VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

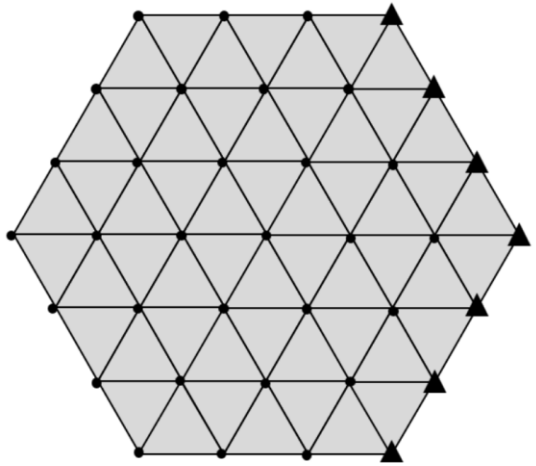
GPU acceleration of a FETI solver

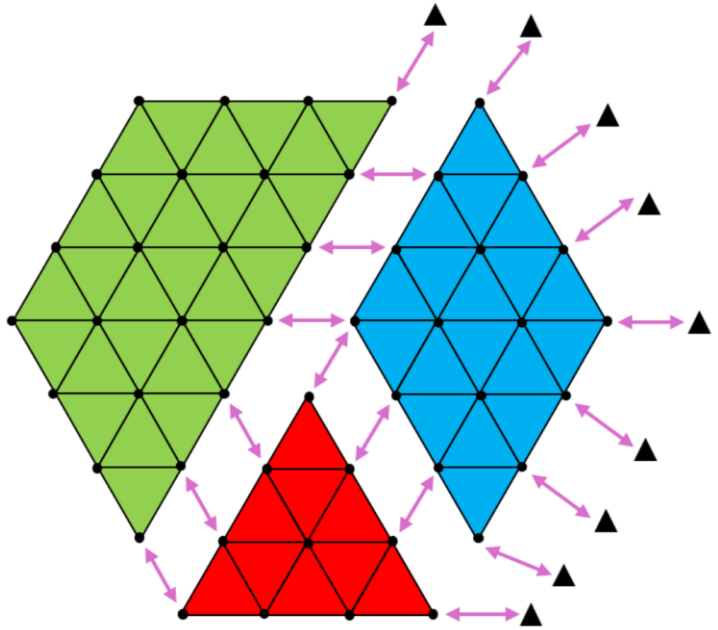
Jakub Homola
IT4Innovations, VSB-TUO

2026-05-19

FETI – Finite Element Tearing and Interconnecting

- Finite Element Method (FEM)
- Domain decomposition





$$\mathbf{Ku} = \mathbf{f}$$

$$\begin{bmatrix} \mathbf{K}_1 & & & & \mathbf{B}_1^\top \\ & \mathbf{K}_2 & & & \mathbf{B}_2^\top \\ & & \ddots & & \vdots \\ & & & \mathbf{K}_N & \mathbf{B}_N^\top \\ \mathbf{B}_1 & \mathbf{B}_2 & \dots & \mathbf{B}_N & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_N \\ \mathbf{c} \end{bmatrix}$$

FETI – Finite Element Tearing and Interconnecting

Algorithm 2: The Preconditioned Conjugate Projected Gradient method

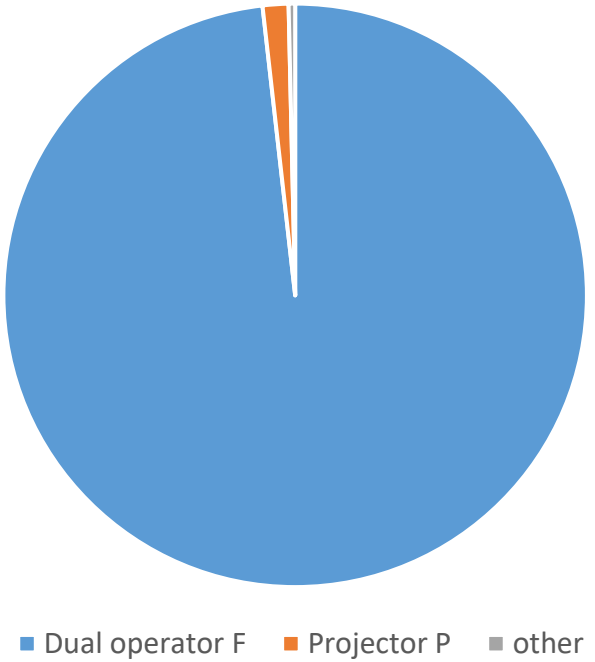
$\boldsymbol{\lambda}_0 \leftarrow \boldsymbol{\lambda}_I$
$\mathbf{r}_0 \leftarrow \mathbf{d} - \mathbf{F}\boldsymbol{\lambda}_I$
$\mathbf{w}_0 \leftarrow \mathbf{P}\mathbf{r}_0$
$\mathbf{y}_0 \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_0$
$\mathbf{p}_0 \leftarrow \mathbf{y}_0$
for $k = 0, 1, 2, \ldots$ *until convergence* **do**
  $\mathbf{q}_k \leftarrow \mathbf{F}\mathbf{p}_k$
  $\delta_k \leftarrow (\mathbf{w}_k^\top \mathbf{y}_k)/(\mathbf{p}_k^\top \mathbf{q}_k)$
  $\boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \delta_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \delta_k \mathbf{q}_k$
  $\mathbf{w}_{k+1} \leftarrow \mathbf{P}\mathbf{r}_{k+1}$
  $\mathbf{y}_{k+1} \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_{k+1}$
  $\beta_k \leftarrow (\mathbf{w}_{k+1}^\top \mathbf{y}_{k+1})/(\mathbf{w}_k^\top \mathbf{y}_k)$
  $\mathbf{p}_{k+1} \leftarrow \mathbf{y}_{k+1} + \beta_k \mathbf{p}_k$

Bottleneck in FETI

Algorithm 2: The Preconditioned Conjugate Projected Gradient method

1 $\boldsymbol{\lambda}_0 \leftarrow \boldsymbol{\lambda}_I$
2 $\mathbf{r}_0 \leftarrow \mathbf{d} - \mathbf{F}\boldsymbol{\lambda}_I$
3 $\mathbf{w}_0 \leftarrow \mathbf{P}\mathbf{r}_0$
4 $\mathbf{y}_0 \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_0$
5 $\mathbf{p}_0 \leftarrow \mathbf{y}_0$
6 **for** $k = 0, 1, 2, \ldots$ *until convergence* **do**
7 $\quad \mathbf{q}_k \leftarrow \mathbf{F}\mathbf{p}_k$
8 $\quad \delta_k \leftarrow (\mathbf{w}_k^\top \mathbf{y}_k)/(\mathbf{p}_k^\top \mathbf{q}_k)$
9 $\quad \boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \delta_k \mathbf{p}_k$
10 $\quad \mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \delta_k \mathbf{q}_k$
11 $\quad \mathbf{w}_{k+1} \leftarrow \mathbf{P}\mathbf{r}_{k+1}$
12 $\quad \mathbf{y}_{k+1} \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_{k+1}$
13 $\quad \beta_k \leftarrow (\mathbf{w}_{k+1}^\top \mathbf{y}_{k+1})/(\mathbf{w}_k^\top \mathbf{y}_k)$
14 $\quad \mathbf{p}_{k+1} \leftarrow \mathbf{y}_{k+1} + \beta_k \mathbf{p}_k$

Fraction of solver runtime



FETI dual operator

- Dual operator $\mathbf{F}$
- Applied in each iteration of PCPG

- Per-subdomain local dual operator $\mathbf{F}_i$

$$\mathbf{F}_i = \mathbf{B}_i \mathbf{K}_i^{-1} \mathbf{B}_i^\top$$

- = negative of the Schur complement of matrix

$$\begin{bmatrix} \mathbf{K}_i & \mathbf{B}_i^\top \\ \mathbf{B}_i & \mathbf{O} \end{bmatrix}$$

Preconditioned Conjugate Projected Gradient method

1 $\boldsymbol{\lambda}_0 \leftarrow \boldsymbol{\lambda}_I$
2 $\mathbf{r}_0 \leftarrow \mathbf{d} - \mathbf{F}\boldsymbol{\lambda}_I$
3 $\mathbf{w}_0 \leftarrow \mathbf{P}\mathbf{r}_0$
4 $\mathbf{y}_0 \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_0$
5 $\mathbf{p}_0 \leftarrow \mathbf{y}_0$
6 **for** $k = 0, 1, 2, \ldots$ **do**
7 $\quad \mathbf{q}_k \leftarrow \mathbf{F}\mathbf{p}_k$
8 $\quad \delta_k \leftarrow (\mathbf{w}_k^\top \mathbf{y}_k)/(\mathbf{p}_k^\top \mathbf{q}_k)$
9 $\quad \boldsymbol{\lambda}_{k+1} \leftarrow \boldsymbol{\lambda}_k + \delta_k \mathbf{p}_k$
10 $\quad \mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \delta_k \mathbf{q}_k$
11 $\quad \mathbf{w}_{k+1} \leftarrow \mathbf{P}\mathbf{r}_{k+1}$
12 $\quad \mathbf{y}_{k+1} \leftarrow \mathbf{P}\mathbf{M}\mathbf{w}_{k+1}$
13 $\quad \beta_k \leftarrow (\mathbf{w}_{k+1}^\top \mathbf{y}_{k+1})/(\mathbf{w}_k^\top \mathbf{y}_k)$
14 $\quad \mathbf{p}_{k+1} \leftarrow \mathbf{y}_{k+1} + \beta_k \mathbf{p}_k$

