First Excursion in Parallel Preconditioning

Deflation Methods

Riccardo Rossi

Concept of Scalability

- Strong Scalability → Solve in less time a given problem by using more resources → Largely about implementation
- Weak Scalability →Use more resources to solve, in the same time, a larger problem → Largely about algorithms

Number of iterations DEPENDS ON h

Let’s consider our Laplacian problem (really in the 2D and 3D cases).

The condition number can be proved to be $O(1/h^2)$

That is, **the problem becomes harder to solve when the element size shrinks.**

Target toy problem

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{12}^t & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{13}^t & A_{11}^t & A_{33} & A_{34} & A_{35} \\ A_{14}^t & A_{24}^t & A_{34}^t & A_{44} & A_{45} \\ A_{15}^t & A_{25}^t & A_{35}^t & A_{45}^t & A_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

Where we expect **A** to be SPD

Jacobi **Preconditioner**

$$\begin{pmatrix} x_1^{i+1} \\ x_2^{i+1} \\ x_3^{i+1} \\ x_4^{i+1} \\ x_5^{i+1} \end{pmatrix} = \begin{pmatrix} x_1^{i} \\ x_2^{i} \\ x_3^{i} \\ x_4^{i} \\ x_5^{i} \end{pmatrix} + \begin{pmatrix} A_{11} & 0 & 0 & 0 & 0 \\ 0 & A_{22} & 0 & 0 & 0 \\ 0 & 0 & A_{33} & 0 & 0 \\ 0 & 0 & 0 & A_{44} & 0 \\ 0 & 0 & 0 & 0 & A_{55} \end{pmatrix}^{-1} \left(\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} - \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{12}^t & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{13}^t & A_{11}^t & A_{33} & A_{34} & A_{35} \\ A_{14}^t & A_{24}^t & A_{34}^t & A_{44} & A_{45} \\ A_{15}^t & A_{25}^t & A_{35}^t & A_{45}^t & A_{55} \end{pmatrix} \begin{pmatrix} x_1^{i} \\ x_2^{i} \\ x_3^{i} \\ x_4^{i} \\ x_5^{i} \end{pmatrix} \right)$$

NOTE: if we had 5 processors, we could update independently the 5 components of x

PERFECT PARALLELISM! (but as discussed, it is a weak preconditioner)

Block Jacobi **Preconditioner**

$$\begin{pmatrix} x_1^{i+1} \\ x_2^{i+1} \\ x_3^{i+1} \\ x_4^{i+1} \\ x_5^{i+1} \end{pmatrix} = \begin{pmatrix} x_1^{i} \\ x_2^{i} \\ x_3^{i} \\ x_4^{i} \\ x_5^{i} \end{pmatrix} + \left(\begin{pmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ A_{12}^t & A_{22} & 0 & 0 & 0 \\ 0 & 0 & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{34}^t & A_{44} & 0 \\ 0 & 0 & 0 & 0 & A_{55} \end{pmatrix} \right)^{-1} \left(\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} - \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{12}^t & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{13}^t & A_{11}^t & A_{33} & A_{34} & A_{35} \\ A_{14}^t & A_{24}^t & A_{34}^t & A_{44} & A_{45} \\ A_{15}^t & A_{25}^t & A_{35}^t & A_{45}^t & A_{55} \end{pmatrix} \begin{pmatrix} x_1^{i} \\ x_2^{i} \\ x_3^{i} \\ x_4^{i} \\ x_5^{i} \end{pmatrix} \right)$$

If we had 3 processors, we could also decide to update each color independently on the owner processor.

This becomes a better preconditioner (closer to the exact inverse) as the number of subdomains diminishes.

HOWEVER:

1 – cost of preconditioner is higher for fewer processors

2 – total number of iterations changes with the number of processors used (gets worst as we add parallelism)

→NOT A SCALABLE PRECONDITIONER! (neither in weak nor strong sense)

3 – work imbalance may often become problematic

Why Block preconditioner not (so) effective?

The number of iterations is governed by $k(PA) = {}^{\lambda_{max}}/_{\lambda_{min}}$

Block preconditioning improves the behaviour on eigenvectors “in the middle” but not of the extreme values.

