The Augmented Block-Cimmino Distributed Method

Improvements to the Conjugate Gradient accelerated Block Cimmino Method

Mohamed Zenadi[1]

Iain S. Duff[2] Ronan Guivarch[1] Daniel Ruiz[1]

IRIT - ENSEEIHT[1]
CERFACS and Rutherford Appleton Laboratory[2]

September 13, 2013

Block Cimmino : The basic facts

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

Partitioning the system $Ax = b$

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}$$

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

$$\begin{aligned} \delta_i^{(k)} &= A_i^+ b_i - P_{\mathcal{R}(A_i^T)} x^{(k)} \\ x^{(k+1)} &= x^{(k)} + \nu \sum_{i=1}^{p} \delta_i^{(k)} \end{aligned}$$

where: $A_i^+ = A_i^T \left(A_i A_i^T\right)^{-1}$ and $P_{\mathcal{R}(A_i^T)} = A_i^+ A_i$

Block Cimmino : The basic facts

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

Acceleration

Apply CG to solve the SPD system

$$\sum_{i=1}^{p} A_i^+ A_i \; x = \sum_{i=1}^{p} A_i^+ b_i$$

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

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

$$\text{where : } u_i = A_i^+ (b_i - A_i x) = \delta_i$$

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

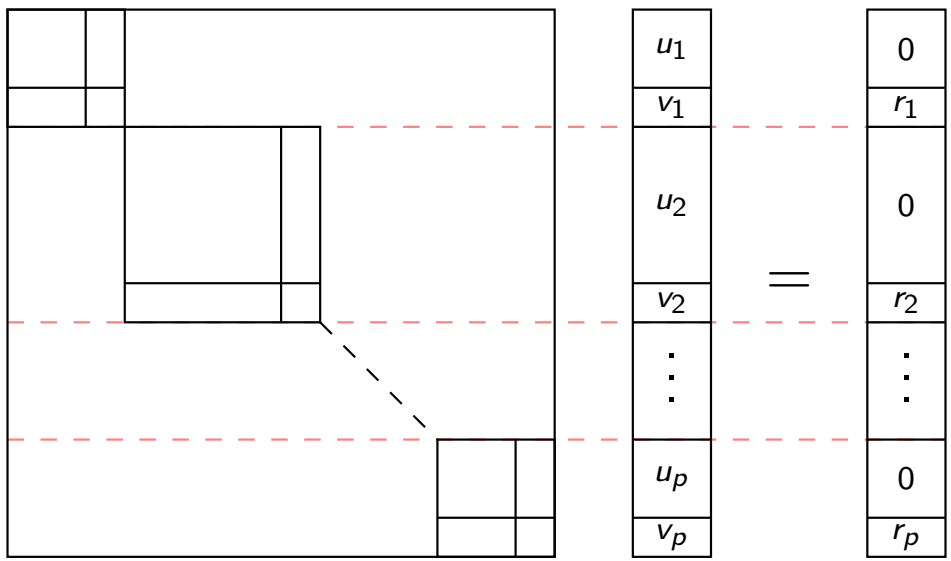
Solve independently using a direct solver the systems for each partition