Application of the local dual operator

- Implicit

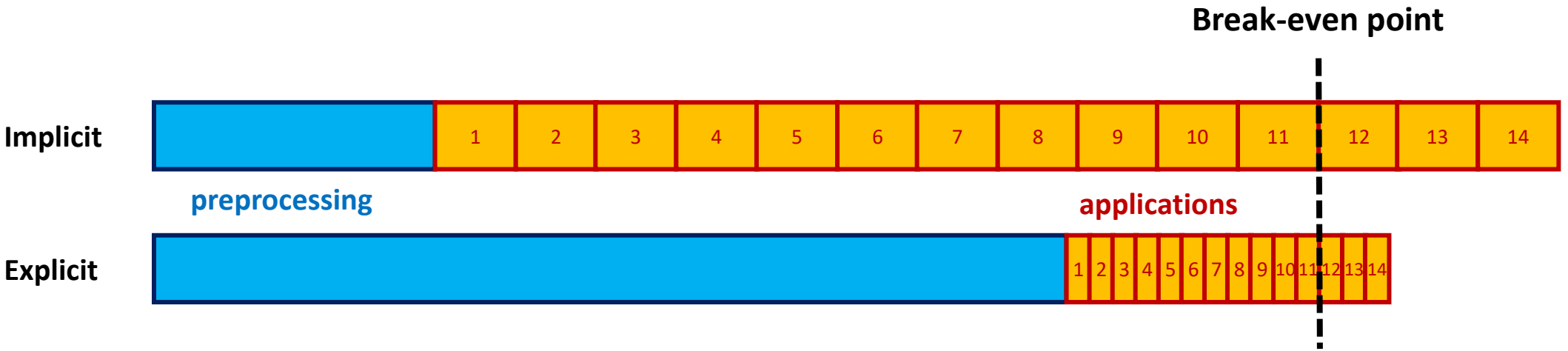
$$\mathbf{F}_i\mathbf{p}_i = \mathbf{B}_i(\mathbf{K}_i^{-1}(\mathbf{B}_i^\top\mathbf{p}_i))$$

  - Fast preprocessing, slow iterations

- Explicit

$$\mathbf{F}_i\mathbf{p}_i = (\mathbf{B}_i\mathbf{K}_i^{-1}\mathbf{B}_i^\top)\mathbf{p}_i$$

  - Slow preprocessing, fast iterations



Application of the local dual operator

- Implicit

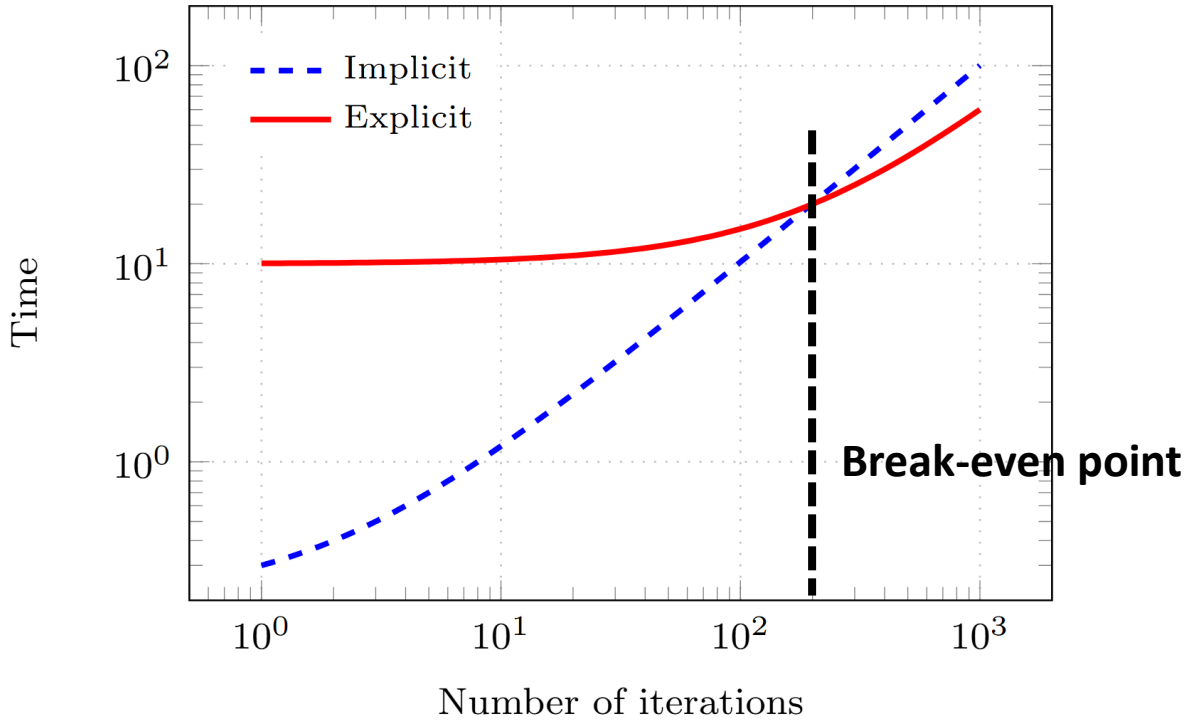
$$\mathbf{F}_i\mathbf{p}_i = \mathbf{B}_i(\mathbf{K}_i^{-1}(\mathbf{B}_i^\top \mathbf{p}_i))$$

  - Fast preprocessing, slow iterations

- Explicit

$$\mathbf{F}_i\mathbf{p}_i = (\mathbf{B}_i\mathbf{K}_i^{-1}\mathbf{B}_i^\top)\mathbf{p}_i$$

  - Slow preprocessing, fast iterations

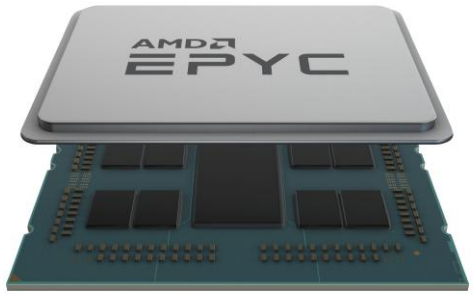


GPU-acceleration of the dual operator

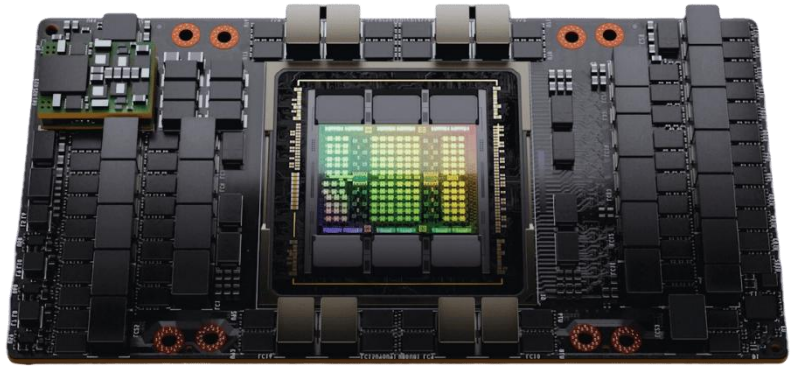
- Explicit approach
  - Application = dense matrix-vector multiplication, good for GPUs

- Previous GPU-acceleration attempts:

