Parallel Design and Performance of Nested Filtering Factorization Preconditioner (NFF)

Long Qu

PhD supervisor L.Grigori

Collaborator F. Nataf

Sparse Days 2013 - June 18th, 2013

Plan

Plan

Nested Dissection

$$A = \left(\begin{array}{ccc|ccc|c} T_2^1 & & U_2^1 & & & & \\ & T_2^2 & U_2^2 & & & & U_1^1 \\ \hline L_2^1 & L_2^2 & S_2^1 & & & & \\ \hline & & & T_2^3 & & U_2^3 & \\ & & & & T_2^4 & U_2^4 & U_1^2 \\ & & & L_2^3 & L_2^4 & S_2^2 & \\ \hline & L_1^1 & & & L_1^2 & & S_1^1 \end{array}\right).$$

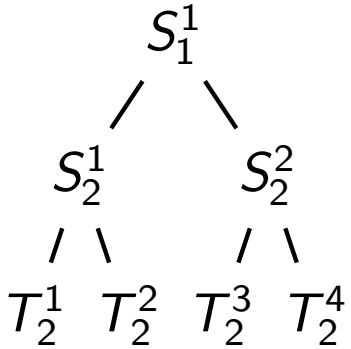


FIGURE : Separator tree of matrix A

Generalization

- Nested K-way partitioning with vertex separator.
- Nested dissection = nested 2-way partitioning with vertex separator.

Objective

Context

- Solving large linear systems of equations $Ax = b$.
- Preconditioning : $B^{-1}Ax = B^{-1}b$

Nested Filtering Factorization preconditioner (NFF)

- a multilevel parallel preconditioning technique for solving large sparse linear systems of equations by using iterative methods.
- based on nested dissection or nested K-way partitioning with vertex separator.
- using filtering techinque to preserve some direction of interest of input matrix A. (Bt=At)

Plan

Nested Exact Factorization [Grigori et al., 2010]

$$
\begin{aligned}
A &= \begin{pmatrix} T_1^1 & & U_1^1 \\ & T_1^2 & U_1^2 \\ L_1^1 & L_1^2 & S_1^1 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & & \\ & 0 & \\ L_1^1 & L_1^2 & 0 \end{pmatrix}}_{L_1} + \underbrace{\begin{pmatrix} T_1^1 & & \\ & T_1^2 & \\ & & S_1^1 \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & & U_1^1 \\ & 0 & U_1^2 \\ & & 0 \end{pmatrix}}_{U_1} \\
&= (L_1 + D)D^{-1}(D + U_1) - L_1 D^{-1} U_1 \\
&= (L_1 + F_1)F_1^{-1}(F_1 + U_1)
\end{aligned}
$$

where

$$
\begin{aligned}
F_1 &= D - L_1 F_1^{-1} U_1 \\
&= \begin{pmatrix} T_1^1 & & \\ & T_1^2 & \\ & & S_1^1 - L_1^1 (T_1^1)^{-1} U_1^1 - L_1^2 (T_1^2)^{-1} U_1^2 \end{pmatrix}
\end{aligned}
$$

Parallelism in each recursion level !

NFF preconditioner

- Nested exact factorization

$$
\begin{aligned}
A &= F_0 \\
F_k &= (L_{k+1} + F_{k+1}) F_{k+1}^{-1} (F_{k+1} + U_{k+1}), \ \textit{for } k = 0 \ldots K-1 \\
F_K &= D - \sum_{k=1}^{K} L_k F_k^{-1} U_k
\end{aligned}
$$

- NFF Preconditioner

$$F_K = D - \sum_{k=1}^{K} L_k \beta_k^{-1} U_k$$

NFF preconditioner

- Nested exact factorization

$$\begin{aligned} A &= F_0 \\ F_k &= (L_{k+1} + F_{k+1}) F_{k+1}^{-1} (F_{k+1} + U_{k+1}), \ \textit{for } k = 0 \ldots K-1 \\ F_K &= D - \sum_{k=1}^{K} L_k F_k^{-1} U_k \end{aligned}$$

- NFF Preconditioner

$$F_K = D - \sum_{k=1}^{K} L_k \beta_k^{-1} U_k$$

NFF preconditioner

The matrix β_k is a matrix introduces approximation which satisfies

$$\beta_k^{-1} U_k \mathbf{t} = F_k^{-1} U_k \mathbf{t}, \forall k \in [1..K-1]. \quad (1)$$

$\Longrightarrow$ $A\mathbf{t} = B_{NFF}\mathbf{t}$ (filtering on vector $\mathbf{t}$.)

Construction of β

- $\beta = diag((F_k^{-1} U_k t)./(U_k t))$
- we note $F_k^{-1} U_k \mathbf{t}$ as u and $U_k \mathbf{t}$ as v, $\beta = diag(u./v)$.
- As in BFD preconditioner [Grigori et al., 2012], to avoid division by zero :

$$\beta(i,j) = \begin{cases} u(i)/v(i), & \text{if } i = j \text{ and } |v(i)| > \varepsilon, \\ u(i)/v(j), & \text{if } |v(i)| \leq \varepsilon \text{ and } j = argmax(|v(l)|)_l, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

NFF preconditioner

The matrix β_k is a matrix introduces approximation which satisfies

$$\beta_k^{-1} U_k \mathbf{t} = F_k^{-1} U_k \mathbf{t}, \forall k \in [1..K-1]. \quad (1)$$

$\Longrightarrow$ $A\mathbf{t} = B_{NFF}\mathbf{t}$ (filtering on vector $\mathbf{t}$.)

Construction of β

- $\beta = diag((F_k^{-1} U_k t)./(U_k t))$
- we note $F_k^{-1} U_k \mathbf{t}$ as u and $U_k \mathbf{t}$ as v, $\beta = diag(u./v)$.
- As in BFD preconditioner [Grigori et al., 2012], to avoid division by zero :

$$\beta(i,j) = \begin{cases} u(i)/v(i), & \text{if } i = j \text{ and } |v(i)| > \varepsilon, \\ u(i)/v(j), & \text{if } |v(i)| \leq \varepsilon \text{ and } j = argmax(|v(l)|)_l, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

NFF preconditioner

The matrix β_k is a matrix introduces approximation which satisfies

$$\beta_k^{-1} U_k \mathbf{t} = F_k^{-1} U_k \mathbf{t}, \forall k \in [1..K-1]. \tag{1}$$

$\Longrightarrow$ $A\mathbf{t} = B_{NFF}\mathbf{t}$ (filtering on vector $\mathbf{t}$.)