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

$$\text{where : } u_i = A_i^+ (b_i - A_i x) = \delta_i$$

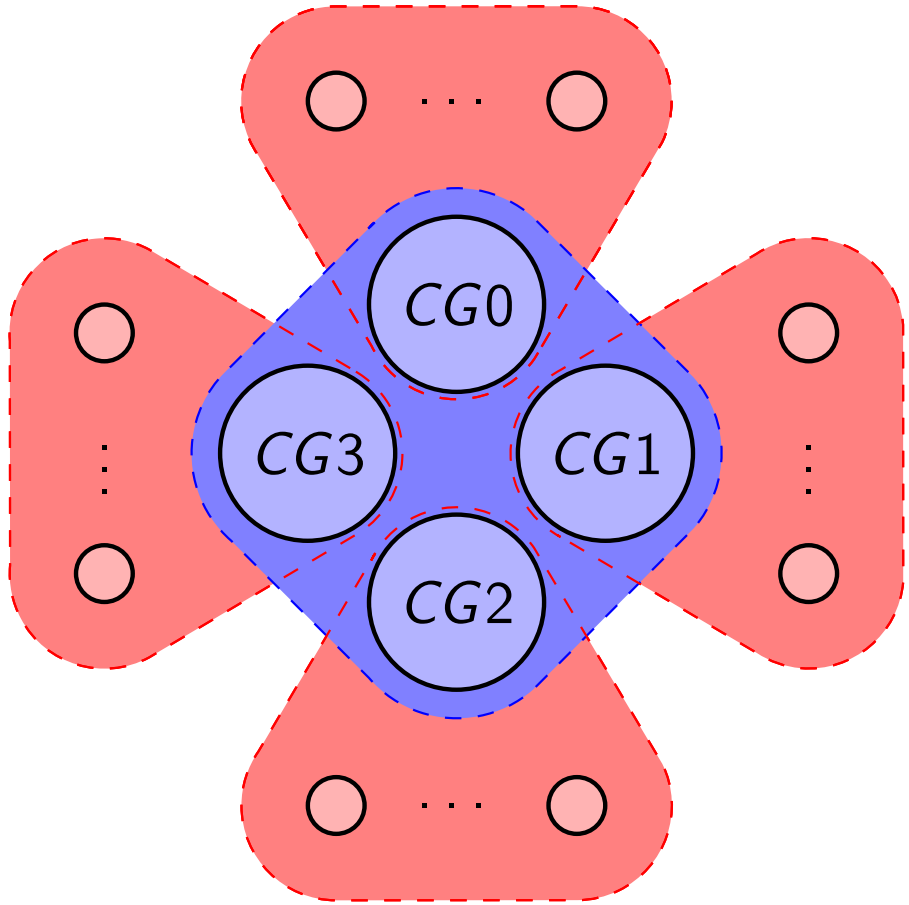
The Hybrid Idea : Computing projections

- 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

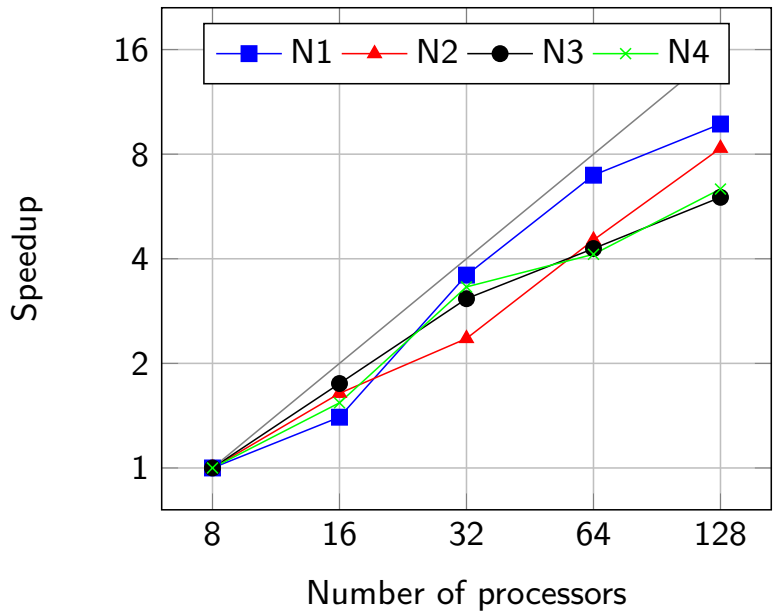
There is only a single Block-CG distributed over these processes.

Parallelism results: Distributed B-CG

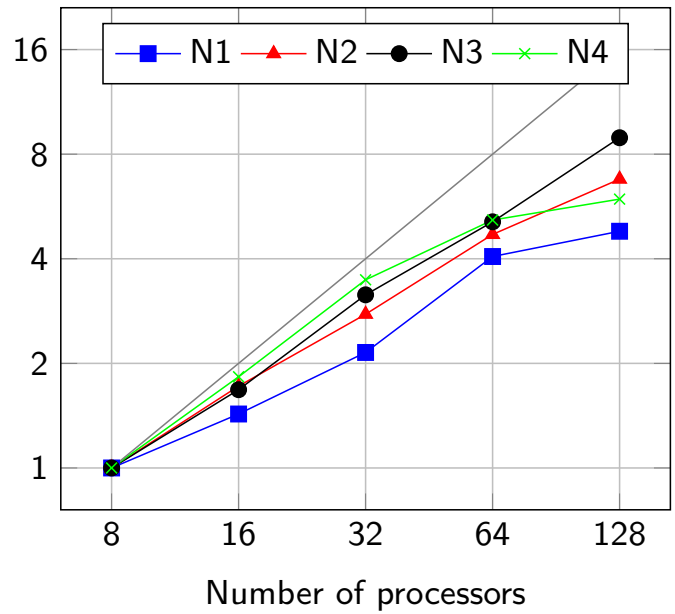
Problem	**Size**	**Nonzeros**	**Application**
N_1: torso3	259,156	4,429,042	3D model of torso
N_2: CoupCons3D	416,800	17,277,420	structural problem
N_3: cage13	445,315	7,479,343	DNA electrophoresis
N_4: Hamrle3	1,447,360	5,514,242	Circuit Simulation