CPU



GPU

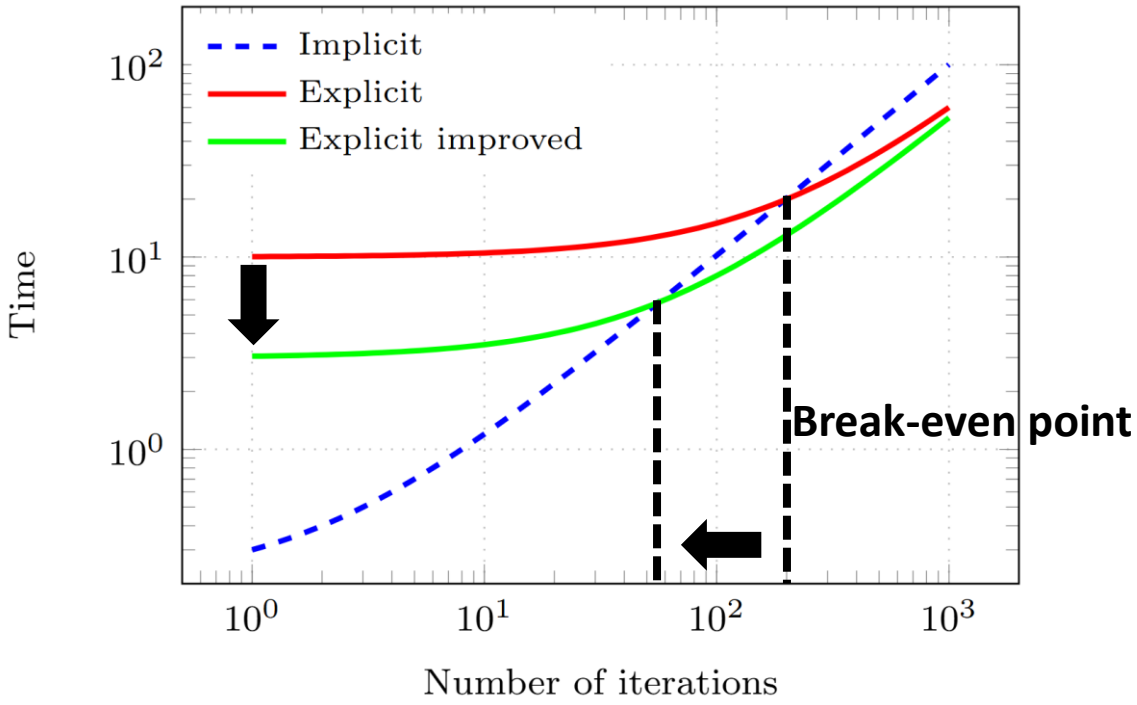


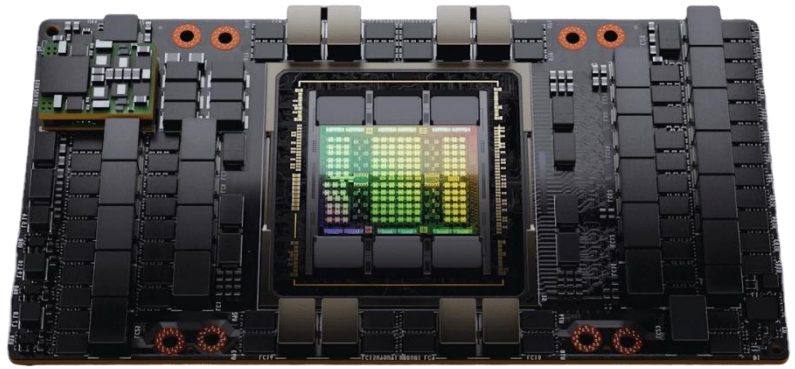
$$\mathbf{F}_i = \mathbf{B}_i \mathbf{K}_i^{-1} \mathbf{B}_i^\top$$

$$\mathbf{q}_i = \mathbf{F}_i \mathbf{p}_i$$

Goals

- Speed up preprocessing
- Assemble $\mathbf{F}_i$ on the GPU





$$\mathbf{F}_i = \mathbf{B}_i \mathbf{K}_i^{-1} \mathbf{B}_i^\top$$

$$\mathbf{q}_i = \mathbf{F}_i \mathbf{p}_i$$

Local dual operator assembly using GPU

$$\begin{aligned}\mathbf{F}_i &= \mathbf{B}_i \mathbf{K}_{i,reg}^{-1} \mathbf{B}_i^\top \\ &= \mathbf{B}_i (\mathbf{L}_i \mathbf{L}_i^\top)^{-1} \mathbf{B}_i^\top \\ &= \mathbf{B}_i \mathbf{L}_i^{-\top} \mathbf{L}_i^{-1} \mathbf{B}_i^\top\end{aligned}$$

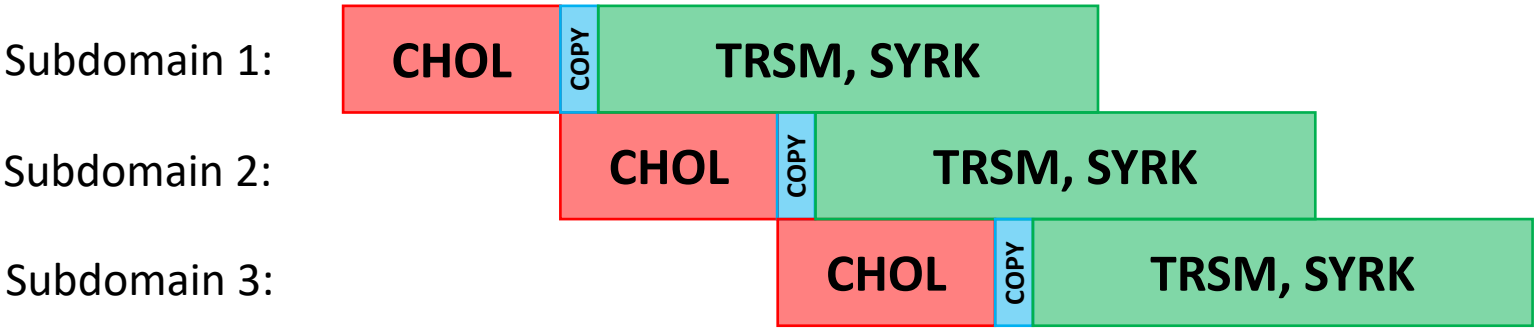
$$\mathbf{F}_i = (\mathbf{L}_i^{-1} \mathbf{B}_i^\top)^\top (\mathbf{L}_i^{-1} \mathbf{B}_i^\top)$$

Local dual operator assembly using GPU

$$\mathbf{F}_i = (\mathbf{L}_i^{-1}\mathbf{B}_i^\top)^\top(\mathbf{L}_i^{-1}\mathbf{B}_i^\top)$$

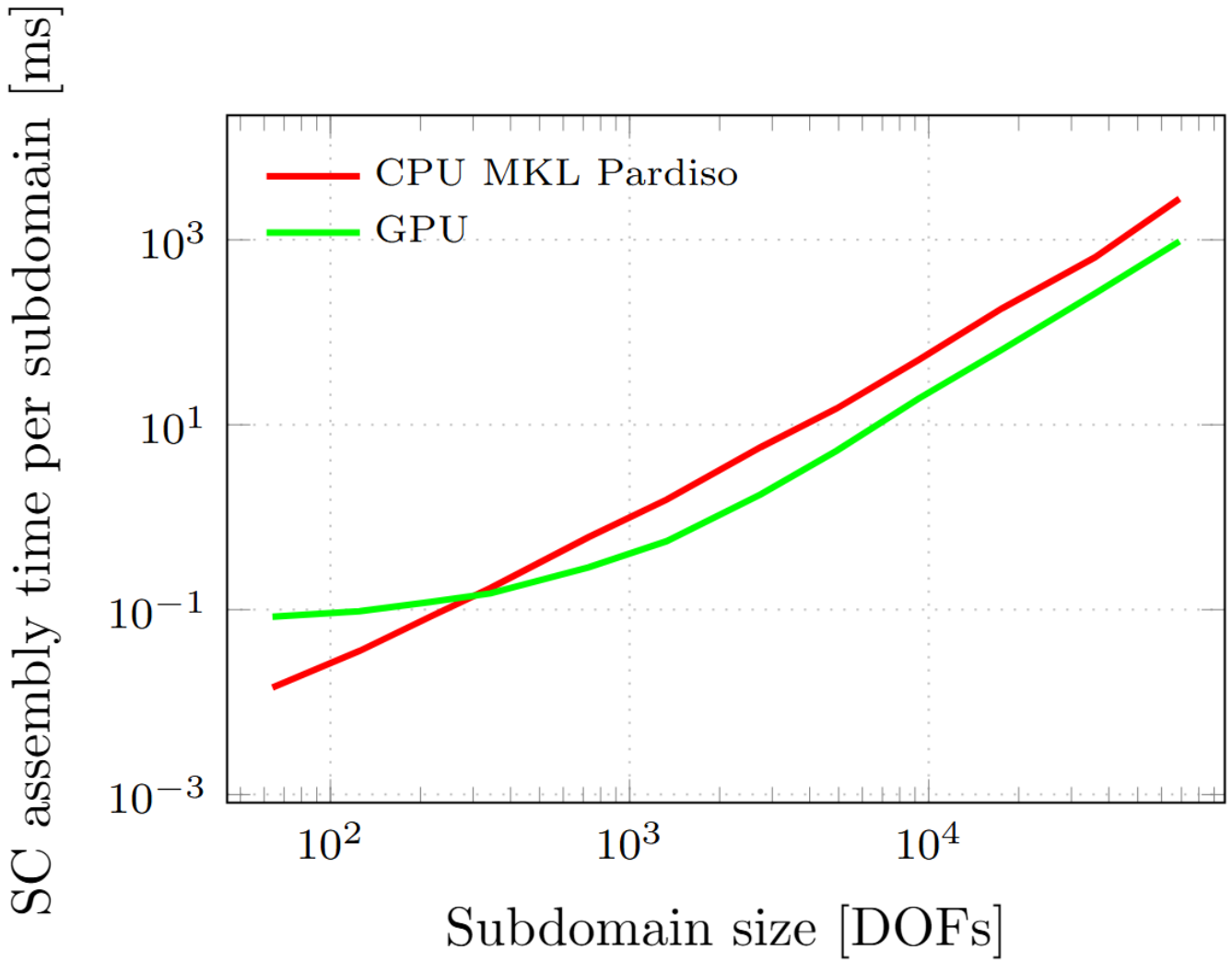
For each subdomain i:

1. Cholesky factorization on CPU $\mathbf{K}_{i,reg} = \mathbf{L}_i\mathbf{L}_i^\top$ (CHOLMOD library)
2. Copy matrices $\mathbf{L}_i$ and $\mathbf{B}_i$ to GPU
3. Call TRSM kernel on GPU $\mathbf{X}_i = \mathbf{L}_i^{-1}\mathbf{B}_i^\top$ (cuBLAS, cuSPARSE)
4. Call SYRK kernel on GPU $\mathbf{F}_i = \mathbf{X}_i^\top\mathbf{X}_i$

