Optimizing BEM computation through compression and paralleization

Stefan Bornhofen, ETIS

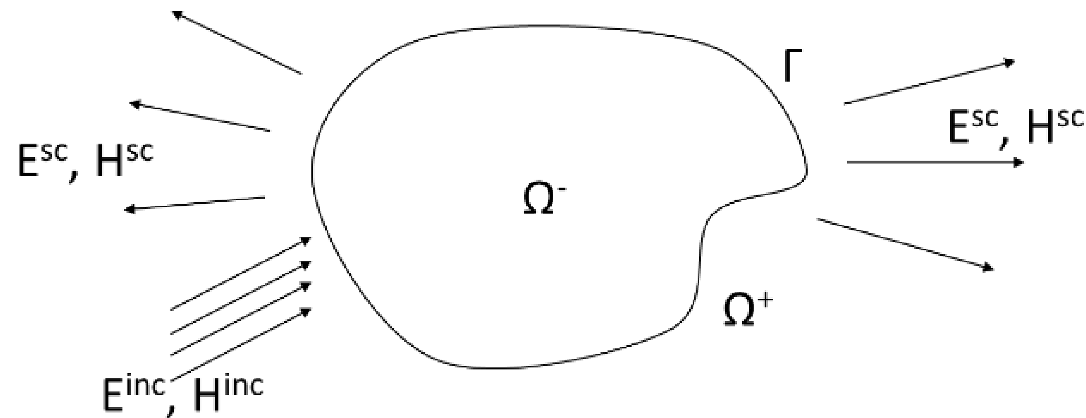
13/01/2023

Plan

- Electromagnetic scattering problem and BEM
- H-matrix and ACA Compression
- Parallelization (OpenMP + MPI)
- Results
- Perspectives

Electromagnetic scattering problem

We consider the scattering problem of electromagnetic waves for a perfect conducting body with a dielectric coating.

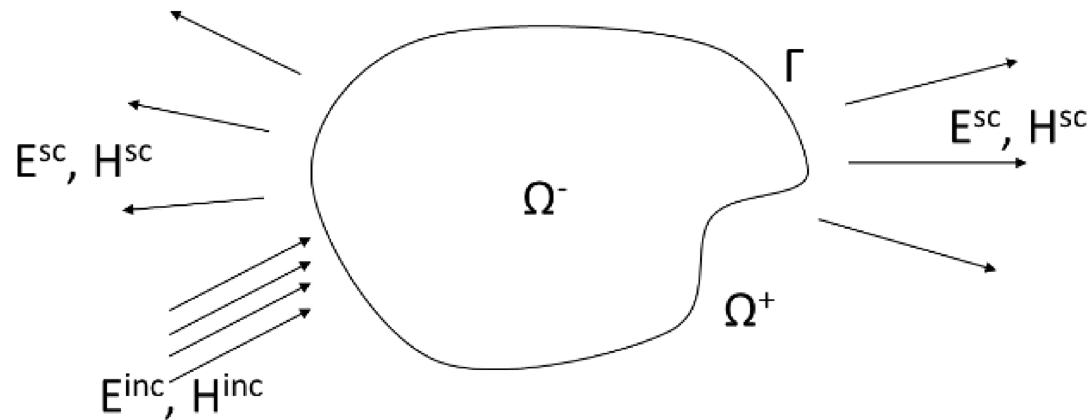


- We illuminate this system by incident electromagnetic waves.
- Electromagnetic waves propagate in $\Omega+ = R^n \backslash \Omega$.
- Scattering waves occur as the incident waves bounce off in a variety of directions, depending on the wavelength of the incident waves and the structure of the object.

Electromagnetic scattering problem

We define total electromagnetic fields (E, H) in Ω+ as:

$$E = E^{inc} + E^{sc}$$
$$H = H^{inc} + H^{sc}$$



$Find\ (\boldsymbol{E}, \boldsymbol{H})\ such\ that$

Faraday's Law
Maxwell's Law
Boundary condition with impedance operator
Silver-Müller radiation condition

$$\begin{cases} \mathbf{rot}\,\boldsymbol{E} + ik_0\mu\boldsymbol{H} = 0 \quad in\ \Omega^+ \\ \mathbf{rot}\,\boldsymbol{H} - ik_0\varepsilon\boldsymbol{E} = 0 \quad in\ \Omega^+ \\ \boldsymbol{E}_{tg} - Z(\boldsymbol{n} \times \boldsymbol{H}) = 0 \quad on\ \Gamma \\ \lim_{r\to\infty} r(\boldsymbol{E} \times \boldsymbol{n}_r + \boldsymbol{H}) = 0 \end{cases}$$

RCS

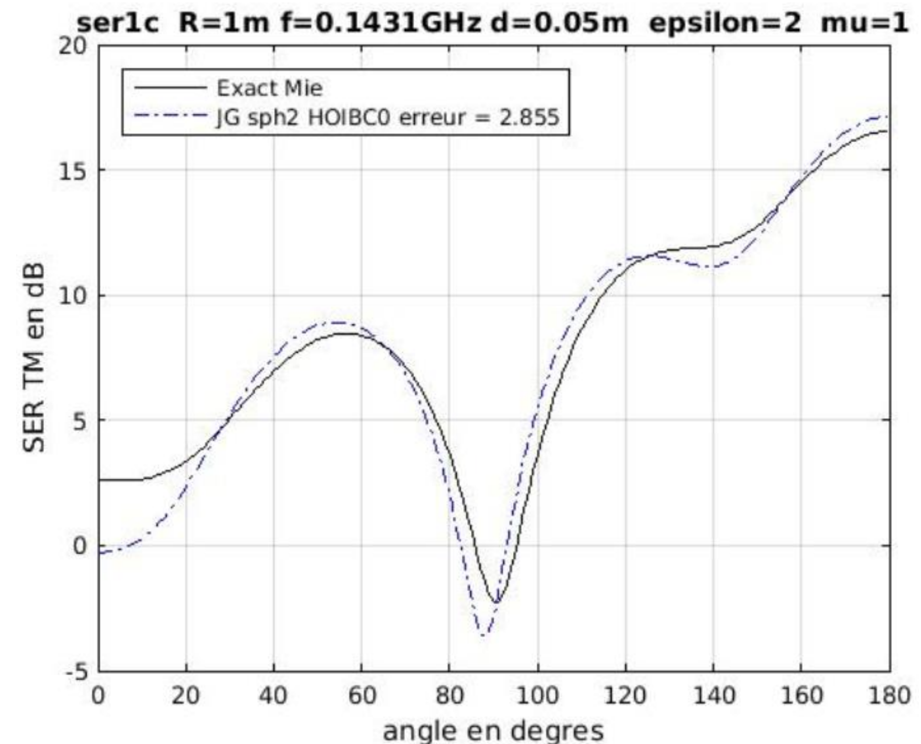
Radar Cross Section is a measure of how detectable an object is by radar.

$$\sigma = \lim_{r \to \infty} 4\pi r^2 \frac{|\mathbf{E}^{sc}|^2}{|\mathbf{E}^{inc}|^2}$$

where r is the observation distance.