This is so since:

- High frequency behaviour is still triggered by “jumps” across the borders of the subdomains
- Global behaviour is not improved since the preconditioner does not span the entire domain but only subparts of it.

How to control rate of convergence?

We need to design our preconditioner $\boldsymbol{P}$ such that $\mathrm{k}(\boldsymbol{PA}) \ll k(\boldsymbol{A})$

- Either by making smaller λ_{max} → getting rid of high frequency modes
- Or by making larger λ_{min} → finding a way to solve for the “smooth frequencies”

And … we want to do that in parallel.

Deflation Solver – getting rid of low frequency modes

Let's imagine for a second we know the m lowest eigenvalues in the system (**although we don't in real life!!).**

→We organize the eigenvectors as the column of a matrix $\boldsymbol{Z_{N*M}}$

Now if we want to solve the problem $\boldsymbol{Ax = b}$ we may look for a special case solution in the form $\boldsymbol{x = y + Z\lambda}$ (here λ is simply a symbol, nothing to do with the eigenvalues of A)

We hence need to fulfil $\boldsymbol{Ay + AZ\lambda = b}$ **(which corresponds to making the residual orthogonal to the full basis of $\boldsymbol{R^N}$**

PROBLEM: we now have N+M unkowns instead of just N!

IDEA: look also for the solution of $\boldsymbol{Ax = b}$ in the space spanned by the columns of $\boldsymbol{Z}$ → $\boldsymbol{Z^t Ay + Z^t AZ\lambda = Z^t b}$

Deflation Solver

Those two conditions together take the form

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{AZ} \\ \boldsymbol{Z^t A} & \boldsymbol{Z^t AZ} \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{Z^t b} \end{pmatrix} \rightarrow \begin{pmatrix} \boldsymbol{A} & \boldsymbol{AZ} \\ \boldsymbol{Z^t A} & \boldsymbol{E} \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{Z^t b} \end{pmatrix}$$

$$\boldsymbol{E_{M*M} := Z^t AZ}$$

Which **is a singular system even if A** was originally SPD

Next step is to express symbolically $\boldsymbol{\lambda}$ in terms of $\boldsymbol{y}$ (static condensation)

$$\boldsymbol{\lambda = \mathbf{E}^{-1} Z^t (b - Ay)}$$

Which substituted into the first gives

$$\boldsymbol{Ay + AZ\mathbf{E}^{-1} Z^t (b - Ay) = \mathbf{b}}$$

$$\boldsymbol{\rightarrow \left(I - AZ\mathbf{E}^{-1} Z^t\right) Ay = \left(I - AZ\mathbf{E}^{-1} Z^t\right)\mathbf{b}}$$

Deflation Solver

The last equation can be written in **terms of a projection P onto a subspace V_Z of $\mathbf{R^N}$ orthogonal to the eigenvectors contained in Z**

$$P(Ay) = P(b)$$

With

$$P(x) := \left(I - AZE^{-1}Z^t\right)x$$

Note that It can be easily verified that $P(x) = P\left(P(x)\right)$

The nice thing is that since we look for $y \in V_Z$, y has no component onto the lowest eigenmodes, which are thus effectively taken out from the system which thus sees improved its conditioning.

REMARK: The problem $P(Ay) = P(b)$ is actually Rank deficient (deflated of m eigenvectors!), nevertheless we can solve it using CG (or PCG) since y is guaranteed to be orthogonal to the Kernel

Important “Detail”

We have however an outstanding problem: **we don’t know the lowest eigenvalues**

It turns out that **it is good enough to use instead slowly varying solutions,** defined globally within the entire domain.