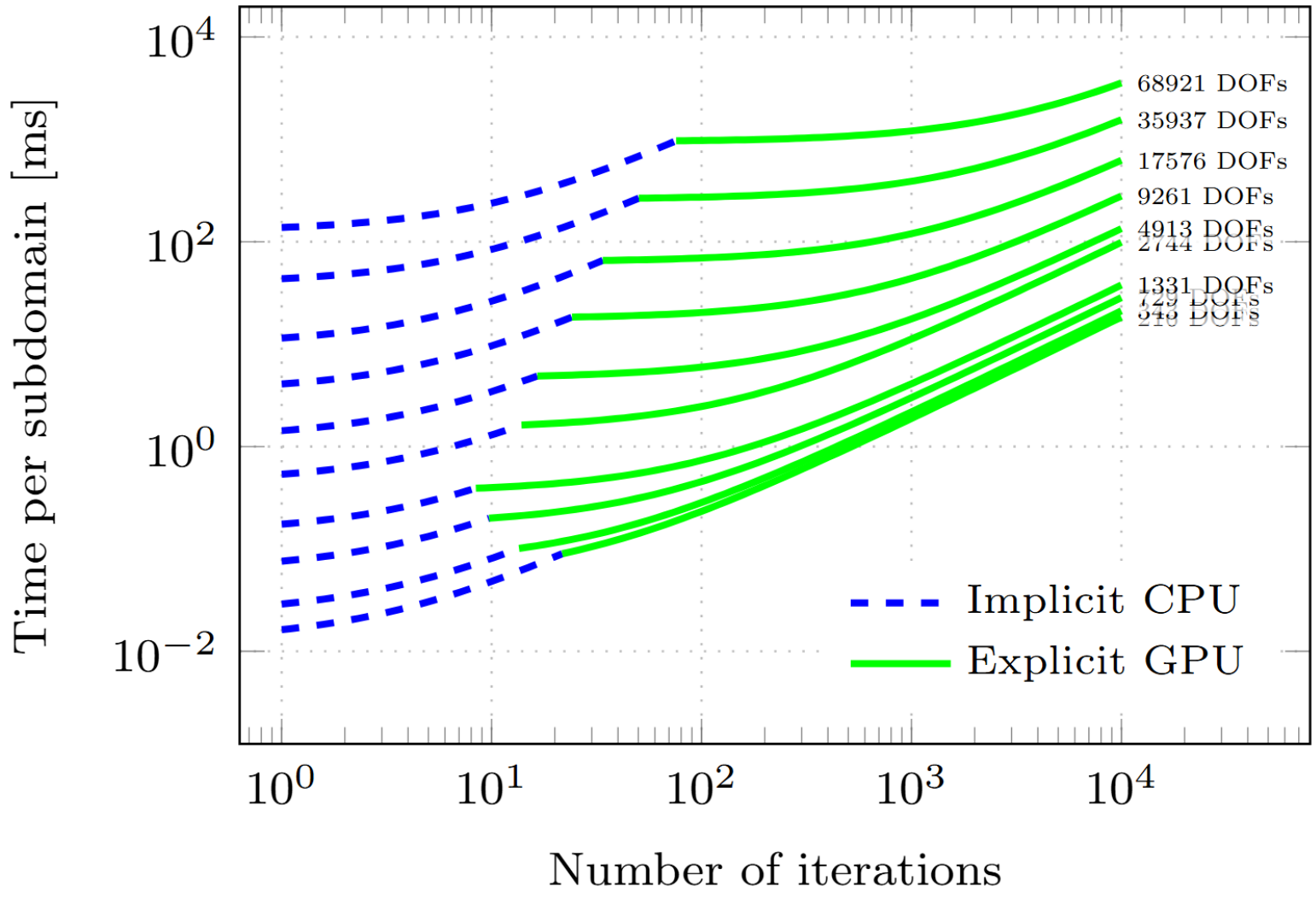


Results: preprocessing time

Karolina
1/8th of GPU node
16 cores of AMD EPYC 7763
NVIDIA A100



Results: break-even point

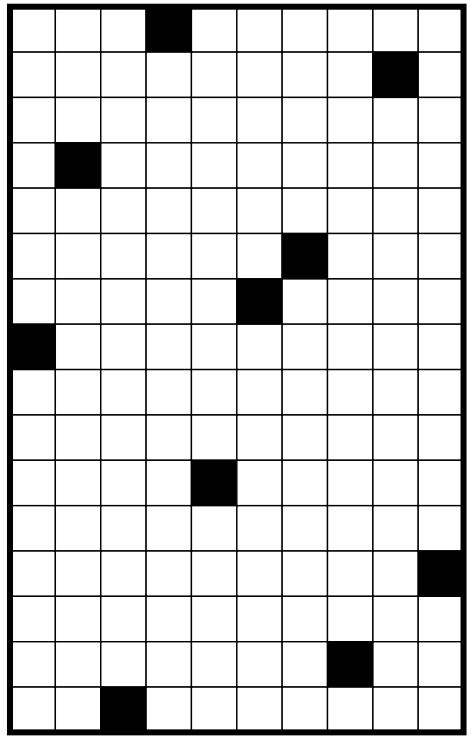


Can we do better?

- Yes

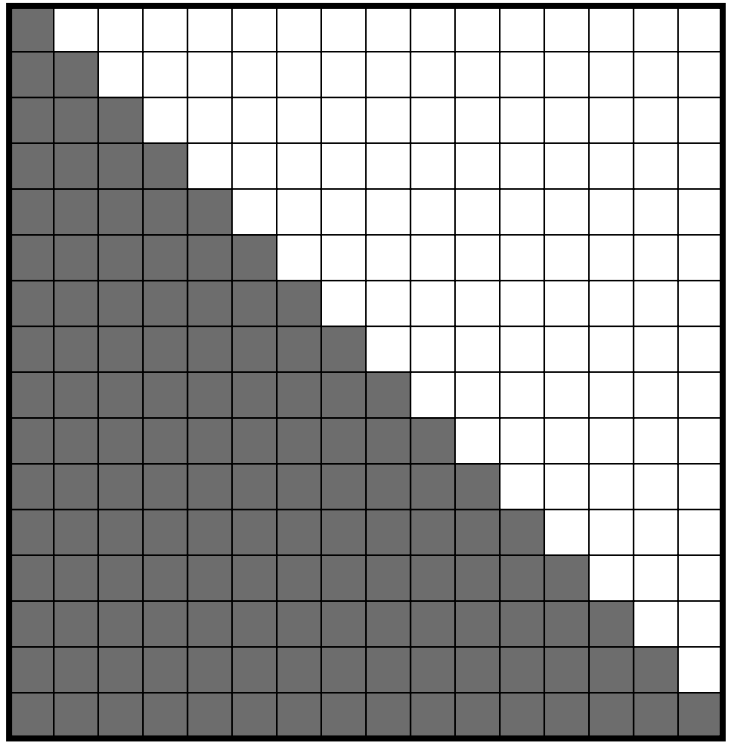
- TRSM and SYRK are still bottlenecks
- $\mathbf{B}_i$ is sparse but not treated as such