High RCS values mean high radar detectability.

Stealth aircrafts are designed to have low RCS, passenger airliners to have a high RCS.



BEM

$$\textit{Find } (\boldsymbol{E}, \boldsymbol{H}) \textit{ such that}$$

$$\begin{cases} \mathbf{rot}\boldsymbol{E} + ik_0\mu\boldsymbol{H} = 0 & in\ \Omega^+ \\ \mathbf{rot}\boldsymbol{H} - ik_0\varepsilon\boldsymbol{E} = 0 & in\ \Omega^+ \\ \boldsymbol{E}_{tg} - Z(\boldsymbol{n} \times \boldsymbol{H}) = 0 & on\ \Gamma \\ \lim_{r\to\infty} r(\boldsymbol{E} \times \boldsymbol{n}_r + \boldsymbol{H}) = 0 \end{cases}$$

The boundary element method (BEM) is a method of solving linear partial differential equations which have been formulated in boundary integral form.

We introduce current densities J and M on the boundary Γ as

$$\mathbf{M} = [\mathbf{E} \times \mathbf{n}]_-^+ \quad \mathbf{J} = [\mathbf{n} \times \mathbf{H}]_-^+$$

where [ ]+− denotes difference between upper (+) and lower (-) values of the interface, n is the exterior normal vector to the surface.

The Stratton-Chu integral representation allows characterizing the electromagnetic fields in terms of surface current densities.

$$\mathbf{E}(x) = ikZ_0(B - S)\mathbf{J}(x) - Q\mathbf{M}(x)$$
$$\mathbf{H}(x) = -Q\mathbf{J}(x) + ikZ_0^{-1}(B - S)\mathbf{M}(x)$$

The current densities are unknowns in the integral formulation of the problem. The knowledge of J and M on the boundary of the volume is sufficient to determine the field throughout the space.

BEM

Find $(\boldsymbol{E}, \boldsymbol{H})$ *such that*

$$\begin{cases} \mathbf{rot}\,\boldsymbol{E} + ik_0\mu\boldsymbol{H} = 0 \;\; in \;\Omega^+ \\ \mathbf{rot}\,\boldsymbol{H} - ik_0\varepsilon\boldsymbol{E} = 0 \;\; in \;\Omega^+ \\ \boldsymbol{E}_{tg} - Z(\boldsymbol{n}\times\boldsymbol{H}) = 0 \;\; on \;\Gamma \\ \lim_{r\to\infty} r(\boldsymbol{E}\times\boldsymbol{n}_r + \boldsymbol{H}) = 0 \end{cases}$$

The boundary element method (BEM) is a method of solving linear partial differential equations which have been formulated in boundary integral form.

We introduce current densities J and M on the boundary Γ as

$$\mathbf{M} = [\mathbf{E}\times\mathbf{n}]^+_- \quad \mathbf{J} = [\mathbf{n}\times\mathbf{H}]^+_-$$

where [ ]+− denotes difference between upper (+) and lower (-) values of the interface, n is the exterior normal vector to the surface.

$$(B-S)\mathbf{J} := \int_\Gamma \left(G(x,y)\mathbf{J}(y) + \frac{1}{k^2}\nabla_x G(x,y)\mathrm{div}_\Gamma \mathbf{J} \right) d\Gamma(y)$$

$$Q\mathbf{M} := \int_\Gamma \nabla_y G(x,y)\times\mathbf{M}d\Gamma(y)$$

The Stratton-Chu integral representation allows characterizing the electromagnetic fields in terms of surface current densities.

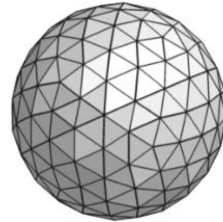
$$\mathbf{E}(x) = ikZ_0(B-S)\mathbf{J}(x) - Q\mathbf{M}(x)$$
$$\mathbf{H}(x) = -Q\mathbf{J}(x) + ikZ_0^{-1}(B-S)\mathbf{M}(x)$$

The current densities are unknowns in the integral formulation of the problem. The knowledge of J and M on the boundary of the volume is sufficient to determine the field throughout the space.

Rao-Wilton-Glisson

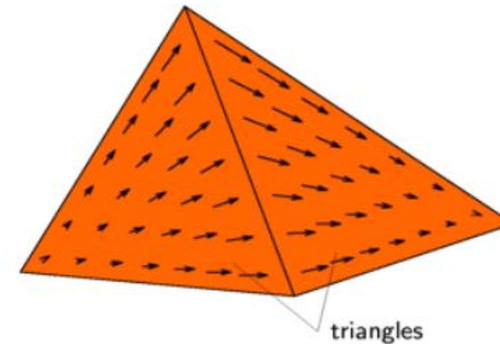
We approach the surface of the obstacle by a surface Γ_h composed of finite number of triangles. T_i for i = 1 to N_T.

$$\Gamma_h = \bigcup_{i=1}^{N_T} T_i$$



We denote by N_e the total number of edges of the mesh component Γ_h. Let $\{f_i\}_{i=1,Ne}$ be a base of Rao-Wilton-Glisson functions, where each function correspond to one edge. We decompose the electric and magnetic currents:

$$\mathbf{J}(y) = \sum_{i=1}^{N_e} J_i \mathbf{f}_i(y), \quad \mathbf{M}(y) = \sum_{j=1}^{N_e} M_j \mathbf{f}_j(y)$$