Construction of β

- $\beta = diag((F_k^{-1} U_k t)./(U_k t))$
- we note $F_k^{-1} U_k \mathbf{t}$ as u and $U_k \mathbf{t}$ as v, $\beta = diag(u./v)$.
- As in BFD preconditioner [Grigori et al., 2012], to avoid division by zero :

$$\beta(i,j) = \begin{cases} u(i)/v(i), & \text{if } i = j \text{ and } |v(i)| > \varepsilon, \\ u(i)/v(j), & \text{if } |v(i)| \leq \varepsilon \text{ and } j = argmax(|v(l)|)_l, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Example of building β

Example : $v(1) = 0, v(4) < \varepsilon, v(5) = max(v)$

$$\begin{pmatrix} 0 & & & u(1)/v(5) \\ & u(2)/v(2) & & \\ & & u(3)/v(3) & \\ & & 0 & u(4)/v(5) \\ & & & u(5)/v(5) \end{pmatrix} \cdot \begin{pmatrix} 0 \\ v(2) \\ v(3) \\ \frac{\varepsilon}{2} \\ v(5) \end{pmatrix} = \begin{pmatrix} u(1) \\ u(2) \\ u(3) \\ u(4) \\ u(5) \end{pmatrix}.$$

Plan

Data Distribution

Example on 2-level 4-way partitioning with vertex separator

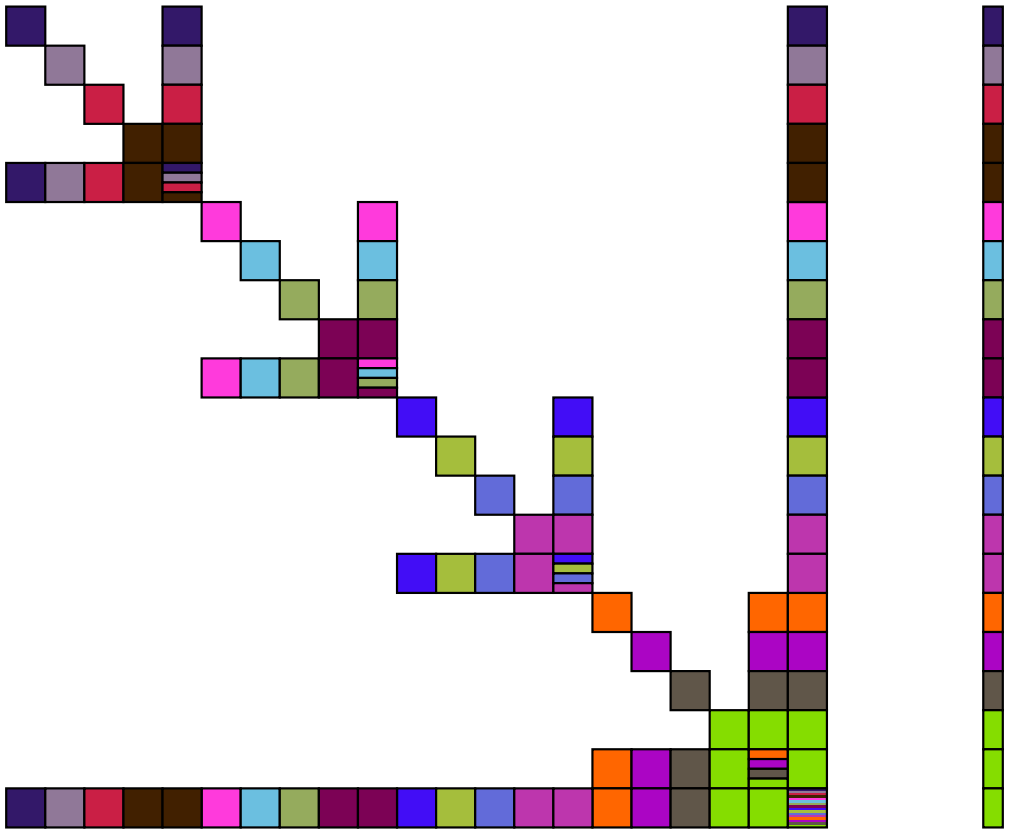


FIGURE : Data distribution of a matrix with a two-level bordered block diagonal structre and a vector involved in the iterative process.

Construction of NFF

$$F_K = D - \sum_{k=1}^{K} L_k \beta_k^{-1} U_k$$

Product $L\beta U$: How to compute it in parallel ?

How to compute $L_k \beta_k^{-1} U_k$ in parallel ?

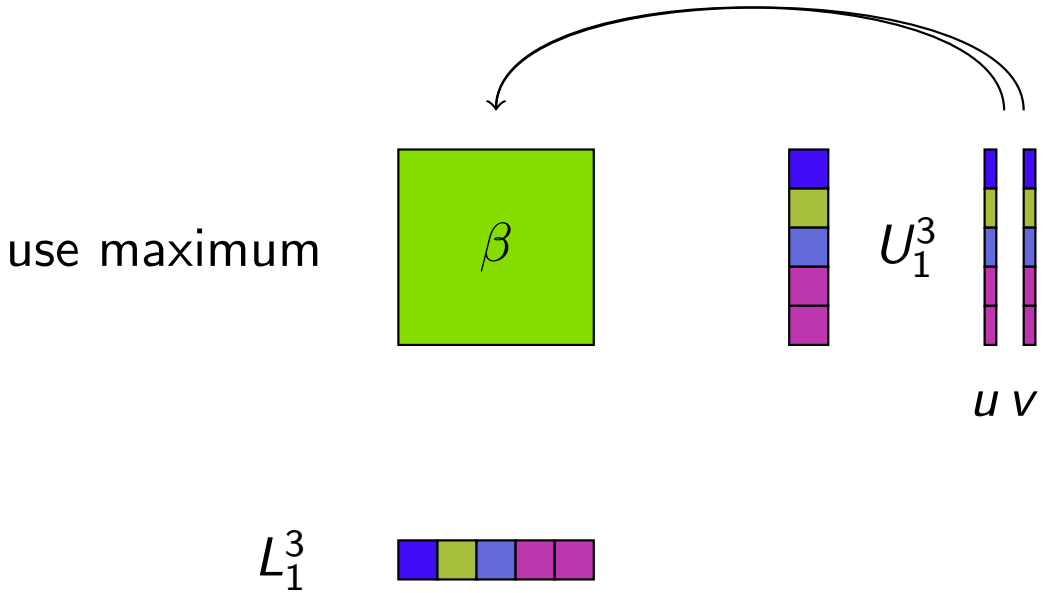
Example :

Blocks L_1^3 ,U_1^3, vectors u,v are all distributed on processors P_9,P_{10},P_{11},P_{12}

- Local $\beta \implies$ No communication between processors during multiplication
- NFF mainly uses original blocks, and only diagonal blocks correspond to separator are modified.