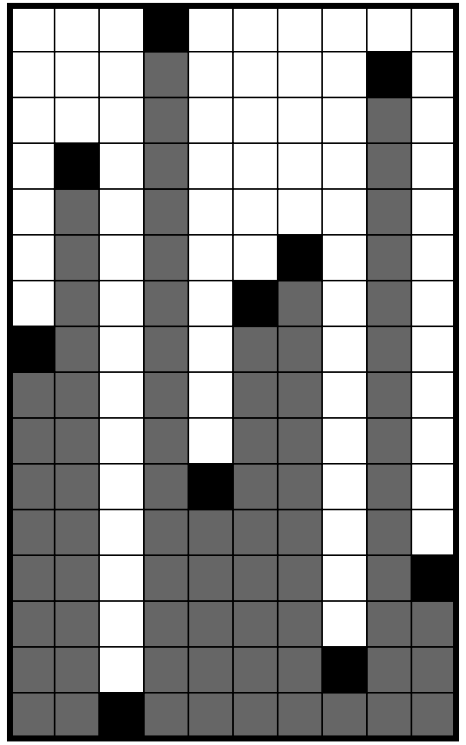
$$\mathbf{B}_i^\top$$

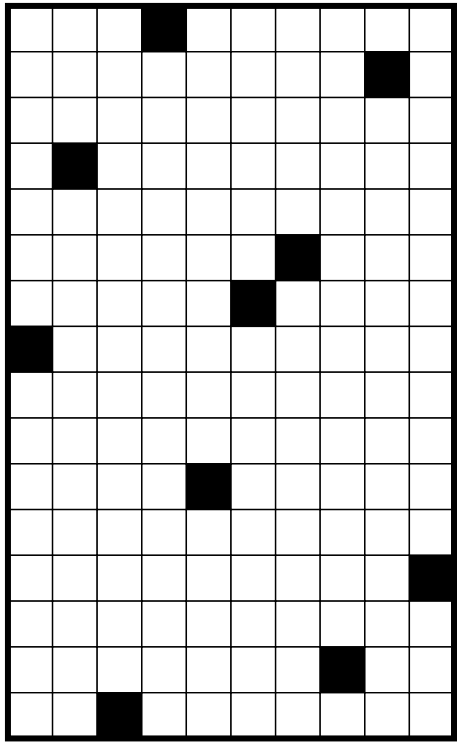


TRSM optimization towards sparsity

$$\mathbf{L}_i \qquad \mathbf{X}_i \qquad = \qquad \mathbf{B}_i^\top$$

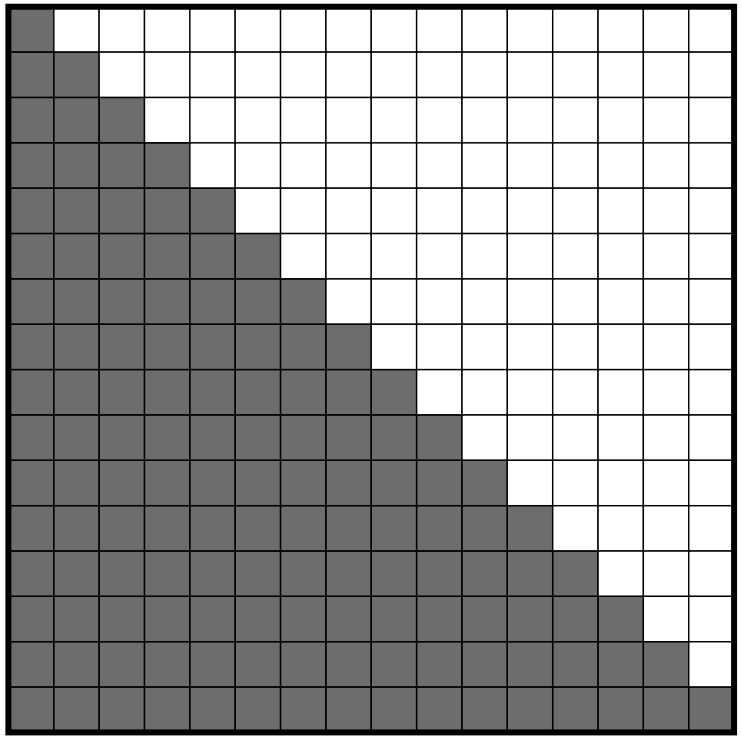


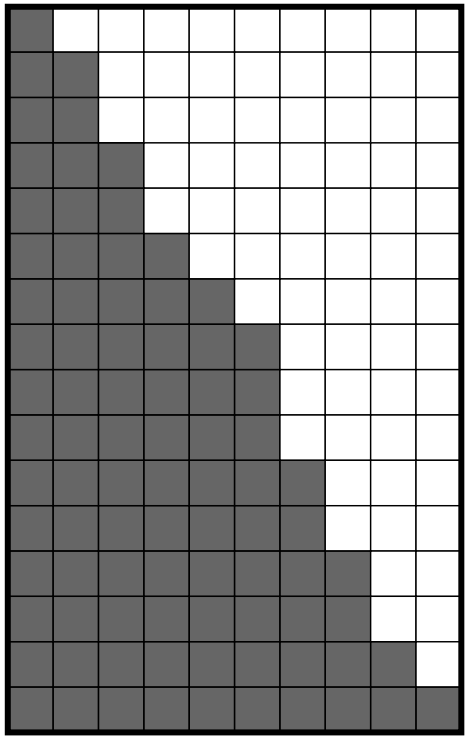


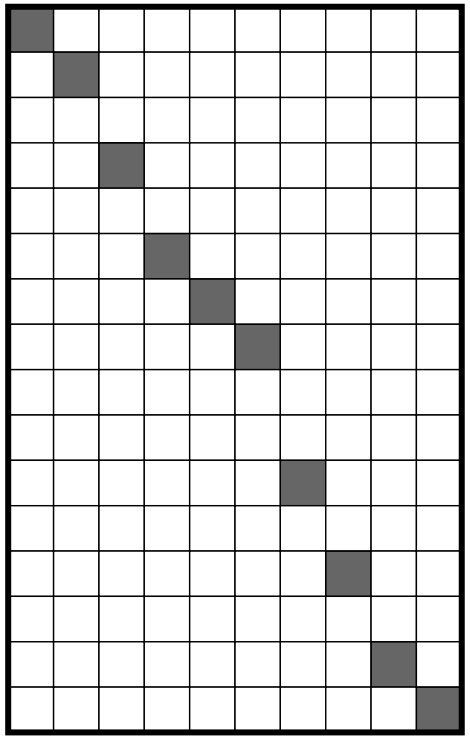


TRSM optimization towards sparsity

$$\mathbf{L}_i \qquad \mathbf{X}_i \qquad = \qquad \mathbf{B}_i^\top$$



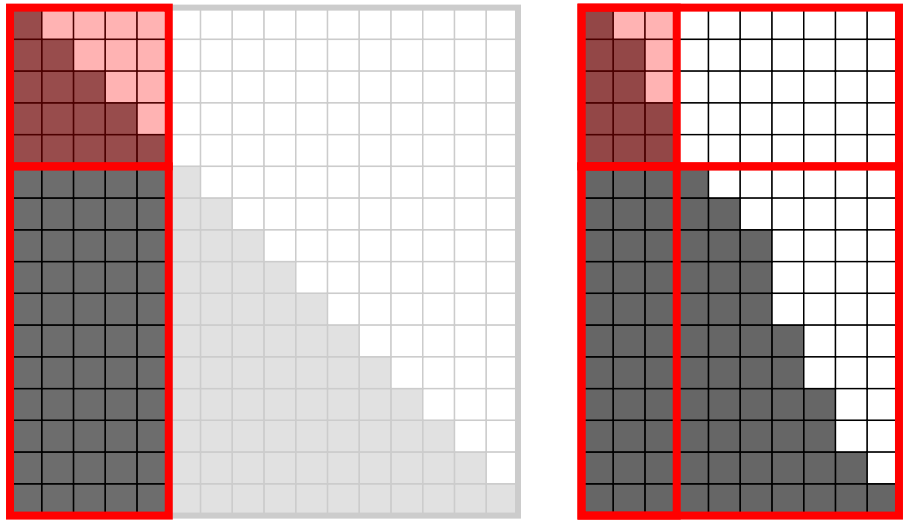




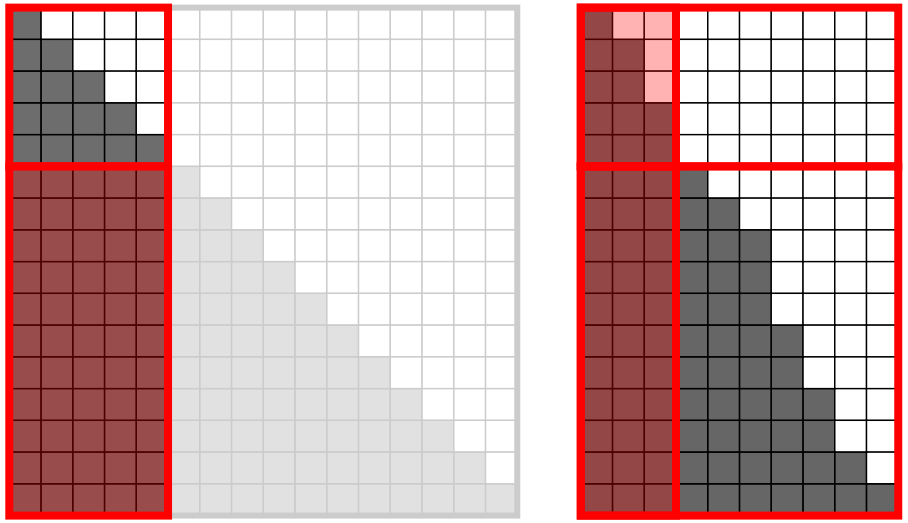
TRSM optimization towards sparsity

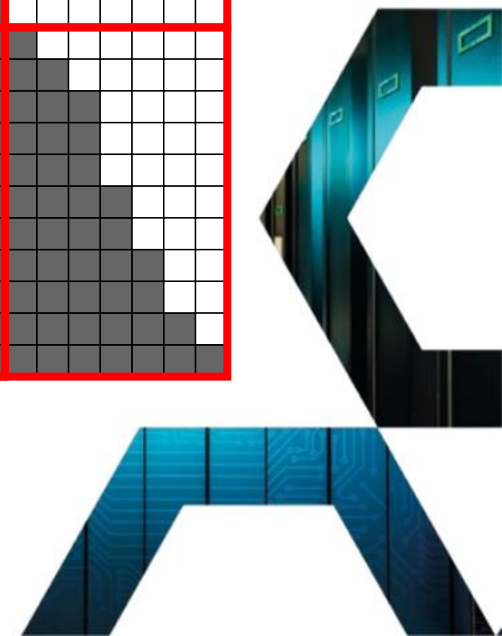
$$\begin{bmatrix} \mathbf{L}_{11} & \mathbf{O} & \mathbf{X}_{11} & \mathbf{O} \\ \mathbf{L}_{21} & \mathbf{L}_{22} & \mathbf{X}_{21} & \mathbf{X}_{22} \end{bmatrix} \xrightarrow{\mathbf{r}_1 = \mathbf{L}_{11}^{-1}\mathbf{r}_1} \begin{bmatrix} \mathbf{I} & \mathbf{O} & \mathbf{L}_{11}^{-1}\mathbf{X}_{11} & \mathbf{O} \\ \mathbf{L}_{21} & \mathbf{L}_{22} & \mathbf{X}_{21} & \mathbf{X}_{22} \end{bmatrix} \xrightarrow{\mathbf{r}_2 = \mathbf{r}_2 - \mathbf{L}_{21}\mathbf{r}_1} \begin{bmatrix} \mathbf{I} & \mathbf{O} & \mathbf{L}_{11}^{-1}\mathbf{X}_{11} & \mathbf{O} \\ \mathbf{O} & \mathbf{L}_{22} & \mathbf{X}_{21} - \mathbf{L}_{21}\mathbf{L}_{11}^{-1}\mathbf{X}_{11} & \mathbf{X}_{22} \end{bmatrix}$$