One option, called **“constant deflation” is to take the columns of Z to be 1 in one of the subdomains, and zero on all the others.**

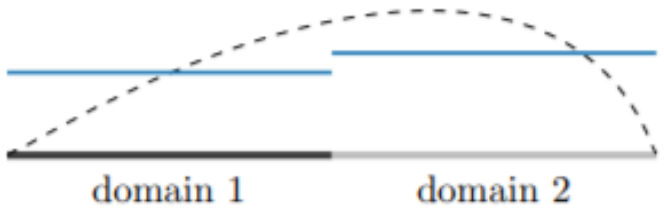
For example for our “toy problem”

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{12}^t & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{13}^t & A_{11}^t & A_{33} & A_{34} & A_{35} \\ A_{14}^t & A_{24}^t & A_{34}^t & A_{44} & A_{45} \\ A_{15}^t & A_{25}^t & A_{35}^t & A_{45}^t & A_{55} \end{pmatrix} \begin{pmatrix} x_1^i \\ x_2^i \\ x_3^i \\ x_4^i \\ x_5^i \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

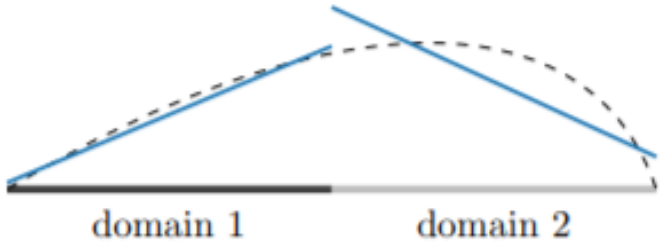
We would take out Z as

$$Z := \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Visual Interpretation of Deflation



(a) Constant deflation



(b) Linear deflation

Simplest approach of having 1 constant per subdomain

One more level of sofistication would be to allow deflated functions to behave linearly.
Idea is the same but basis a Little better

Combining Constant deflation and block preconditioning

Long story short, the combination of block preconditioning and of constant deflation provides a solver that is scalable in both the weak and strong sense.

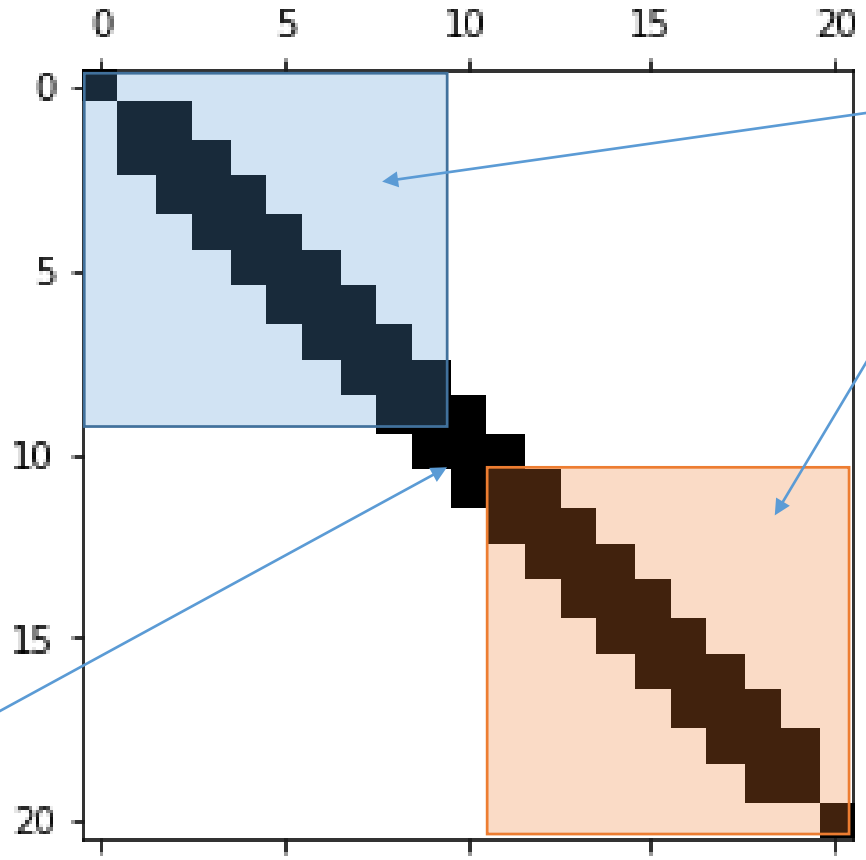
PROBLEM → from our experience still slower than AMG

