = f$ : Test

- Conjugate Gradient acceleration with stopping criteria $1 \times 10^{-8}$.
- A testing (rudimentary) preconditioner (partial build of $S$)

<table>
<thead>
<tr>
<th></th>
<th>size($S$)</th>
<th>CG</th>
<th>PCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayer01</td>
<td>752</td>
<td>F</td>
<td>257</td>
</tr>
<tr>
<td>Hamrle3</td>
<td>54,608</td>
<td>1,911</td>
<td>1,238</td>
</tr>
<tr>
<td>R6</td>
<td>8,536</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>ohne2</td>
<td>48,920</td>
<td>32,301</td>
<td>8,066</td>
</tr>
</tbody>
</table>
The R6 case

Smallest eig. = 2.95252e−12; Largest eig. = 1.00000e+00

Eigenvalue distribution of S
Conclusion

- Building $S$ is fast (working on making it faster)
- ABCD can solve block-Cimmino convergence issues

- $S$ can be really large and containing a large number of entries
- Reducing the size of $S$ can perform better than block-Cimmino in the long run
- Iteratively solving $Sz = f$ is not ready yet, currently studying possible solutions