TRSM

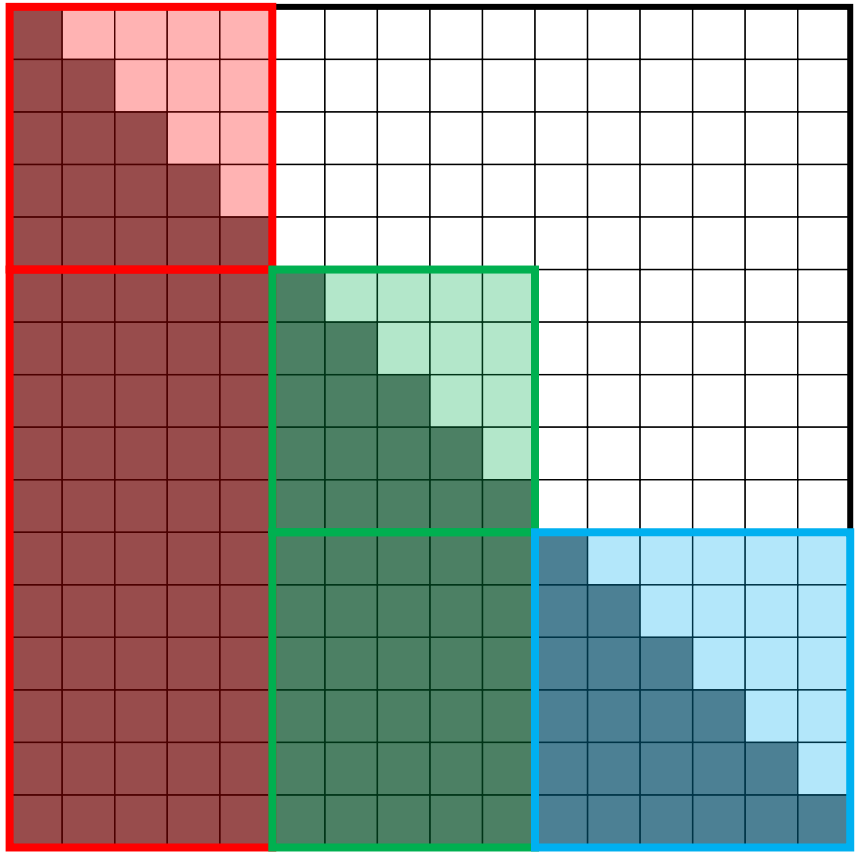


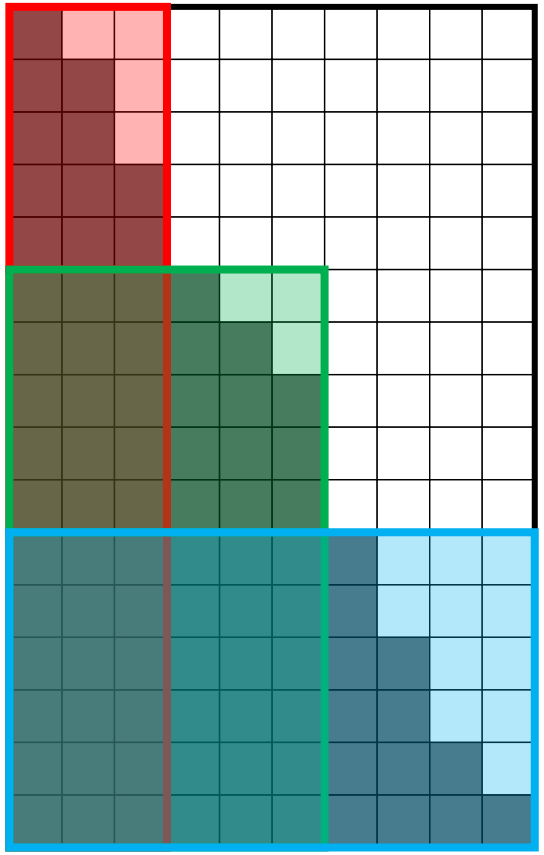
GEMM



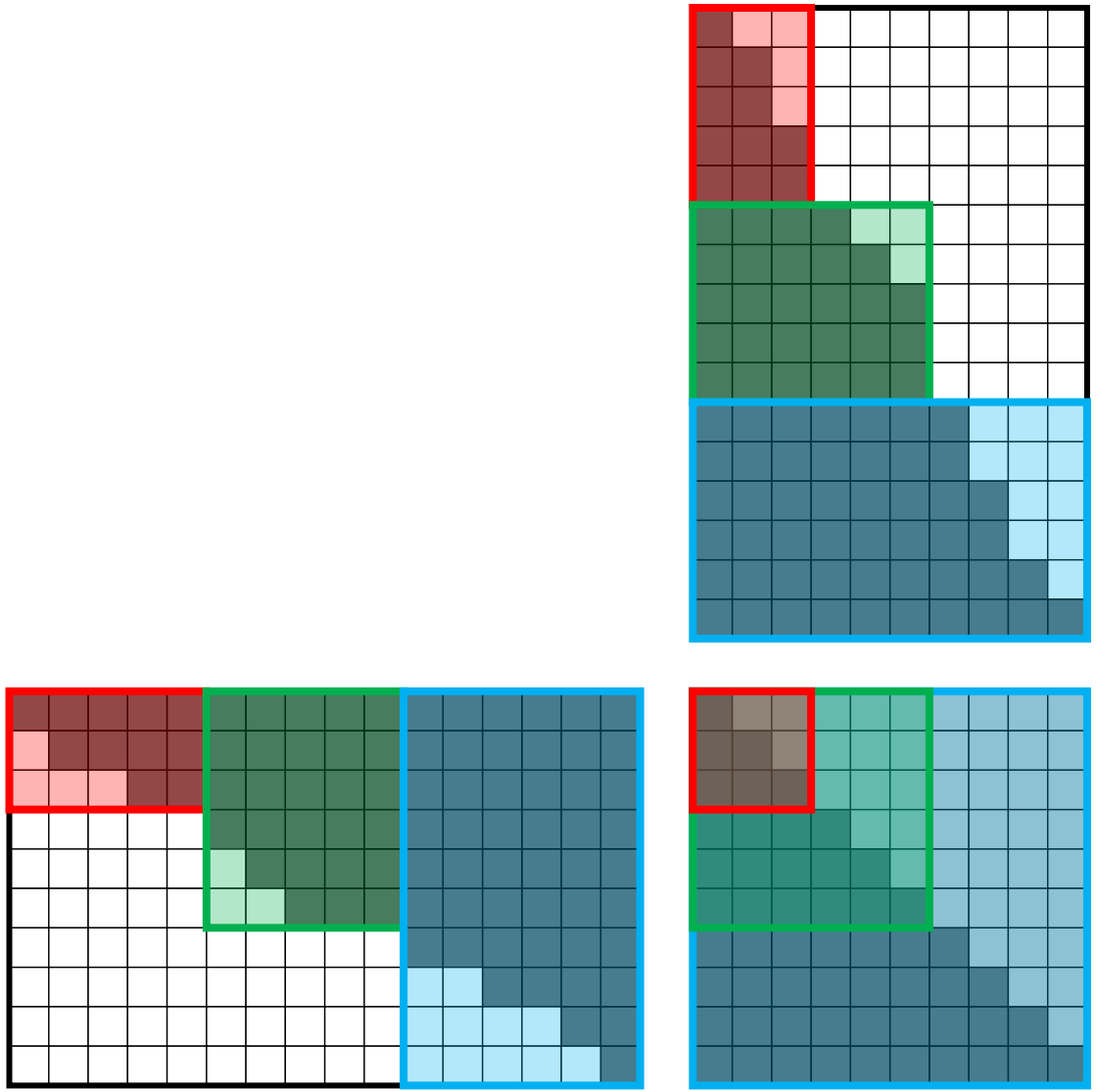


TRSM optimization towards sparsity

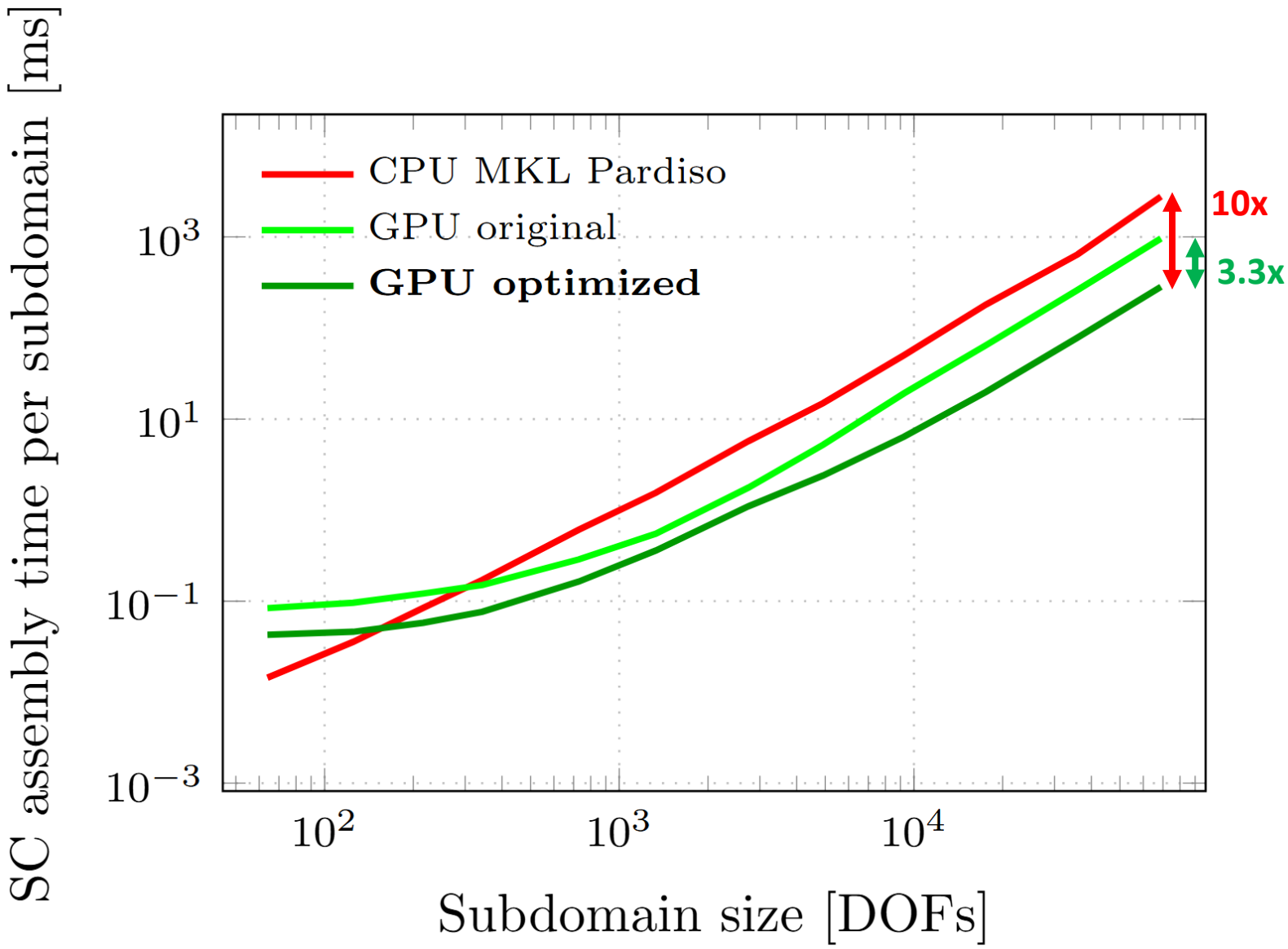




SYRK optimization towards sparsity