The unknown variables to find are now: J_i and M_i, i=1…N_e

Rao-Wilton-Glisson

After discretization and using the RWG functions, the operators

$$(B-S)\mathbf{J} := \int_\Gamma \left(G(x,y)\mathbf{J}(y) + \frac{1}{k^2}\nabla_x G(x,y)\mathrm{div}_\Gamma \mathbf{J} \right) d\Gamma(y)$$

$$Q\mathbf{M} := \int_\Gamma \nabla_y G(x,y) \times \mathbf{M} d\Gamma(y)$$

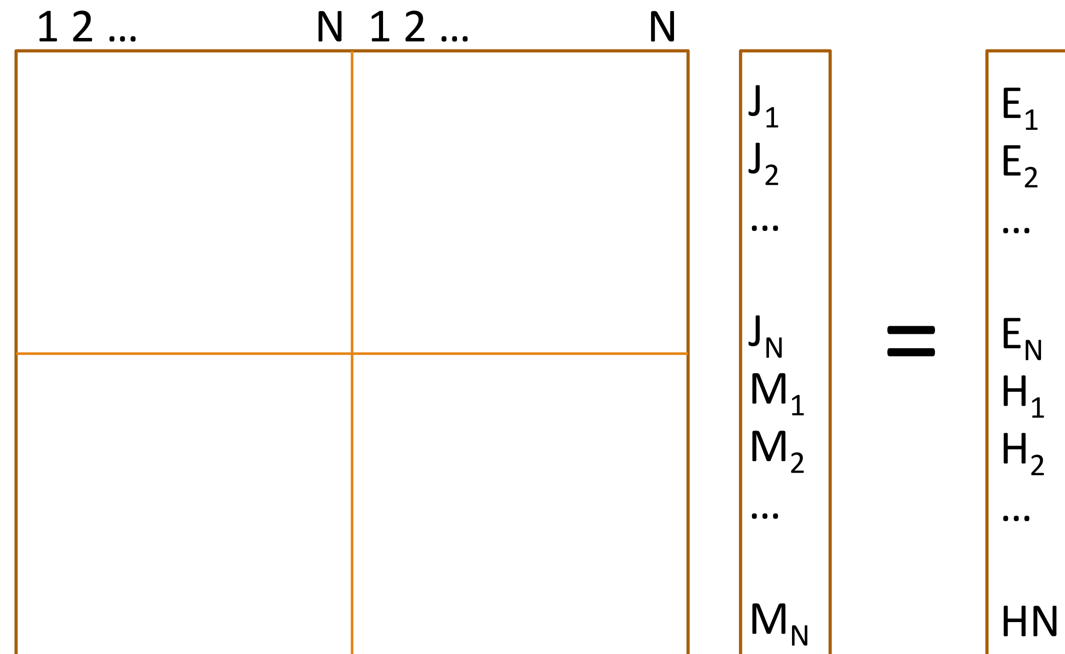
can be written in matrix form

$$(B-S)_{i,j} = i \iint_{\Gamma_h} kG(s,s')\mathbf{f}_j(s') \cdot \mathbf{f}_i(s) - \frac{1}{k} G(s,s')(\mathrm{div}_\Gamma \mathbf{f}_i)(\mathrm{div}'_\Gamma \mathbf{f}_j) ds ds'$$

$$Q_{i,j} = -i \iint_{\Gamma_h} [\mathbf{f}_i(s) \times \mathbf{f}_j(s')] \cdot \nabla'_\Gamma G(s,s') ds ds'$$

Final equation

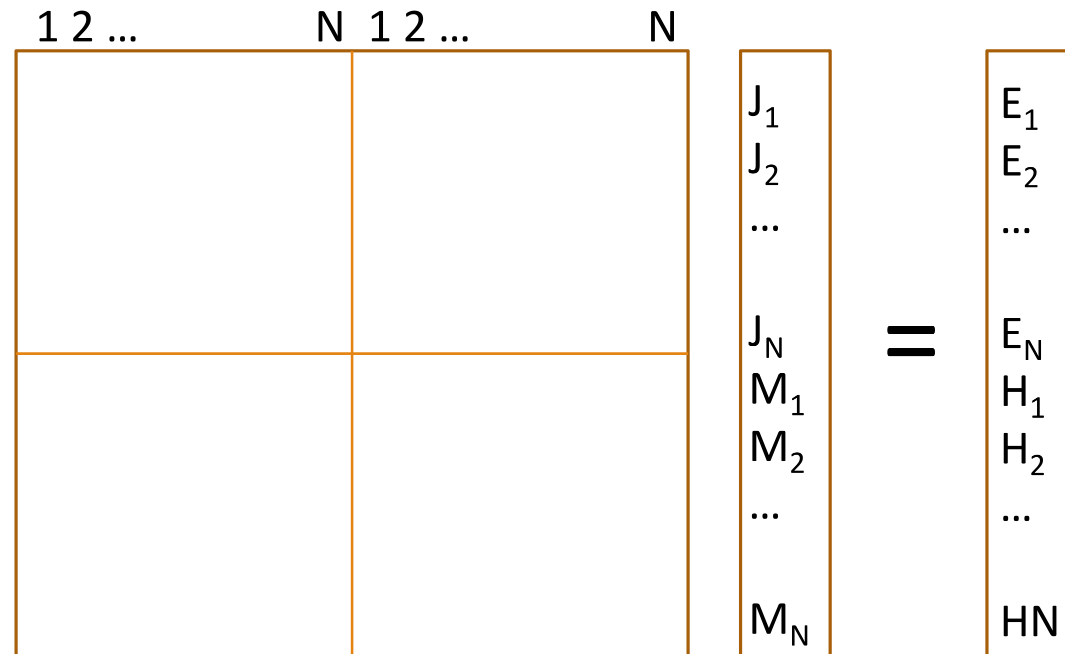
For a mesh with N egdes, we end up with a system of linear equations with size 2*N.



$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Final equation

For a mesh with N egdes, we end up with a system of linear equations with size 2*N.