- 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



(b) B-CG speedup

Problem	Partitions	Factorization at 8 Cores	B-CG at 8 cores
N_1	16	12.11 s.	6.34 s.
N_2	32	14.41 s.	63.49 s.
N_3	256	28.15 s.	13.64 s.
N_4	64	6.67 s.	773 s.

Block Cimmino vs MUMPS

32 cores shared memory machine

Problem	[MUMPS] Factorization	[BC] Factorization	B-CG
torso3	5.03 s.	3.21 s.	5.18 s.
Cage13	1452.44 s.	9.31 s.	3.18 s.
Hamrle3	413.21 s.	2.13 s.	282.90 s.

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
  - `BC + Cage13` : MAX : 685MB / AVG : 263MG

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)

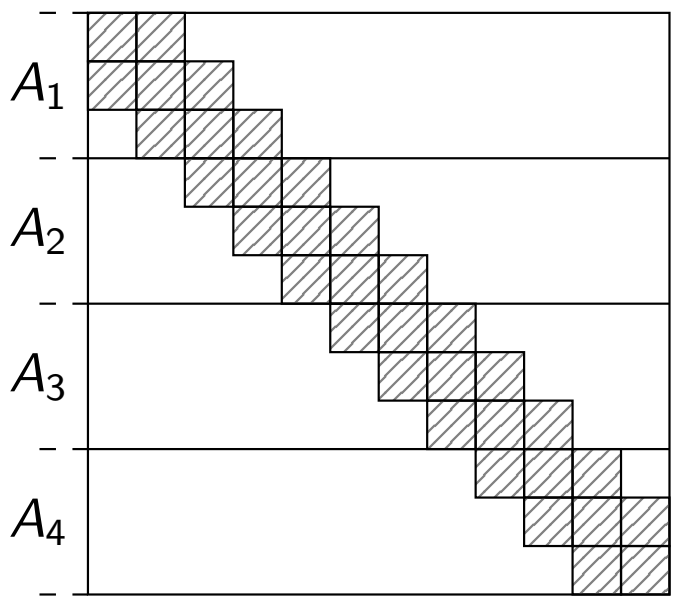
Proposed solution:

- 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

$\Rightarrow$ **A**ugmented **B**lock **C**immino **D**istributed solver (`ABCD solver`)

The augmentation process

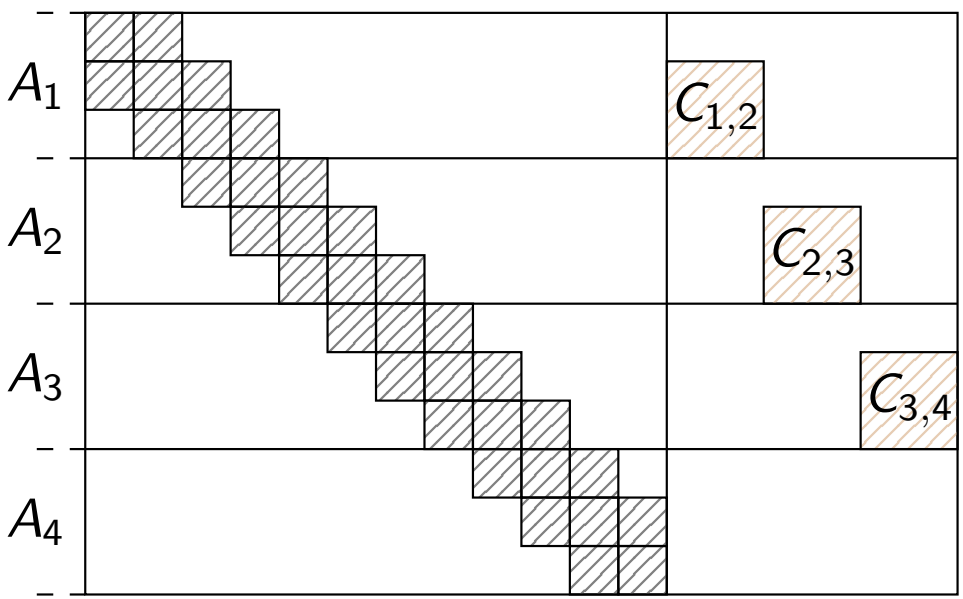
- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)