Results: speedup of preprocessing



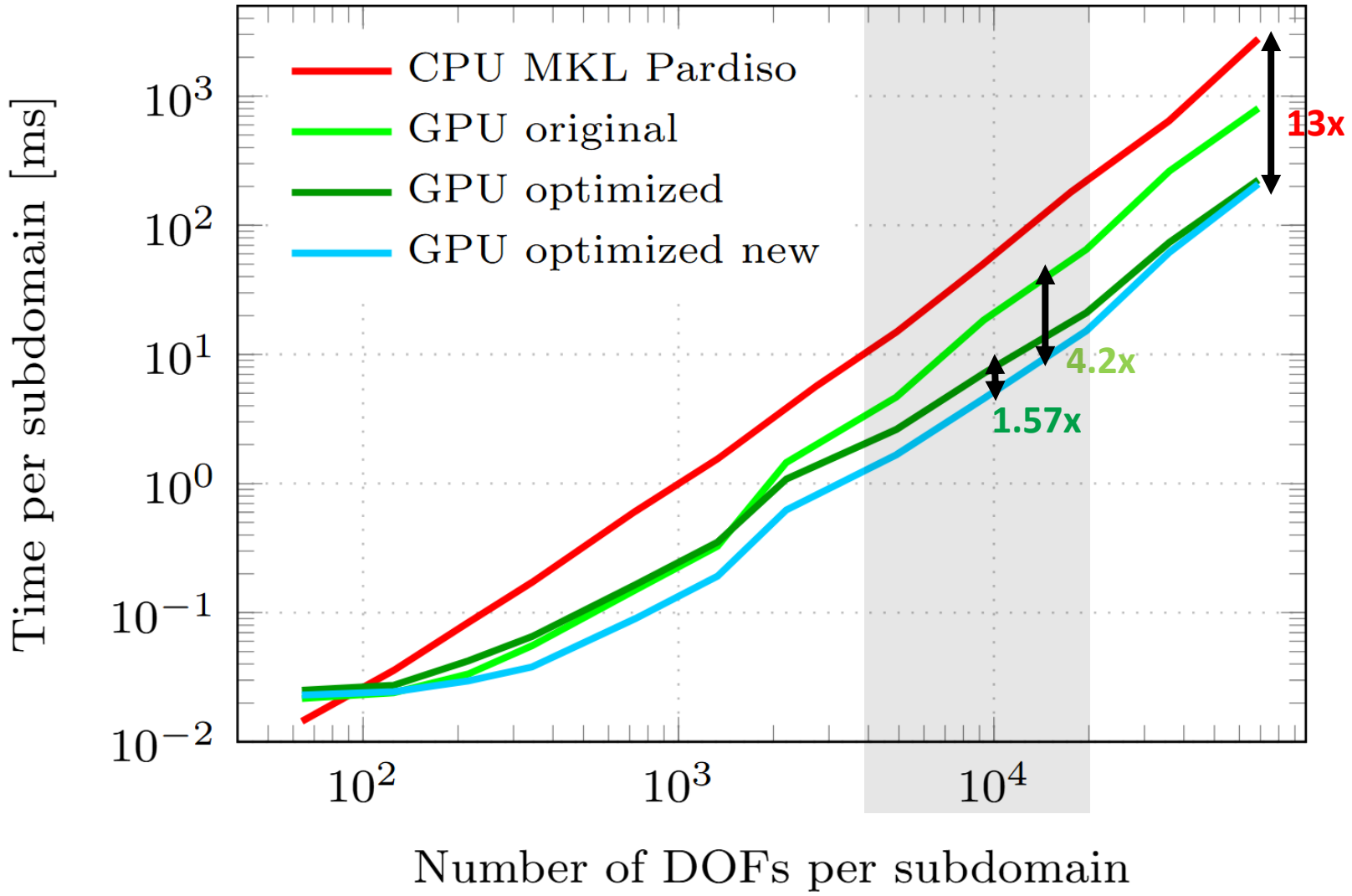
Can we do better?

- Yes

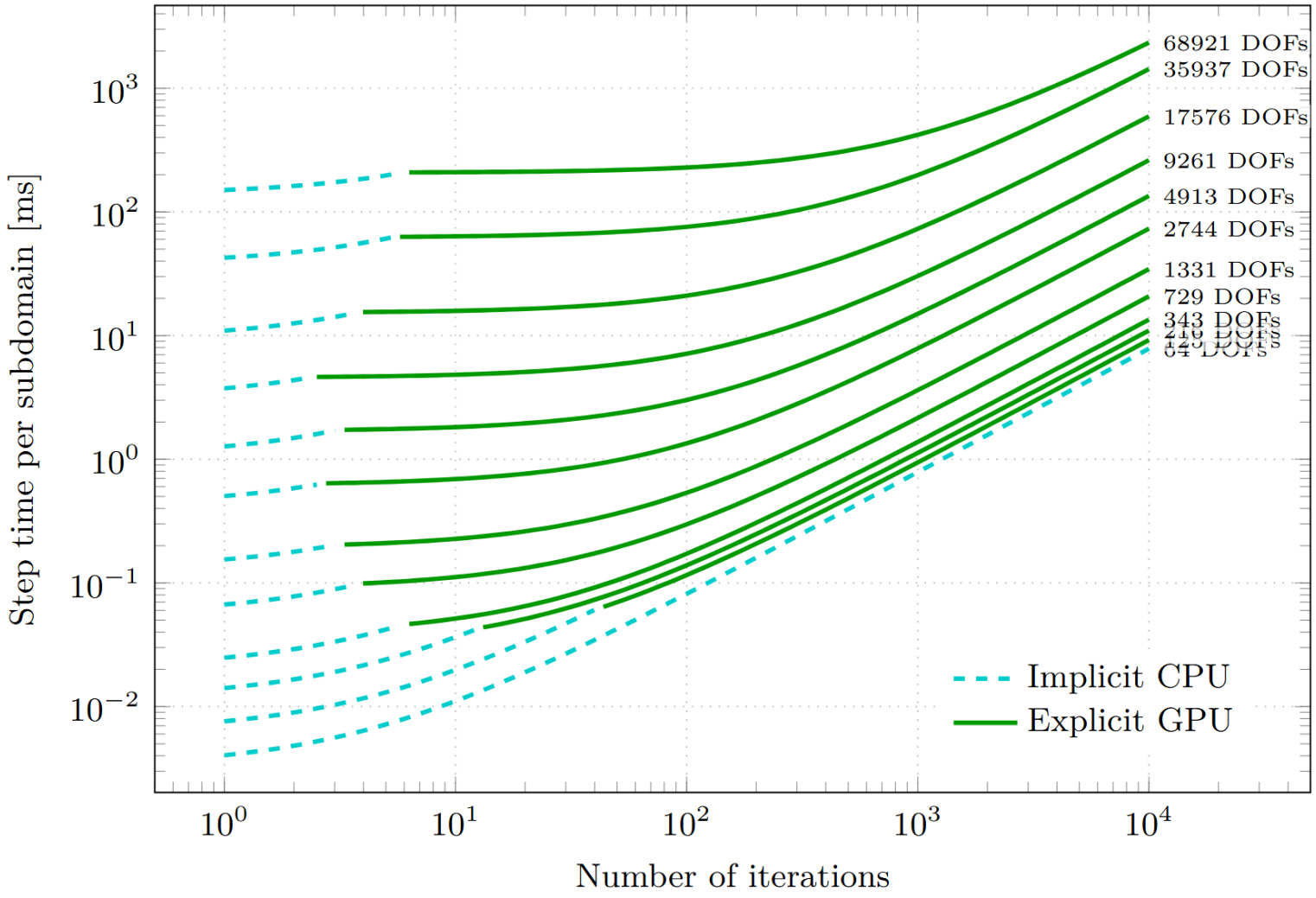
- Extract submatrices: CPU -> GPU
- Need for more advanced GPU memory management
  - Less free memory => less streams executing