How to compute $L_k \beta_k^{-1} U_k$ in parallel ?

Example :

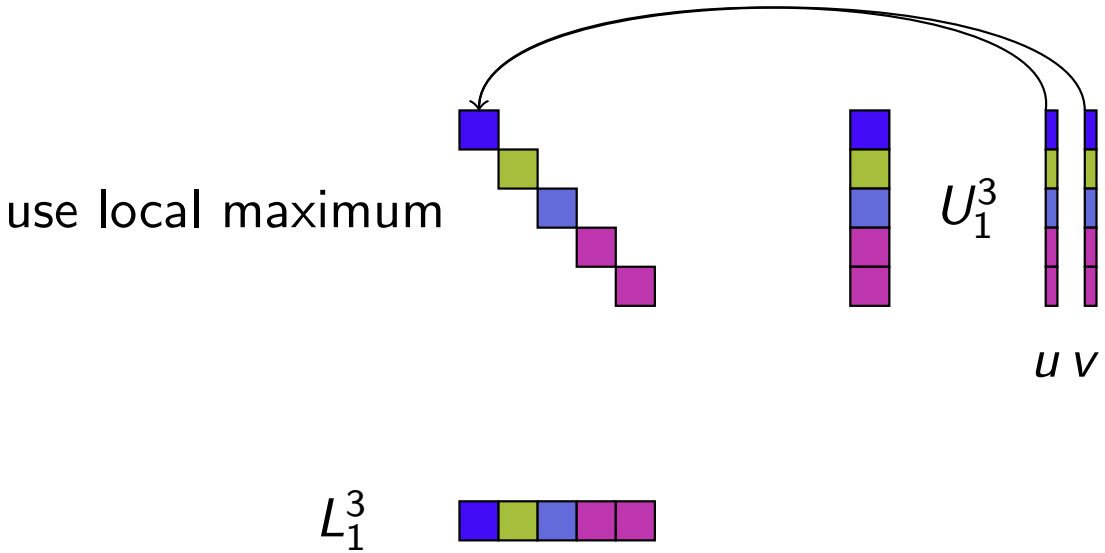


Blocks L_1^3 ,U_1^3, vectors u,v are all distributed on processors P_9,P_{10},P_{11},P_{12}

- Local $\beta \implies$ No communication between processors during multiplication
- NFF mainly uses original blocks, and only diagonal blocks correspond to separator

How to compute $L_k \beta_k^{-1} U_k$ in parallel ?

Example :

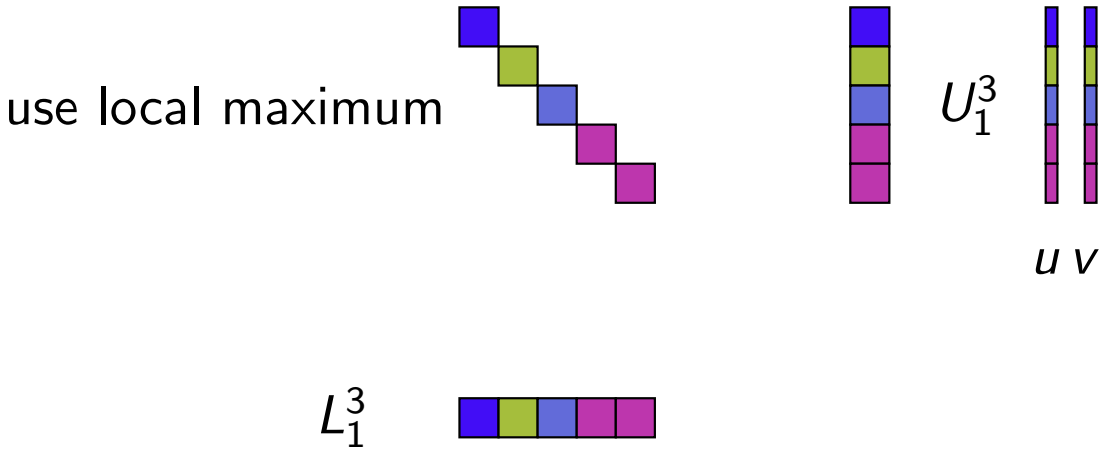


Blocks L_1^3 ,U_1^3, vectors u,v are all distributed on processors P_9,P_{10},P_{11},P_{12}

- Local $\beta \implies$ No communication between processors during multiplication
- NFF mainly uses original blocks, and only diagonal blocks correspond to separator are modified

How to compute $L_k \beta_k^{-1} U_k$ in parallel ?

Example :



Blocks L_1^3 ,U_1^3, vectors u,v are all distributed on processors P_9,P_{10},P_{11},P_{12}

- Local $\beta \implies$ No communication between processors during multiplication
- NFF mainly uses original blocks, and only diagonal blocks correspond to separator are modified.