Some homebrew benchmarks available here

https://arxiv.org/pdf/1710.03940.pdf

Theoretical discussion in the references therein

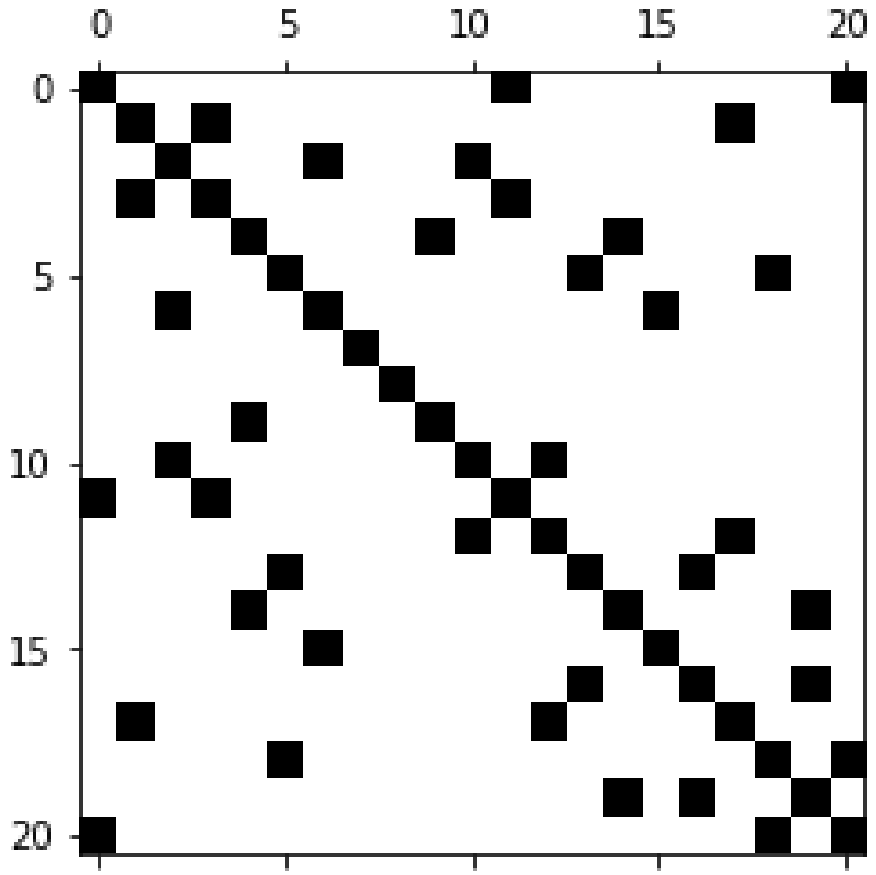
Schur complement solver



Shaded áreas only depend on the value
In the center, so once that value is known
They can be factored independently