O(N²)

$$\begin{bmatrix} [A1]-\frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q]+\frac{b_1}{2}[D][C_{KH}]+\frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T+\frac{a_2}{2a_0}[D][C_{KH}]-\frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2]-\frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

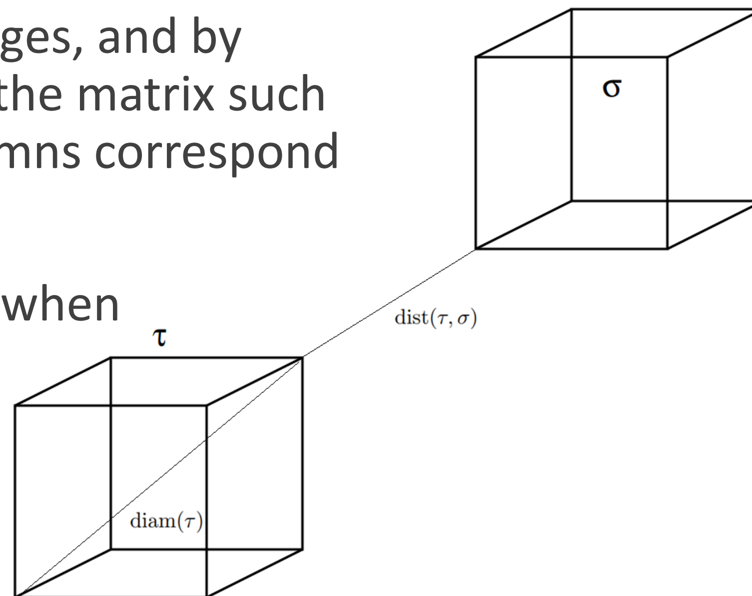
H-matrix

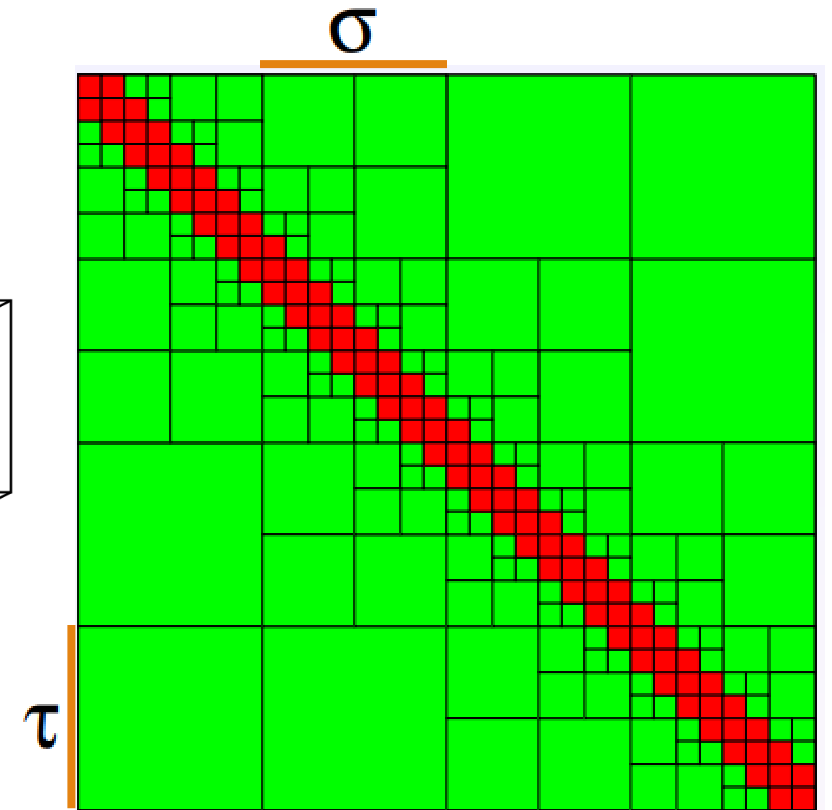
H-matrices allow creating a subdivided and data-sparse representation of dense BEM-matrices.

Idea: Split the matrix into sub matrices called blocks by a recursive subdivision of the geometry defining groups of edges, and by permuting the indices (1..N) in the matrix such that consecutive rows and columns correspond to edges at a close distance.

Stop subdivision of block (τ, σ) when

$$\mathrm{diam}(\tau) \leq \mathrm{dist}(\tau, \sigma)$$



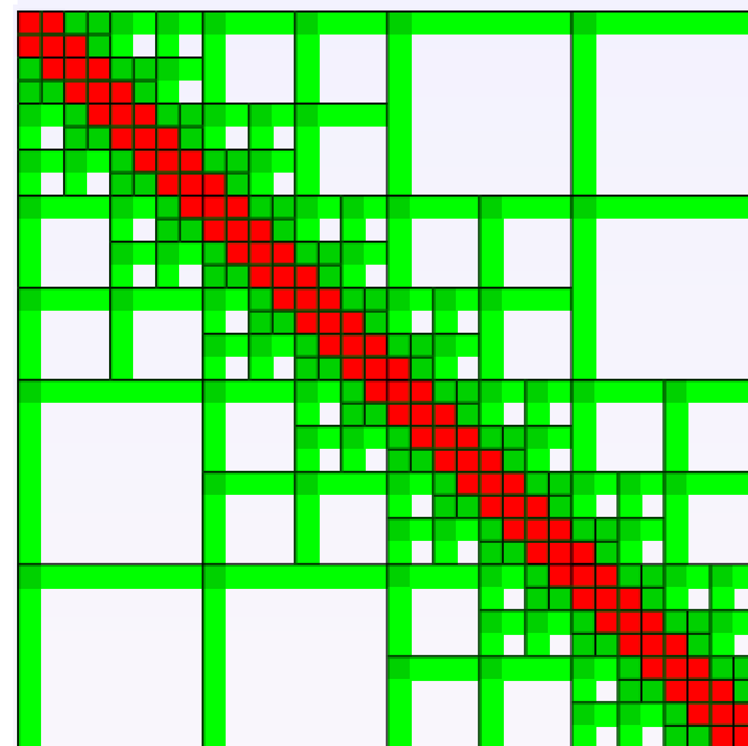


First optimization: Compression

It is known that dense matrix blocks representing the interactions between two well-separated groups can be accurately represented by a reduced set of column vectors.

The diagonal blocks, which represent intra-group interactions, as well as blocks of degenerated form having few rows or columns are not admissible for compression.

All other blocks correspond to interactions of well-separated groups of edges and will be compressed by the ACA algorithm.