Application of preconditioner

Applying B_{NFF} requires solving $B_{NFF}x = b$. It can be written as

$$\begin{aligned} B_{NFF} &= \tilde{A} \\ &= (L_1 + \tilde{F}_1)\tilde{F}_1^{-1}(\tilde{F}_1 + U_1) \\ &= (\underbrace{L_1 + \tilde{F}_1}_{\tilde{T}_L}) \cdot (\underbrace{I + \tilde{F}_1^{-1}U_1}_{\tilde{T}_U}), \end{aligned}$$

and can be solved through a forward-backward solve,

$$\tilde{T}_L y = b, \tag{3}$$

$$\tilde{T}_U x = y. \tag{4}$$

After a K-way partitioning with vertex separator (q partitions with separator)

Forward Solve

$$\underbrace{\begin{pmatrix} \tilde{T}_k^1 & & & \\ & \ddots & & \\ & & \tilde{T}_k^q & \\ L_k^1 & \cdots & L_k^q & \tilde{S}_k^1 \end{pmatrix}}_{\tilde{T}_L} \cdot \underbrace{\begin{pmatrix} y^1 \\ \vdots \\ y^q \\ y^s \end{pmatrix}}_{y} = \underbrace{\begin{pmatrix} b^1 \\ \vdots \\ b^q \\ b^s \end{pmatrix}}_{b}. \tag{5}$$

Backward Solve

$$\underbrace{\begin{pmatrix} I & & & (\tilde{T}_k^1)^{-1} U_k^1 \\ & \ddots & & \vdots \\ & & I & (\tilde{T}_k^q)^{-1} U_k^q \\ & & & I \end{pmatrix}}_{\tilde{T}_U} \cdot \underbrace{\begin{pmatrix} x^1 \\ \vdots \\ x^q \\ x^s \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} y^1 \\ \vdots \\ y^q \\ y^s \end{pmatrix}}_{y}. \tag{6}$$

One MPI_Reduce and one MPI_Bcast are required.

Example

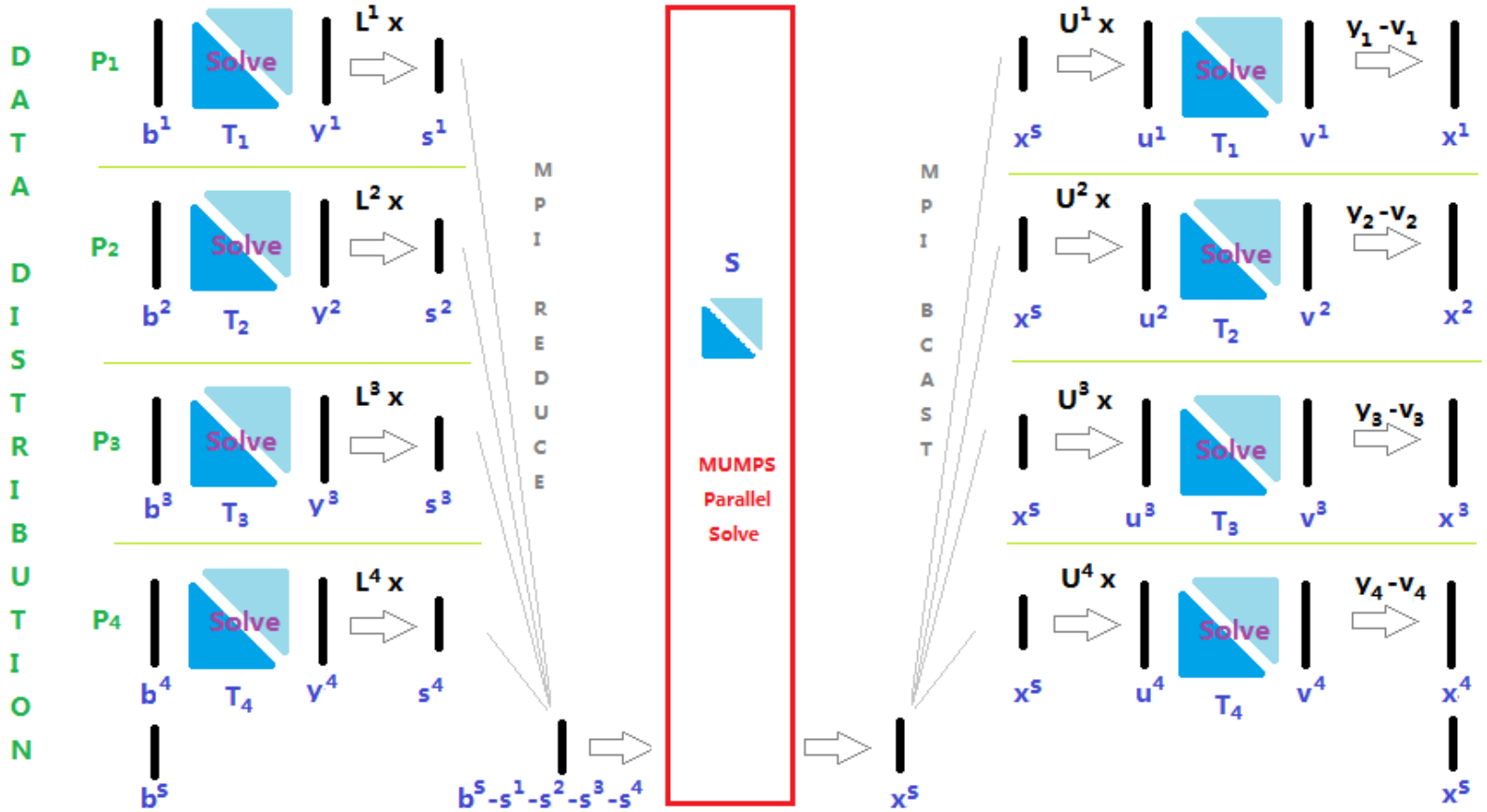
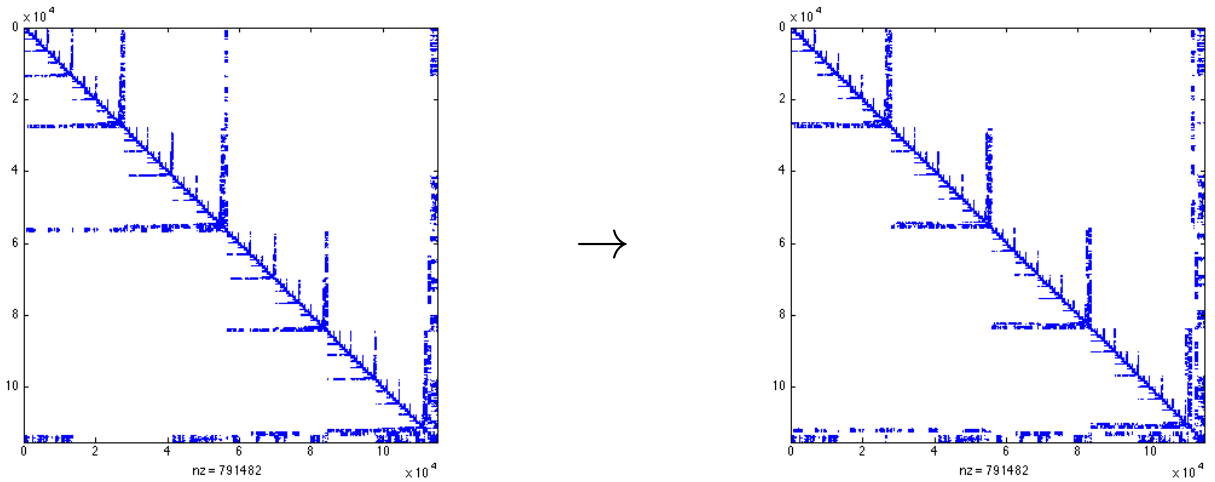