`Illustrative example`

The augmentation process

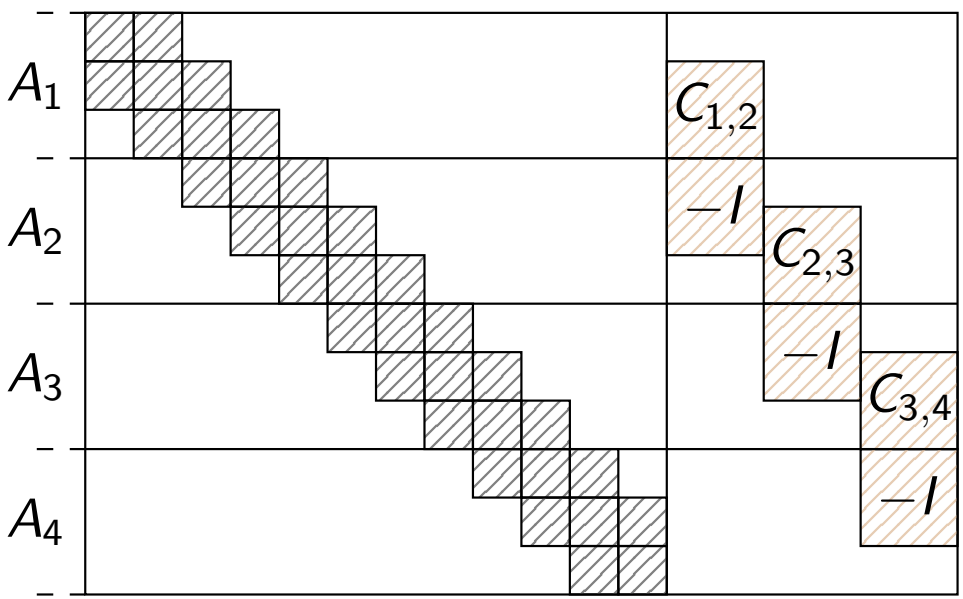
- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions ($j > i$), expand with $C_{i,j} = A_i A_j^T$,



`Illustrative example`

The augmentation process

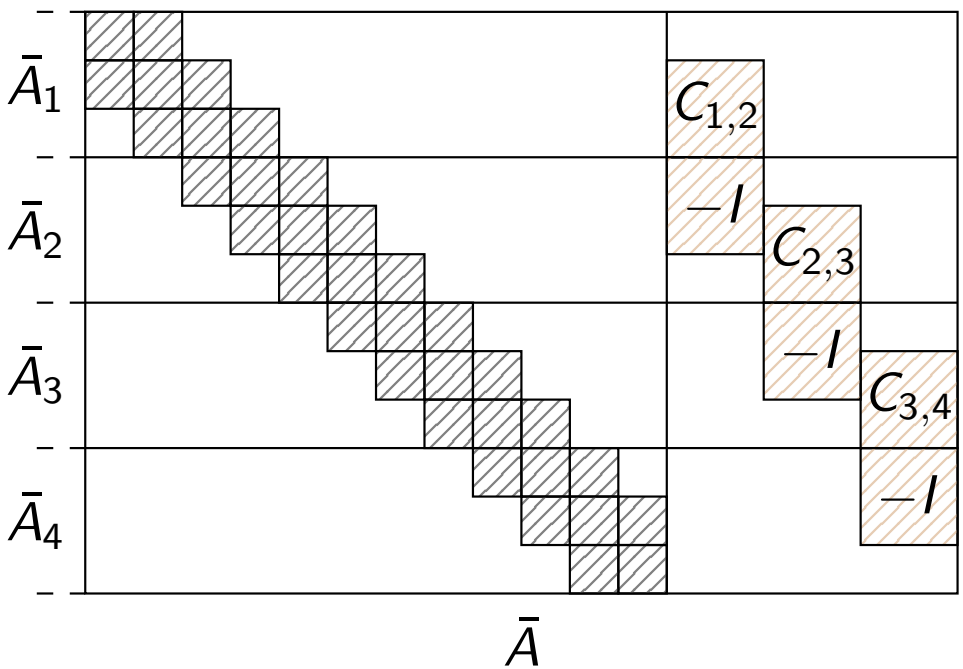
- Partition the matrix with respect to its structure (not necessarily in block-tridiagonal form)
- For each pair of partitions ($j > i$), expand with $C_{i,j} = A_i A_j^T$, and enforce numerical orthogonality



`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 $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$



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 $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$
- 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 $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$
- 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 $Y = \begin{bmatrix} 0 & I \end{bmatrix}$, linked to the constraints equations, is not orthogonal to the previous partitions in $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$.