ACA Compression

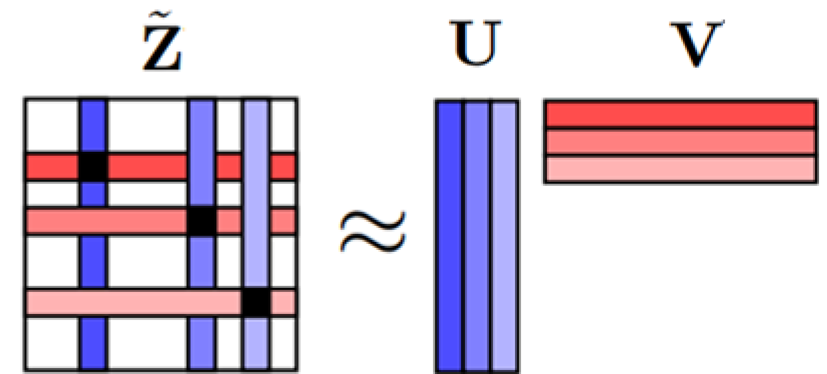
Adaptive Cross Approximation (ACA) is a greedy compression algorithm producing low-rank approximations of a given matrix.

Let $\mathbf{Z}^{m\times n}$ be a matrix of size nxm.

Construct an approximation $\tilde{\mathbf{Z}}^{m\times n} = \mathbf{U}^{m\times r}\mathbf{V}^{r\times n} = \sum_{i=1}^{r} \mathbf{u}_i^{m\times 1}\mathbf{v}_i^{1\times n}$

Such that $||\mathbf{R}^{m\times n}|| = ||\mathbf{Z}^{m\times n} - \tilde{\mathbf{Z}}^{m\times n}|| \leq \varepsilon ||\mathbf{Z}^{m\times n}||$

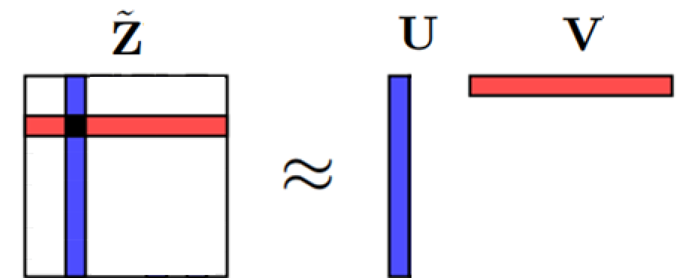
for a given tolerance eps.



ACA Compression

R is constructed by incrementally adding a rank-1 matrix.

For k=0 we have $\tilde{\mathbf{Z}}^{(0)} = 0$ and $\mathbf{R}^{(1)} = \mathbf{Z} - \tilde{\mathbf{Z}}^{(0)}$



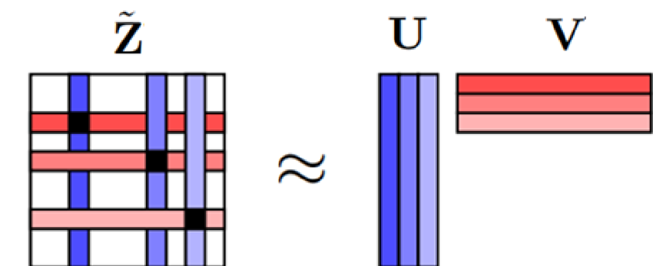
We choose a simple and fast heuristic to define the pivots:

- find max$|R_{ij}|$ in the current row I

- then max$|R_{kj}|$ in the chosen column j

Define u_k and v_k such that the product $u_k \cdot v_k$ exactly reproduces the entries of (I_k)th row and (J_k)th column of the error matrix $R^{(k)}$.

This process continues adaptively until $\|\mathbf{R}^{(k)}\| \leq \varepsilon \|\mathbf{Z}\|$



A compression of rank k requires to store k(m + n) entries instead of m*n entries.

Note that ACA compression also accelerates the matrix-vector product since Zv = UVv.

ACA Compression

ACA compression is unprofitable when $k(m+n) > m*n$

This happens quite often because for high frequencies we use eps as small as 10^{-9}

In that case we abandon ACA compression and use the dense block. To accelerate the generation of the dense block, we store the all the values which have been computed during ACA compression so that we just need to compute the missing ones

- Advantage: gain time

- Drawback: temporarily allocate a matrix of size $m*n$

ACA Compression

Osaka frontal

mesh=SPH3, freqency=300Mhz, coating thickness= 5mm, eps=5

SPH3 is a sphere discretized into a triangle mesh with 2478 edges. The matrix size is therefore 4956*4956.

Sequential filling without ACA: 381,08s
Matrix: 393Mb

Sequential filling with ACA: 454,68s (« FILL0 »)
Matrix: 259Mb (65% compression)
ACA compressed blocks: 28 out of 562 scheduled

Second optimization: Parallel Computing

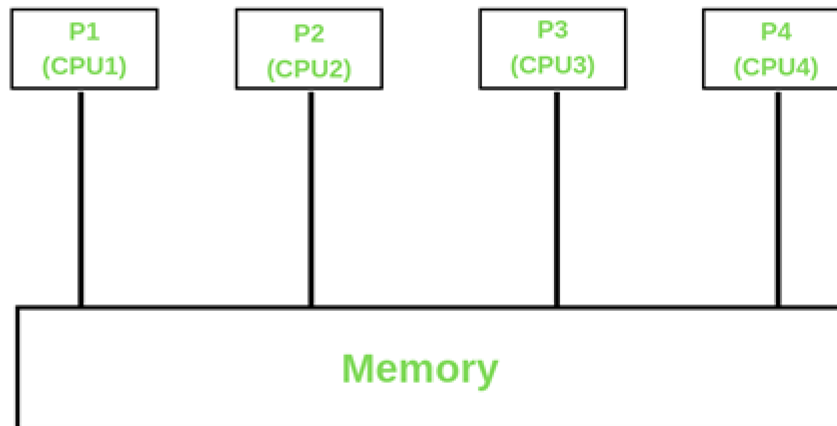
Due to the independence of the blocks, H-matrix computing can be conveniently coupled with parallel computing technologies to

1. distribute the memory load
2. accelerate the matrix filling
3. accelerate the matrix/vector product for fast iterative solvers.