FIGURE : The workflow of solvingprocess for an one-level 4-way partitioning with vertex separator

Implementation

- Use a two-level K-way partitioning with vertex separator
  - to limit the arithmetic complexity
  - a structure is well adapted for hierarchical machines with two levels of parallelism (ex : cluster of multicore processors)
- We perform $\log(P)$ levels of nested dissection to obtain P independent subdomains, and then we group together the separators to obtain the desired separator tree of height two.



Complexity

Analysis based on

- 3D regular grid with nxnxn elements
- partitioned using two-level k-way partitioning with vertex separator
- q_1 and q_2 refer to number of partition in each level ($q = q_1 * q_2$ subdomains in all)
- q_1 and q_2 are both power of 8

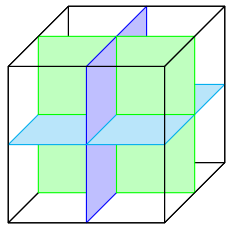


FIGURE : The separator and its neighbours in a simple 3D case.

Complexity - Computational cost

The total computational cost on the critical path is

$$
\begin{aligned}
\#flops \quad \approx \quad & 4 \max_{i=1,\dots P} sol(T_2^i) + 2 \max_{i=1,\dots q_1} sol(S_2^i) + sol(S_1^1) \\
& + 2\left(\frac{24}{q_2} + 3\log q_2\right)\left(\sqrt[3]{q_2} - 1\right)\frac{\sqrt[3]{n^2}}{\sqrt[3]{q_1}} \\
& + \left(\frac{24}{P} + 3\log\left(P\right)\right)\left(\sqrt[3]{q_1} - 1\right)\sqrt[3]{n^2} + 4\frac{n}{P}
\end{aligned}
$$

Complexity - Communication cost

The total communication cost performed on the critical path in terms of volume of communication and number of messages is

$$
\begin{aligned}
\#word &= 12(\sqrt[3]{q_2}-1)\frac{\sqrt[3]{n}}{\sqrt[3]{q_1}}\log q_2 + 2comw(S_1^1) \\
&\quad +6(\sqrt[3]{q_1}-1)\sqrt[3]{n}\log(q_1 q_2) + \max_{i=1,\dots q_1} comw(S_2^i). \\
\#message &= 2\log(q_1 q_2) + 4\log q_2 \\
&\quad +2comm(S_1^1) + \max_{i=1\dots q_1} comm(S_2^i).
\end{aligned}
$$

Plan

Platform

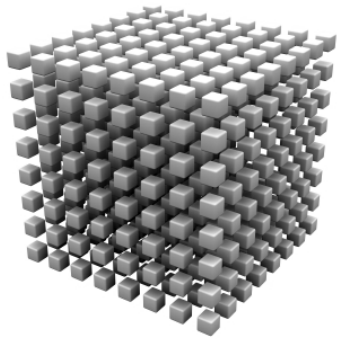
- performed on Curie supercomputer, [1], Intel Xeon Sandy Bridge processors, 2.7GHz,8 cores
- Intel compilers and MKL version 13.1.0.146
- BullxMPI version 1.1.16.5
- MUMPS version 4.10.0
- PETSc version 3.3
- GMRES restarts at 200 iterations, stops when $||r_i|| < 10^{-8}||r_0||$

1. http ://www-hpc.cea.fr/en/complexe/tgcc-curie.htm

Test cases

Boundary value problem - 3D Skyscraper

$$
\begin{aligned}
div(a(x)u) - div(\kappa(x)\nabla u) &= f && \text{in } \Omega \\
u &= 0 && \text{on } \partial\Omega_D \\
\frac{\partial u}{\partial n} &= 0 && \text{on } \partial\Omega_N
\end{aligned}
$$



discretized on a cartesian grid , where

$$
\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1), \textit{if } [10 * x_i] = 0 mod(2),\ i = 1, 2, 3, \\ 1, \qquad\qquad\qquad\qquad\qquad\qquad \textit{otherwise}. \end{cases}
$$

3D Poisson problem

Test cases

- Matrices are generated with FreeFem++,
- Dirichlet boudary condition,
- The domaine is a cube with 225 grid points per edge, (11 millions unkonwns, 70 millions nonzeros)
- 2 threads are used per subdomain, 1 thread per core.

Vector of all ones is used as Filter.

Filtering efficiency

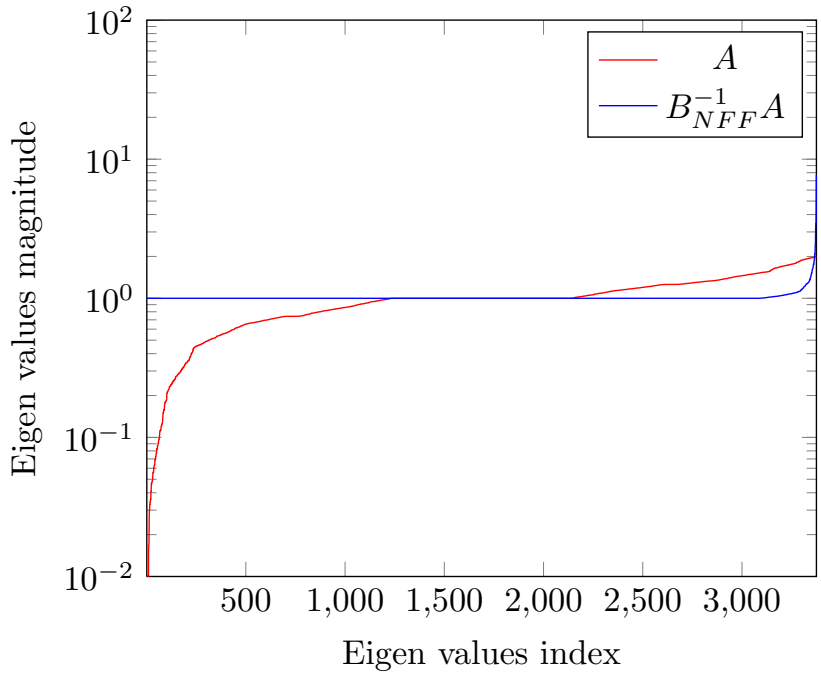


FIGURE : Spectrum of matrix 3DSKY15x15x15.