- GPU assembly is then fully asynchronous, CUDA graph

Results: time of preprocessing



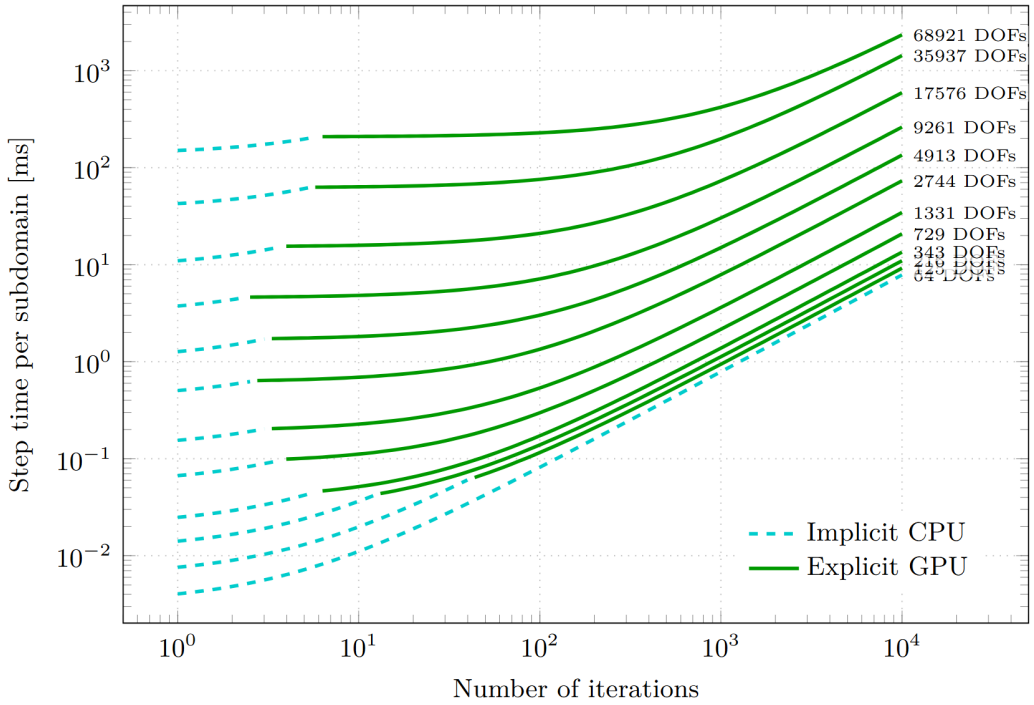
Results: break-even point

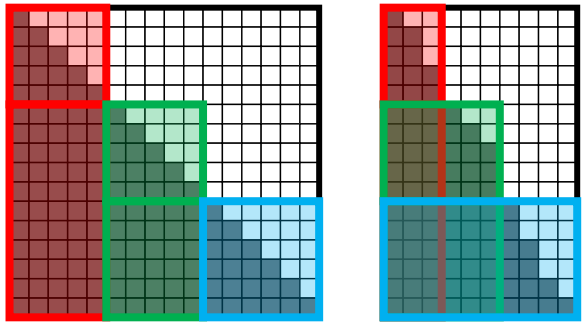


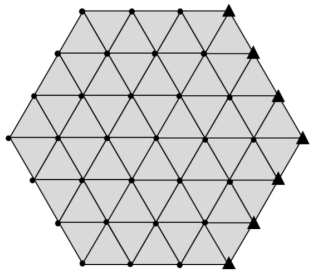
Conclusion

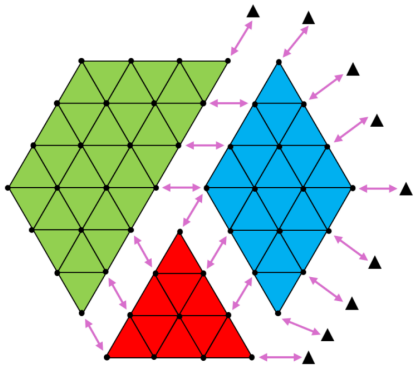
$$\mathbf{F}_i = \mathbf{B}_i \mathbf{L}_i^{-\top} \mathbf{L}_i^{-1} \mathbf{B}_i^{\top}$$

- FETI dual operator
- Schur complement assembly on GPU
- Optimizations for sparsity
- Memory management, CUDA graph
- Up to **13x speedup** of Scur complement assembly
- Break-even point at **3 iterations** on Karolina
  - **7 iterations** on LUMI
- AMD and Intel wrapper ready









VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Questions?

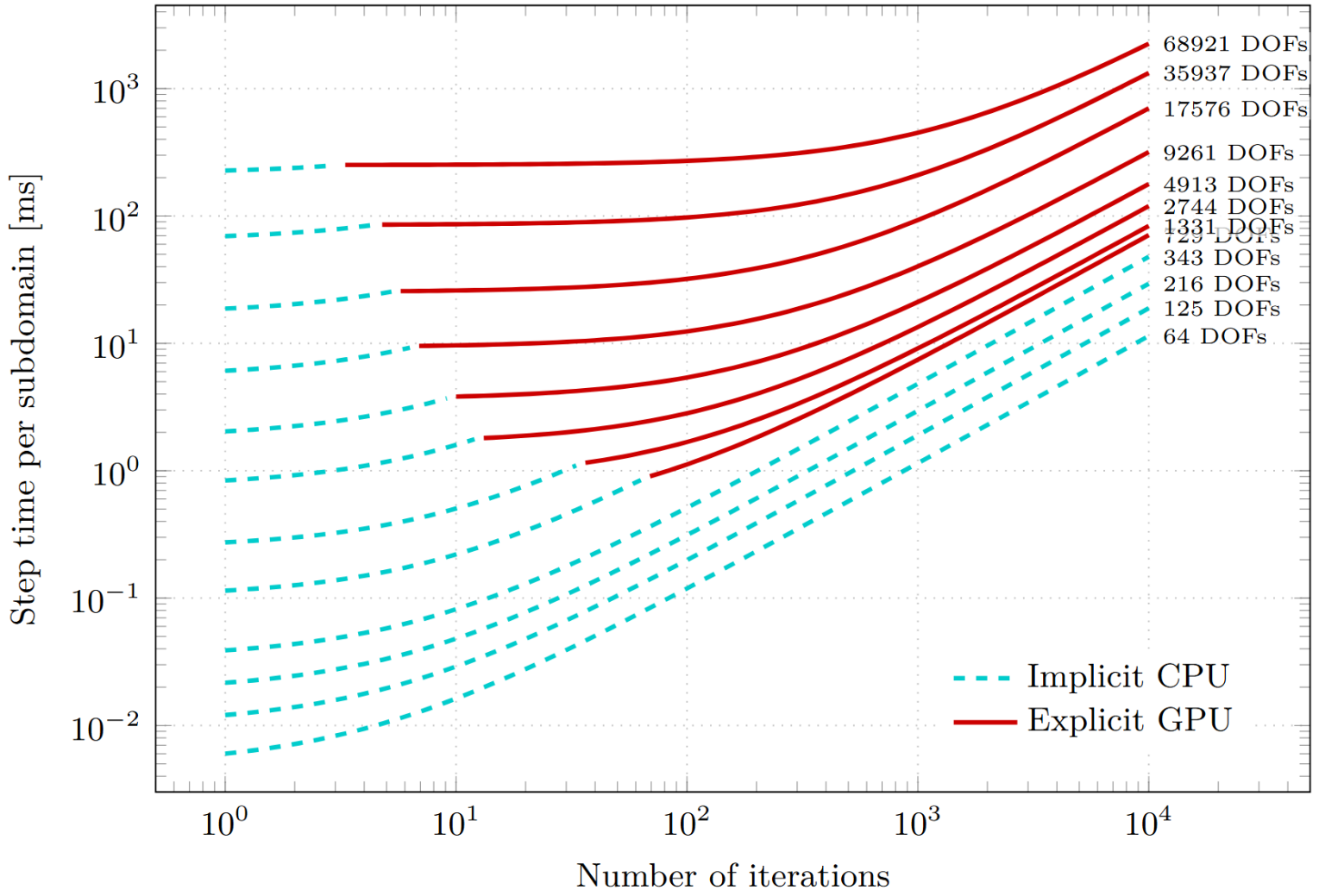
Jakub Homola
jakub.homola@vsb.cz

IT4Innovations National Supercomputing Center
VSB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.eu

Results: break-even point, LUMI

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

1/8th of GPU node
7 cores of AMD EPYC 7A53
1 GCD of AMD MI250x



Outline

- FEM/FETI/DDM quick intro
- ESPRESO
- Schur complement
- Implicit cpu vs explicit gpu, break-even point
- SC assembly
- Optimization for sparsity
- Memory management
- Results, break-even points, apply speedup, update slowdown

Outline

- FEM/FETI/DDM quick intro
- ESPRESO
- Schur complement
- Implicit cpu vs explicit gpu, break-even point
- SC assembly
- Optimization for sparsity
- Memory management
- Results, break-even points, apply speedup, update slowdown