OpenMP vs. MPI

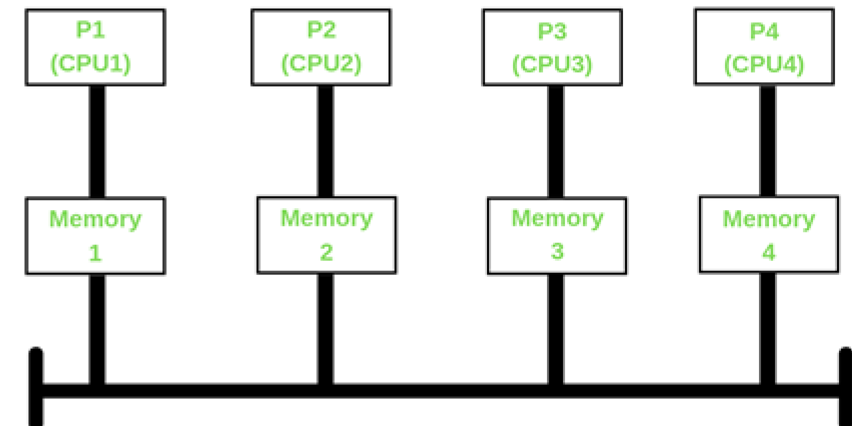
OpenMP

for architectures with shared memory



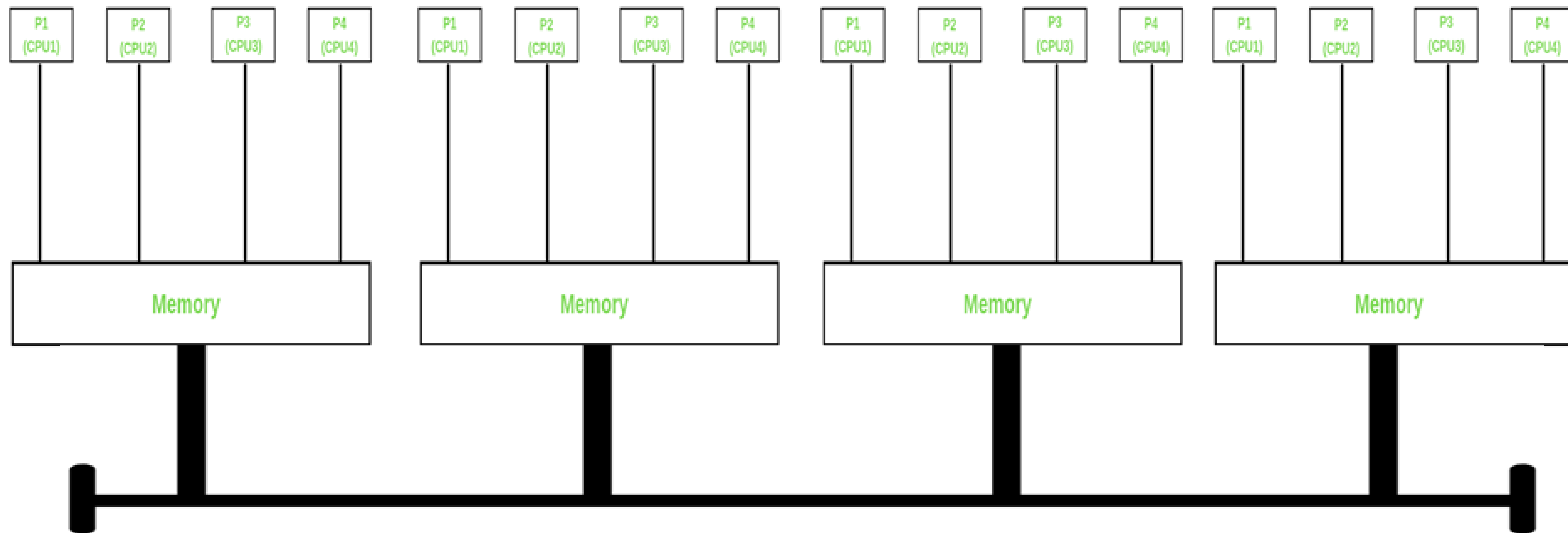
MPI

for architectures with distributed memory



OpenMP vs. MPI

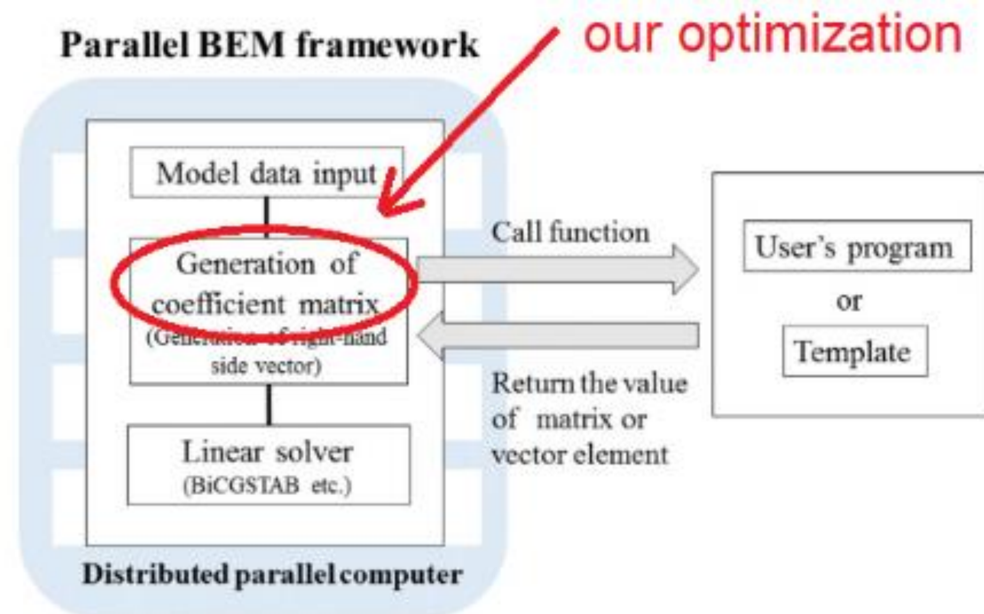
Hybrid OpenMP + MPI



HACApK

The HACApK library (Ida et al., 2014) provides a powerful generic programming framework for CPU clusters with hybrid OpenMP + MPI parallelization, and more recently even GPU parallelization (Ohshima et al., 2018).

We optimized HACApK for our use case notably in the matrix filling stage.



Parallel Computing

In the scope of this study, we only concentrate on the filling stage

Filling

- The H-matrix blocks are assigned to the MPI processes. Each process will hold a share of the blocks.
- The MPI processes fill the blocks in parallel (dense or ACA).
- Multiple blocks of the same MPI process can be filled in parallel through OpenMP.

After the filling stage, the H-matrix is scattered over the MPI processes.