Comparaison of Convergence Rate

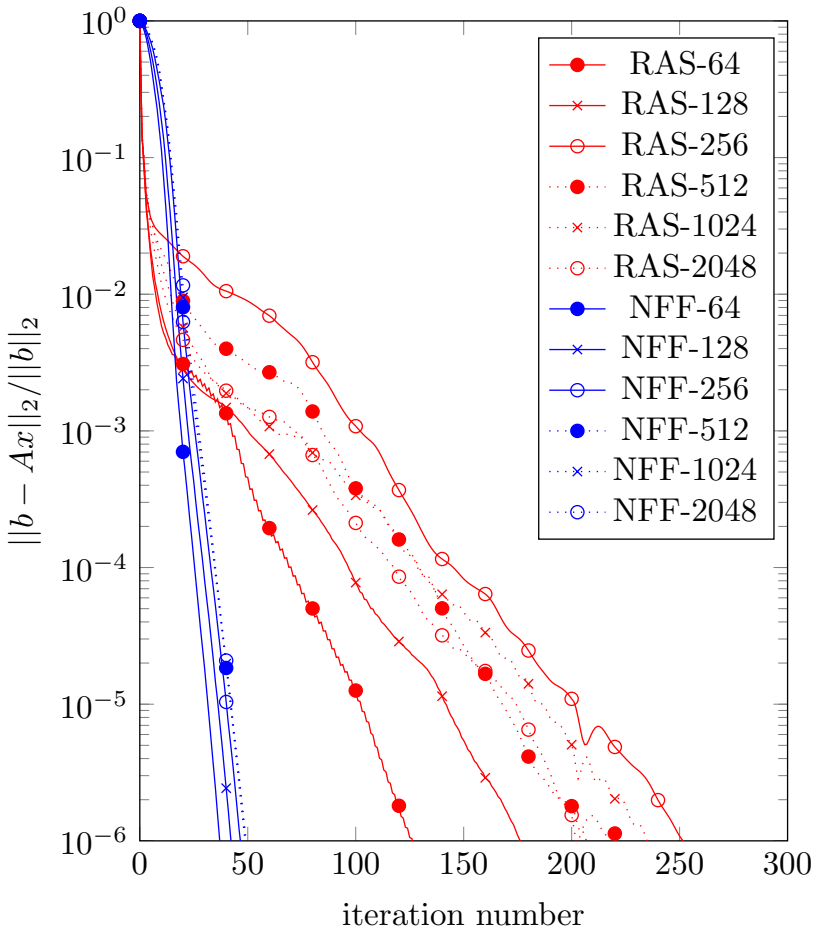


FIGURE : 3DPOI225x225x225

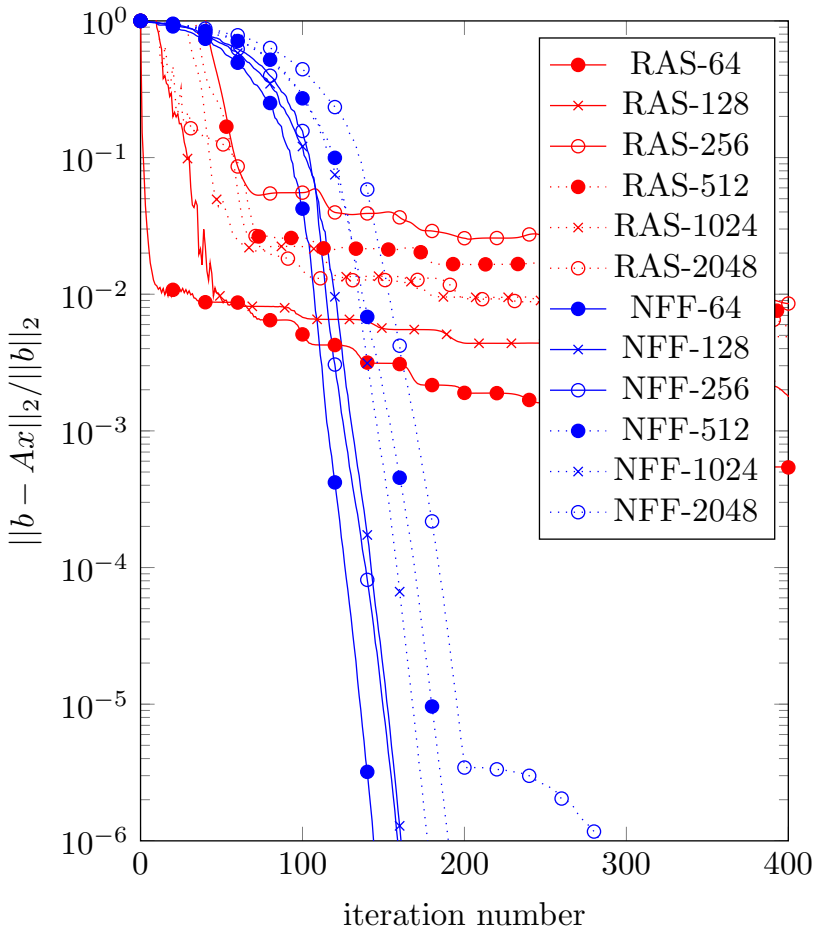


FIGURE : 3D3KY225x225x225

Elapsed Time for building and applying NFF

TABLE : Elapsed time in seconds for NFF using PARDISO on 3DSKY225x225x225

#Core	Partition	Compute NFF preconditioner						Apply NFF preconditioner (per iter)				
		(T_2^i)	(S_2^i)	(S_1^1)	Reduce	$L\beta U$	total	(T_2^i)	(S_2^i)	(S_1^1)	comm	total
64	8x4	135	0.45	0.66	0.16	0.09	139.2	3.39	0.05	0.03	0.005	3.46
128	8x8	30.26	0.25	1.16	0.22	0.06	32.89	1.14	0.02	0.04	0.005	1.21
256	16x8	5.9	0.38	1.16	0.32	0.05	8.69	0.38	0.04	0.04	0.01	0.53
512	16x16	2.5	0.22	2.24	0.7	0.39	6.65	0.13	0.02	0.08	0.02	0.39
1024	32x16	2.34	0.33	2.2	0.87	0.52	7.89	0.05	0.04	0.08	0.02	0.34
2048	32x32	4.49	0.19	4.02	4.42	1.32	16.4	0.01	0.02	0.13	0.02	0.35