The augmentation process

To enforce this orthogonality, we project the column vectors Y^T onto the null space of $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$ (orthogonal complement of $\mathcal{R}\left(\bar{A}^T\right)$) :

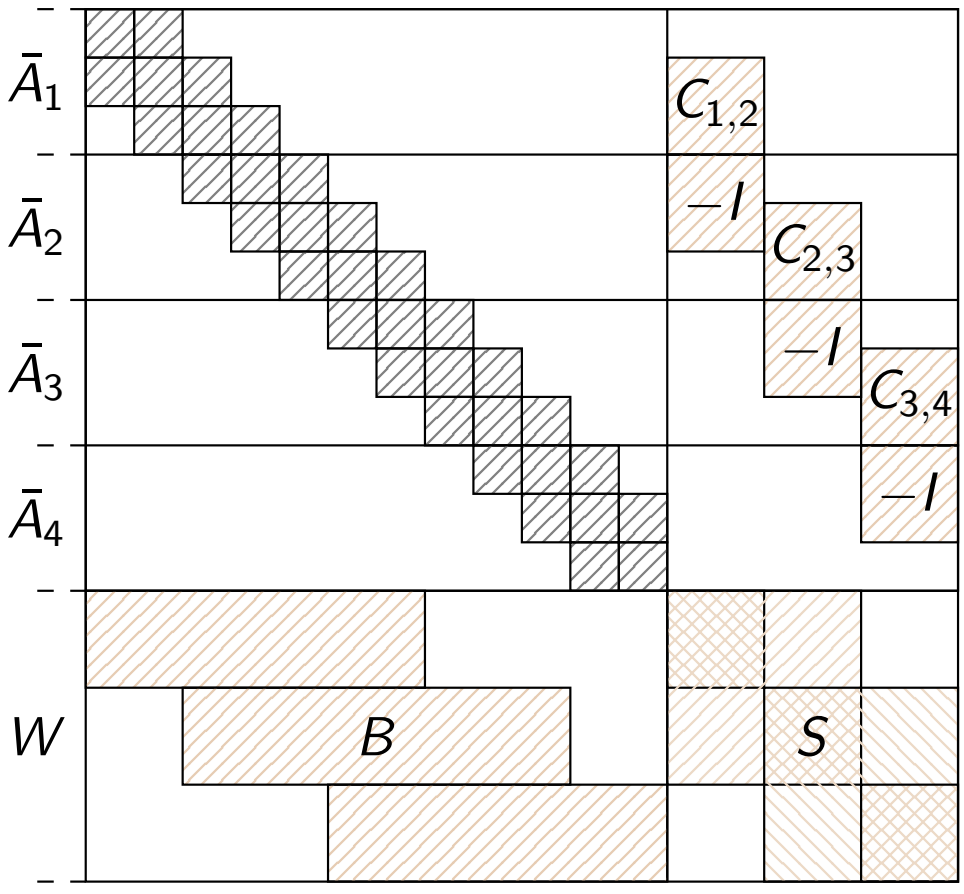
$$W^T = (I - P)\, Y^T,$$

where (as a result of the enforced orthogonality) :

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

We finally obtain $\begin{bmatrix} A & C \\ B & S \end{bmatrix}$, where $\begin{bmatrix} B & S \end{bmatrix} = W$, 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 :

$$\begin{aligned} f &= \begin{bmatrix} B & S \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} &&= Y(I-P) \begin{bmatrix} x \\ 0 \end{bmatrix} \\ &&&= -YP \begin{bmatrix} x \\ 0 \end{bmatrix} \qquad (\text{since } Y = \begin{bmatrix} 0 & I \end{bmatrix}) \\ &&&= -Y\bar{A}^{+}\bar{A} \begin{bmatrix} x \\ 0 \end{bmatrix} \\ f &&&= -Y\bar{A}^{+}b \end{aligned}$$