Value in common between the domains

PROBLEM: it may not be easy to decide where to divide

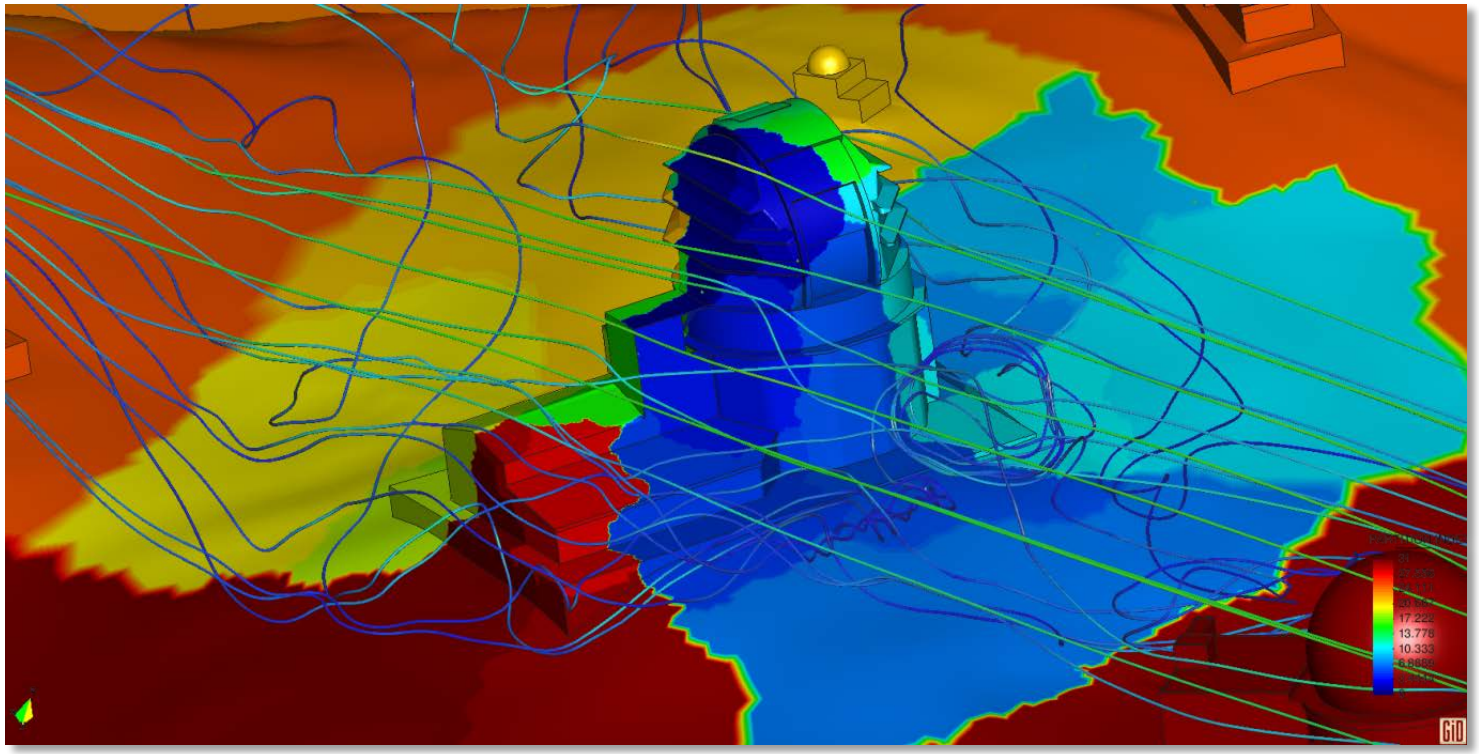


Reordering is crucially needed to make apparent structures in the matrix!

(what is shown in the figure is nothign else than a shuffling of the índices of the matrix in the previous page)

Luckily FEM allows good partitions to

Trick is to group data in disjoint volumes → automatic tools exist to do it in parallel (METIS, SCOTCH, ZOLTAN…)



Schur complement

If the situation is as before (and it typically IS in FEM, at least **after a proper reordering**)

Than we can identify sparse blocks in the matrix and reorder it to achieve something like

$$\begin{pmatrix} A_1 & \mathbf{0} & A_{1s} \\ \mathbf{0} & A_2 & A_{2s} \\ A_{1s}^t & A_{2s}^t & A_s \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_s \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_s \end{pmatrix}$$

The interesting point is that static condensation can be performed independently for the colored components, that is IN PARALLEL

$$\left(A_s - A_{1s}^t A_1^{-1} A_{1s} - A_{2s}^t A_2^{-1} A_{2s}\right) x_s = b_s - A_{1s}^t A_1^{-1} b_1 - A_{2s}^t A_2^{-1} b_2$$

This is the SCHUR COMPLEMENT of the original problem

→If we were to solve it exactly than we would have a direct solver

IDEA:

Use Matrix-Free CG for the schur complement, instead of for the original problem

IMPORTANT RESULT: For a laplacian The condition number of the schur complement is less dependent on the element size than for the original problem → The approach is more scalable than the original one. (although still not optimal!)

PROBLEM1: we need a preconditioner for the schur complement! **Difficult, since we want Matrix-free**

PROBLEM2: solution of the Schur complement is “serial” in the sense that it implies a serialization point. Furthermore the size of the schur complement is proportional to the área of intersections of the domains (instead of their volumen) which is still growing fast with the number of domains→ limit to the scalability → need for multilevel methods.

REMAINING OF THE COURSE:

Much deeper exploration of the idea, and exploration of techniques to achieve scalability (a concept that will also be further refined) while avoiding serial bottlenecks.

SOME REFERENCES

https://www-users.cs.umn.edu/~saad/PS/iter4.pdf