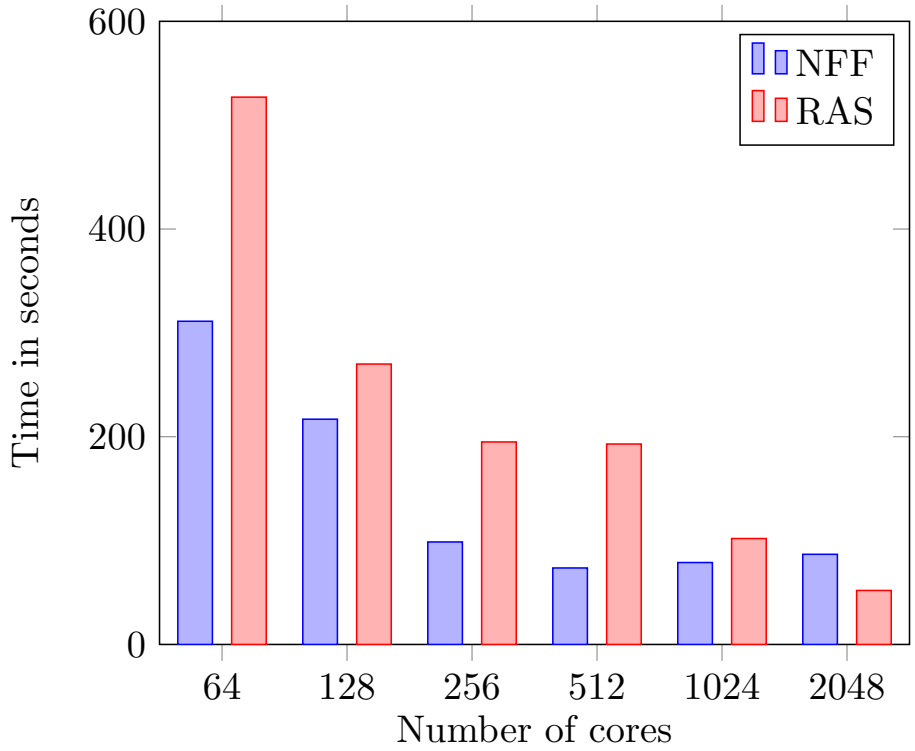
TABLE : Elapsed time in seconds for NFF using PARDISO and MUMPS on 3DSKY225x225x225

#Core	Partition	Compute NFF preconditioner						Apply NFF preconditioner (per iter)				
		(T_2^i)	(S_2^i)	(S_1^1)	Reduce	$L\beta U$	total	(T_2^i)	(S_2^i)	(S_1^1)	comm	total
64	8x4	127.5	0.72	1.36	0.19	0.11	132.52	3.39	0.04	0.02	0.005	3.47
128	8x8	29.65	0.42	1.95	0.23	0.09	33.75	1.21	0.0008	0.046	0.014	1.27
256	16x8	6.07	0.68	1.98	1.07	0.85	11.23	0.4	0.04	0.03	0.04	0.6
512	16x16	2.65	0.37	3.13	1.3	0.87	19.83	0.14	0.03	0.06	0.06	0.44
1024	32x16	2.3	0.54	3.2	1.16	0.36	10.24	0.05	0.027	0.051	0.09	0.38
2048	32x32	4.44	0.3	4.47	4.73	1.02	17.69	0.02	0.02	0.09	0.03	0.33

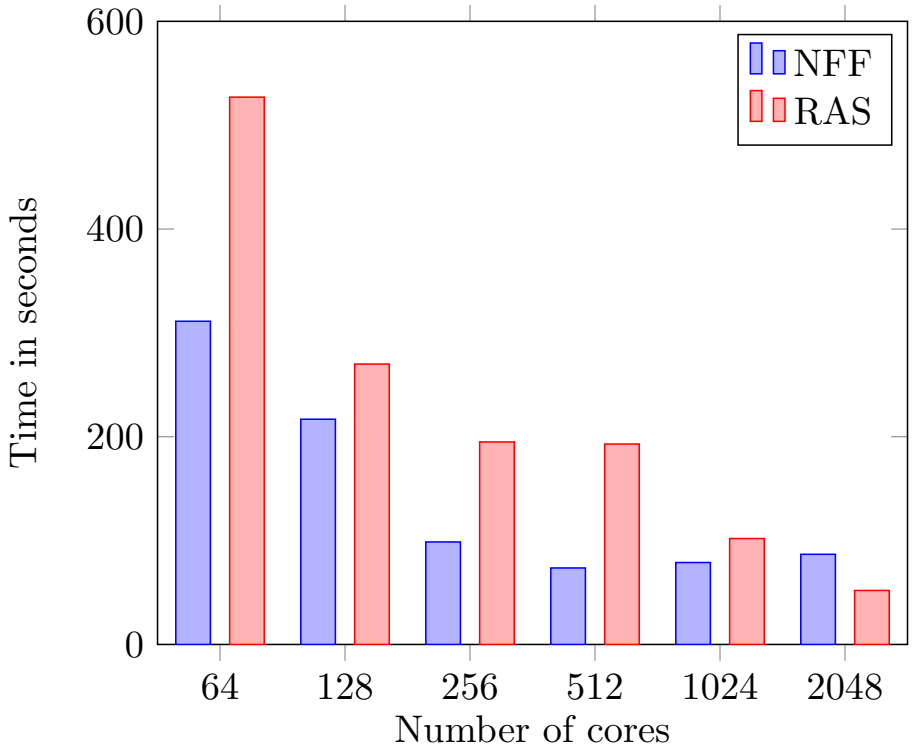
Comparaison between RAS and NFF

For 3DSKY225x225x225 :