Solving (Matrix-vector product)

- Each MPI process multiplies its blocks with the vector and produces a partial result.
- Multiple blocks of the same process can be multiplied in parallel through OpenMP.
- The partial multiplications are broadcasted among the MPI processes and added up to the final product.

First result

Osaka frontal (32 core)
16 MPI * 2 OpenMP

mesh=SPH3, freqency=300Mhz, coating thickness= 5mm, eps=5

Matrix filling with HACApK (« FILL1 »)

454,68s => **44,78s**

10.15x faster

Static scheduling

Osaka frontal (32 core)
16 MPI * 2 OpenMP

mesh=SPH3, freqency=300Mhz, coating thickness= 5mm, eps=5

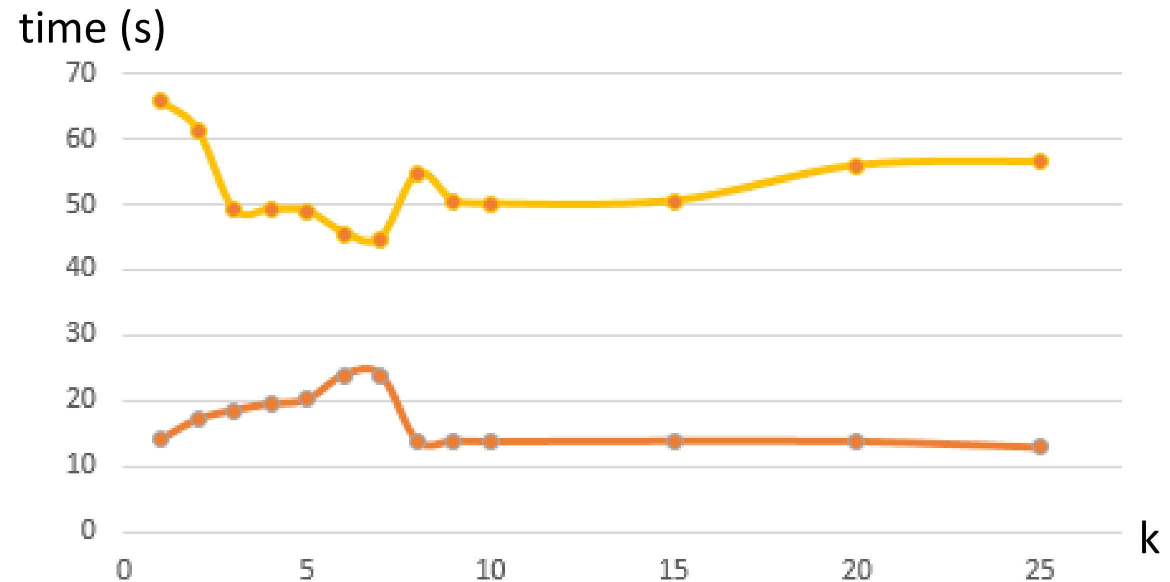
Static scheduling (OpenMP and MPI) may lead to unbalanced workload, as the filling time is bound by the last MPI node to finish.

```
MPI process          0 filling leaves          1  to 98
MPI process          1 filling leaves         99  to 252
MPI process          3 filling leaves        371  to 496
MPI process          4 filling leaves        497  to 590
MPI process          5 filling leaves        591  to 755
MPI process          6 filling leaves        756  to 842
MPI process         13 filling leaves       1492  to 1635
MPI process         14 filling leaves       1636  to 1770
MPI process         15 filling leaves       1771  to 1921
MPI process          2 filling leaves        253  to 370
MPI process          7 filling leaves        843  to 1001
MPI process          8 filling leaves       1002  to 1041
MPI process          9 filling leaves       1042  to 1170
MPI process         10 filling leaves       1171  to 1269
MPI process         11 filling leaves       1270  to 1419
MPI process         12 filling leaves       1420  to 1491
MPI process    8 finished in     23.97s,  40 blocks, mem= 0.03 Gb, compr=  57.8%
MPI process    0 finished in     26.94s,  98 blocks, mem= 0.02 Gb, compr=  36.1%
MPI process    7 finished in     30.22s, 159 blocks, mem= 0.01 Gb, compr= 100.0%
MPI process    2 finished in     30.52s, 118 blocks, mem= 0.01 Gb, compr=  98.0%
MPI process    3 finished in     30.53s, 126 blocks, mem= 0.01 Gb, compr=  96.7%
MPI process   14 finished in     31.16s, 135 blocks, mem= 0.01 Gb, compr=  74.8%
MPI process    1 finished in     31.35s, 154 blocks, mem= 0.01 Gb, compr= 100.0%
MPI process    6 finished in     31.39s,  87 blocks, mem= 0.02 Gb, compr=  97.8%
MPI process   12 finished in     32.19s,  72 blocks, mem= 0.03 Gb, compr=  43.6%
MPI process    5 finished in     32.67s, 165 blocks, mem= 0.01 Gb, compr= 100.0%
MPI process   10 finished in     34.23s,  99 blocks, mem= 0.02 Gb, compr=  80.5%
MPI process   11 finished in     35.50s, 150 blocks, mem= 0.01 Gb, compr= 100.0%
MPI process   13 finished in     35.85s, 144 blocks, mem= 0.01 Gb, compr= 100.0%
MPI process   15 finished in     36.35s, 151 blocks, mem= 0.01 Gb, compr=  99.7%
MPI process    9 finished in     37.89s, 129 blocks, mem= 0.01 Gb, compr=  99.0%
MPI process    4 finished in     44.78s,  94 blocks, mem= 0.04 Gb, compr=  61.7%
```

Static scheduling

HACApK uses a mean estimate of ACA-k in order to precompute the computational load for each block (m*n for uncompressed and k*(m+n) for compressed blocks). This allows deciding upfront how many blocks will be assigned to each MPI node, and then to each OMP thread.

However the estimation often turns out to be bad, especially in our use case where a lot of scheduled ACA-compressions are rejected.



44,78s for k=7 (« FILL1 »)

Dynamic OpenMP scheduling

Parallelizing loops is an OpenMP specialty.

The simple clause

```
 !$OMP DO SCHEDULE(DYNAMIC)
do i=1,N
   ...
```

before a loop construct specifies that each thread executes one element of the loop and then requests another element until there are no more elements available.

Dynamic OpenMP scheduling

Parallelizing loops is an OpenMP specialty.