Implicit Direct Solver

Since all the partitions in the new equivalent linear system

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

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

$$\begin{aligned} \begin{bmatrix} x \\ y \end{bmatrix} &= \bar{A}^{+} b + W^{+} f \\ &= \bar{A}^{+} b - W^{+} Y \bar{A}^{+} b \\ &= \sum_{i=1}^{p} \bar{A}_i^{+} b_i - W^{+} Y \sum_{i=1}^{p} \bar{A}_i^{+} b_i \end{aligned}$$

Computational Ingredients

Knowing that $W = \begin{bmatrix} B & S \end{bmatrix} = Y\,(I - P)$, with $Y = \begin{bmatrix} 0 & I \end{bmatrix}$, we have :

$$\begin{aligned} WW^T &= Y\,(I-P)\,(I-P)^T\,Y^T \\ &= Y\,(I-P)^2\,Y^T \\ &= Y\,(I-P)\,Y^T \\ &= \begin{bmatrix} B & S \end{bmatrix} Y^T \\ &= S \end{aligned}$$

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

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

$$\begin{aligned} W^+ &= W^T S^{-1} \\ W^+ &= (I-P)\,Y^T S^{-1} \end{aligned}$$

Computational Ingredients

The solution is thus given by :

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

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 :

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 :

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

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.

$$u = (I - P) Y^{T} z$$

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 :

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

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 $y = 0$)

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

	BC (blk. size = 16)	**ABCD** (size S = 8536)
Fact.	0.2 s.	0.2s.
CG	*(521itr)* 107.6 s.	-
Augmentation	-	0.16s.
Build S	-	5.5s.
Fact S	-	1.2s.

ABCD results

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

`bmw3_2 (227k)` 16 partitions on 32 cores

	BC (blk. size $=$ 1)	**ABCD** (size $S = 16695$)
Fact.	1.7 s.	1.97s.
CG	*(Failed)* 176.5 s.	-
Augmentation	-	0.6s.
Build S	-	40.0s.
Fact S	-	18.0s.

ABCD results

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

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

	BC (blk. size = 4)	**ABCD** (size $S = 54608$)
Fact.	2.13 s.	3.4s.
CG	*(615itr)* 282.90 s.	-
Augmentation	-	4.6s.
Build S	-	145.4s.
Fact S	-	49.1s.

ABCD results

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

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

	BC (blk. size = 4)	**ABCD** (size $S = 54608$)
Fact.	2.13 s.	2.7s.
CG	*(615itr)* 282.90 s.	-
Augmentation	-	4.6s.
Build S	-	**97.0s.**
Fact S	-	47.1s.

ABCD results

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

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

	BC (blk. size = 4)	**ABCD** (size $S = 54608$)
Fact.	2.13 s.	2.7s.
CG	*(615itr)* 282.90 s.	-
Augmentation	-	4.6s.
Build S	-	**58.4s.**
Fact S	-	43.1s.

ABCD results

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

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

	BC (blk. size = 4)	**ABCD** (size $S = 54608$)
Fact.	2.13 s.	2.7s.
CG	*(615itr)* 282.90 s.	-
Augmentation	-	4.6s.
Build S	-	145s. $\rightarrow$ **58.4s.**
Fact S	-	43.1s.

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

Matrix	**N**	**Pts**	**Size of S**	**Ratio**
`Hamrle3`	1,447,360	64	54,608	3.8%
`R6`	132,106	16	8,536	6.5%
`ohne2`	181,343	16	48,920	27%

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

Results on `bayer01`

Drop threshold	0	0.1	0.2	0.3	0.4	BC
Size of S	752	270	77	46	18	-
$w = \bar{A}^+ b$	1	103	282	466	1183	1700
AVG. iter per column	1	17	45	64	340	-
Total iterations to build S	752	4590	3465	2944	6130	-

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:

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

The matrix-vector product can be writen as:

$$\begin{aligned} Sp_k &= Y(I - P)Y^T p_k \\ &= p_k - YPY^T p_k \end{aligned}$$

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:

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

The matrix-vector product can be writen as:

$$\begin{aligned} Sp_k &= Y(I - P)Y^T p_k \\ &= p_k - YPY^T p_k \end{aligned}$$

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 $Sz = f$: Test

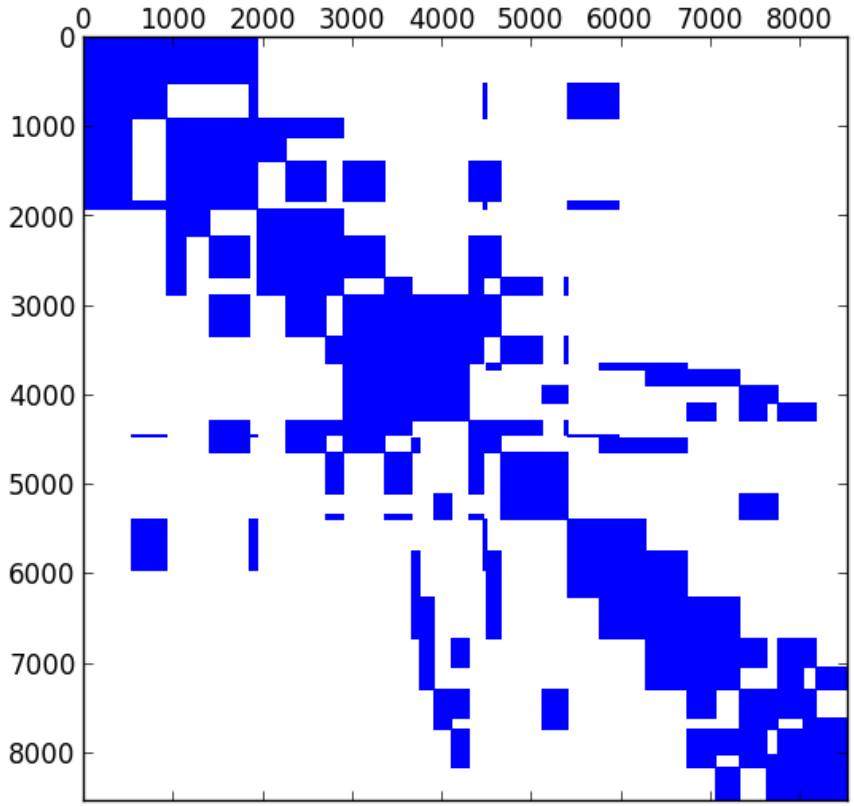
- Conjugate Gradient acceleration with stopping criteria 1×10^{-8}.
- A testing (rudimentary) preconditioner (partial build of S)

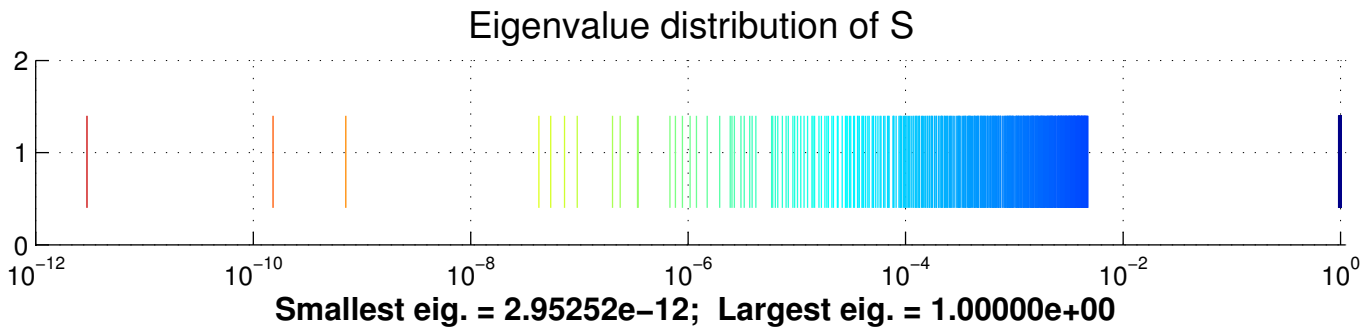
Iterative solution of $Sz = f$: Test

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

	size(S)	CG	PCG
`bayer01`	752	`F`	257
`Hamrle3`	54,608	1,911	1,238
`R6`	8,536	`F`	`F`
`ohne2`	48,920	32,301	8,066

The R6 case


0
1000
2000
3000
4000
5000
6000
7000
8000


Eigenvalue distribution of S
2
1
0
10^{-12}
10^{-10}
10^{-8}
10^{-6}
10^{-4}
10^{-2}
10^{0}
Smallest eig. = 2.95252e−12; Largest eig. = 1.00000e+00

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