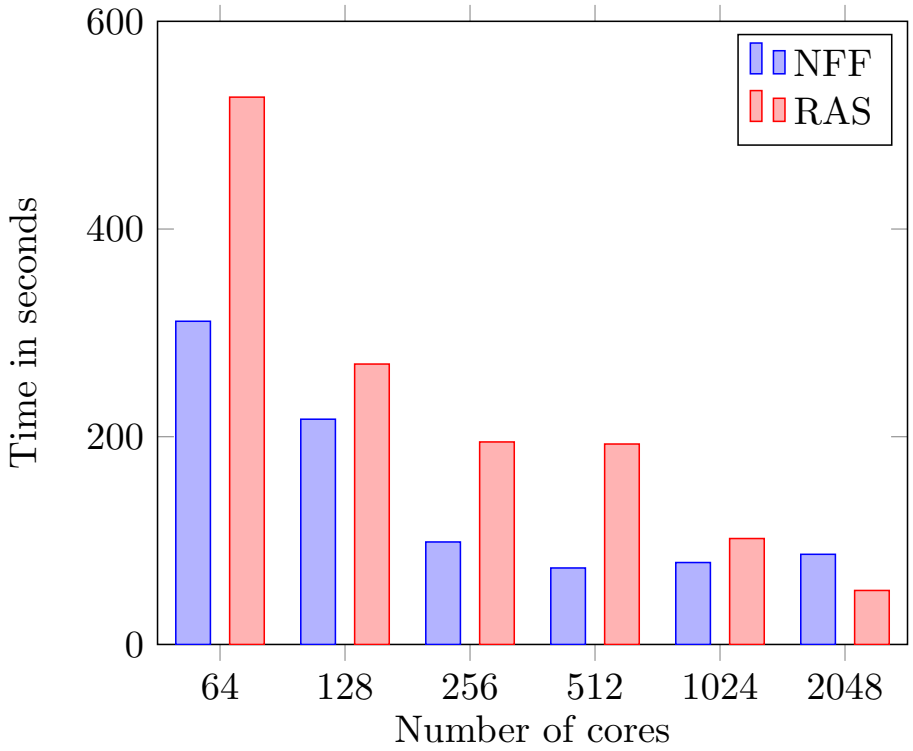
	RAS		NFF	
# subdomains	iteration	error	iteration	error
64	889	2.66e-07	50	2.69e-07
128	1639	3.00e-07	148	3.15e-07
256	2546	5.27e-07	162	7.94e-07
512	4664	6.20e-07	160	8.31e-07
1024	4567	2.07e-07	191	7.28e-07
2048	3564	1.49e-07	179	5.82e-07



- For 512 partitions (1024 cores) :
  - the size of the separator becomes bigger than that of subdomains
  - ($\Longrightarrow$ test with bigger matrices).
  - load balancing, natural k-way partitioning with vertex separator routine
- Elapsed time : 1 iteration of NFF >> that of RAS
  - Direct solver doesn't profit enough multicore. (Spent a lot of time on nested dissection routine used during the initialisation phrase.)



- For 512 partitions (1024 cores) :
  - the size of the separator becomes bigger than that of subdomains
  - ($\Longrightarrow$ test with bigger matrices).
  - load balancing, natural k-way partitioning with vertex separator routine
- Elapsed time : 1 iteration of NFF >> that of RAS
  - Direct solver doesn't profit enough multicore. (Spent a lot of time on nested dissection routine used during the initialisation phrase.)



- For 512 partitions (1024 cores) :
  - the size of the separator becomes bigger than that of subdomains
  - ($\Longrightarrow$ test with bigger matrices).
  - load balancing, natural k-way partitioning with vertex separator routine
- Elapsed time : 1 iteration of NFF >> that of RAS
  - Direct solver doesn't profit enough multicore. (Spent a lot of time on nested dissection routine used during the initialisation phrase.)

Plan

Conclusion

NFF preconditioner

- For a given matrix A and a filtering vector t, satisfies the property $B_{NFF}\,t = At$
- can be computed in parallel since using nested K-way partitioning with vertex separator
- based on original blocks in A.
  - $\Longrightarrow$ only diagonal blocks are modified
  - $\Longrightarrow$ small memory usage (only factors of diagonal blocks are stored)

Inconvenient

- The size of the separator may become quite big compares to that of subdomains, load balancing.
  - For matrices from 3D Skycrapper with 255x255x255 unknowns, NFF is scalable up to 512 cores.
  - + synchronization requires during the application of NFF.

Conclusion

NFF preconditioner

- For a given matrix A and a filtering vector t, satisfies the property $B_{NFF}\,t = At$
- can be computed in parallel since using nested K-way partitioning with vertex separator
- based on original blocks in A.
  - $\Longrightarrow$ only diagonal blocks are modified
  - $\Longrightarrow$ small memory usage (only factors of diagonal blocks are stored)

Inconvenient

- The size of the separator may become quite big compares to that of subdomains, load balancing.
  - For matrices from 3D Skycrapper with 255x255x255 unknowns, NFF is scalable up to 512 cores.
  - + synchronization requires during the application of NFF.

Future Work

Future work

- Build β which filters many vectors using low rank approximation to get a better convergence rate.
  - i.e. Add filtering on high frequency which may remove the plateau in the convergence curve.
- Test on other unstructured matrix. Which filtering vector we should use ?
- Optimization
  - a better configuration of direct solver used in inner most diagonal blocks
  - a better partitioning routine ?

Thank you

Thank Your For Your Attention !

References (1)


Grigori, L., Kumar, P., Nataf, F., and Wang, K. (2010).
A class of multilevel parallel preconditioning strategies.
Research Report RR-7410, INRIA.


Grigori, L., Nataf, F., and Fezzani, R. (submitted 2012).
Block filtering decomposition.
In *Numerical Linear Algebra with Applications (NLAA)*.