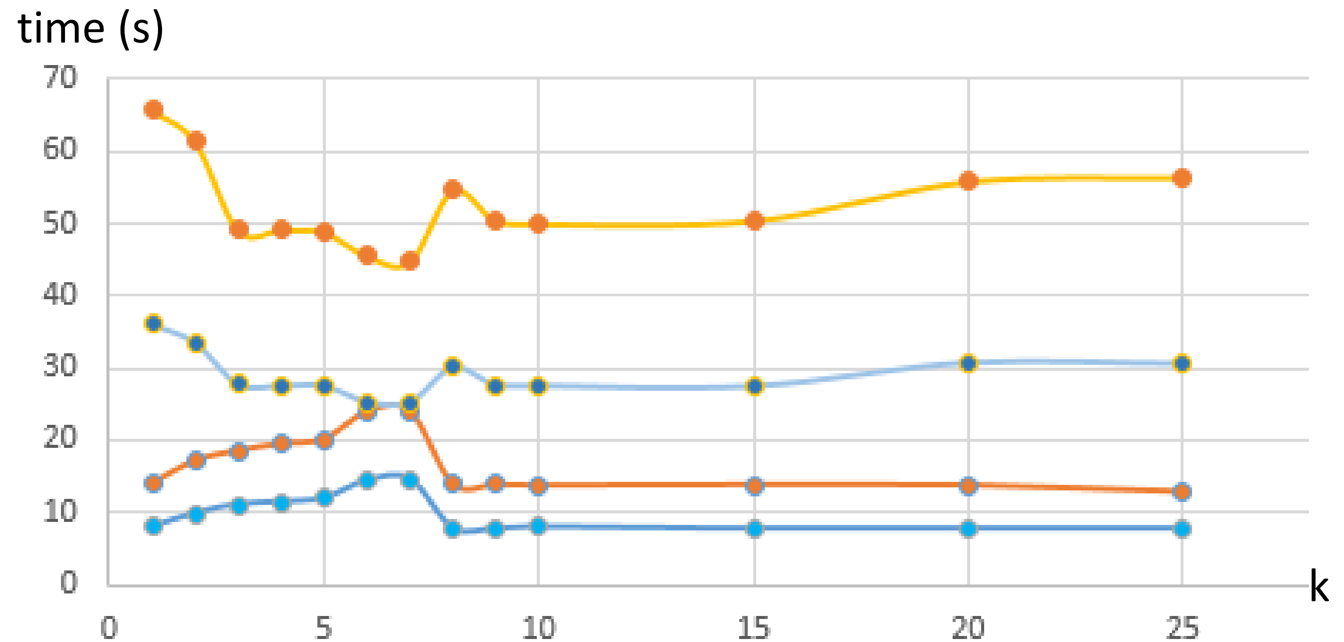
The simple clause

```
 !$OMP DO SCHEDULE(DYNAMIC)
do i=1,N
   ...
```

before a loop construct specifies that each thread executes one element of the loop and then requests another element until there are no more elements available.

44,78s => **25,56s (« FILL2 »)**

(1.75x faster)

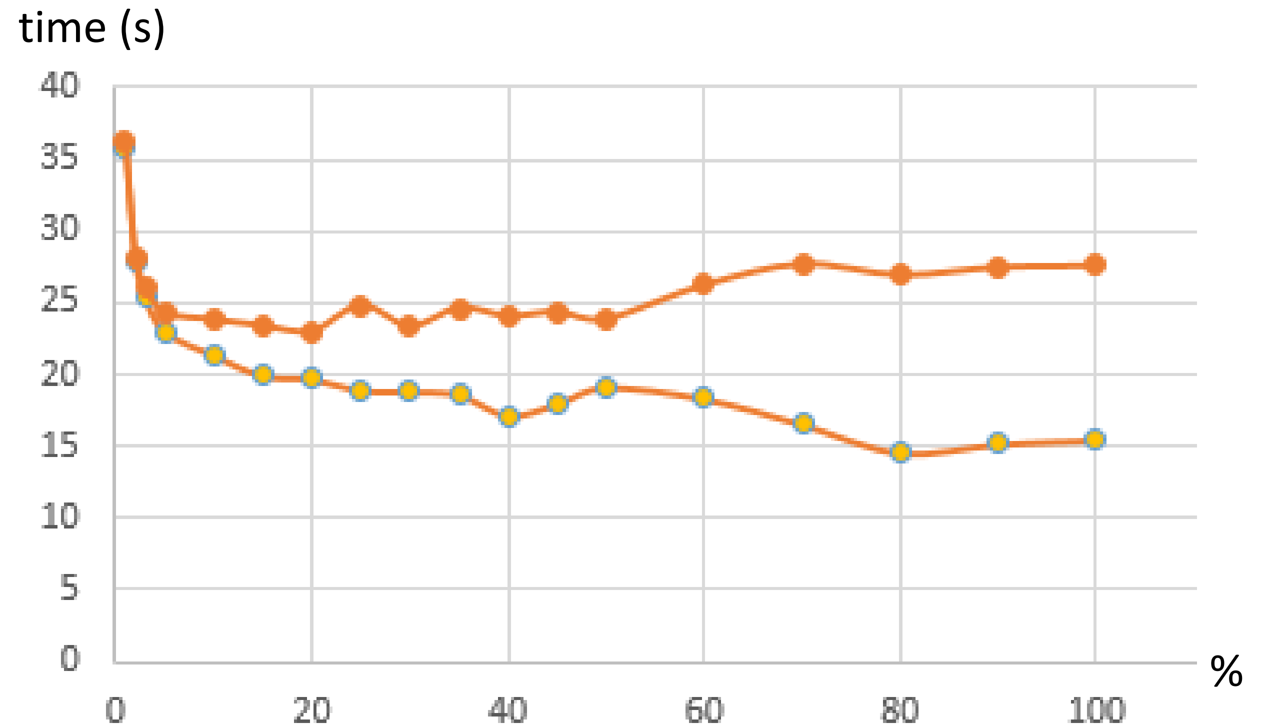


Dynamic MPI scheduling

In a similar way (but with much more programming effort) it is possible to parallelize loops in MPI.

For N MPI nodes, we only send x% of the planned workload (which is 1/N of the total matrix) to each MPI process. After finishing its work, the node requests another x% of the workload until there is no more work available.

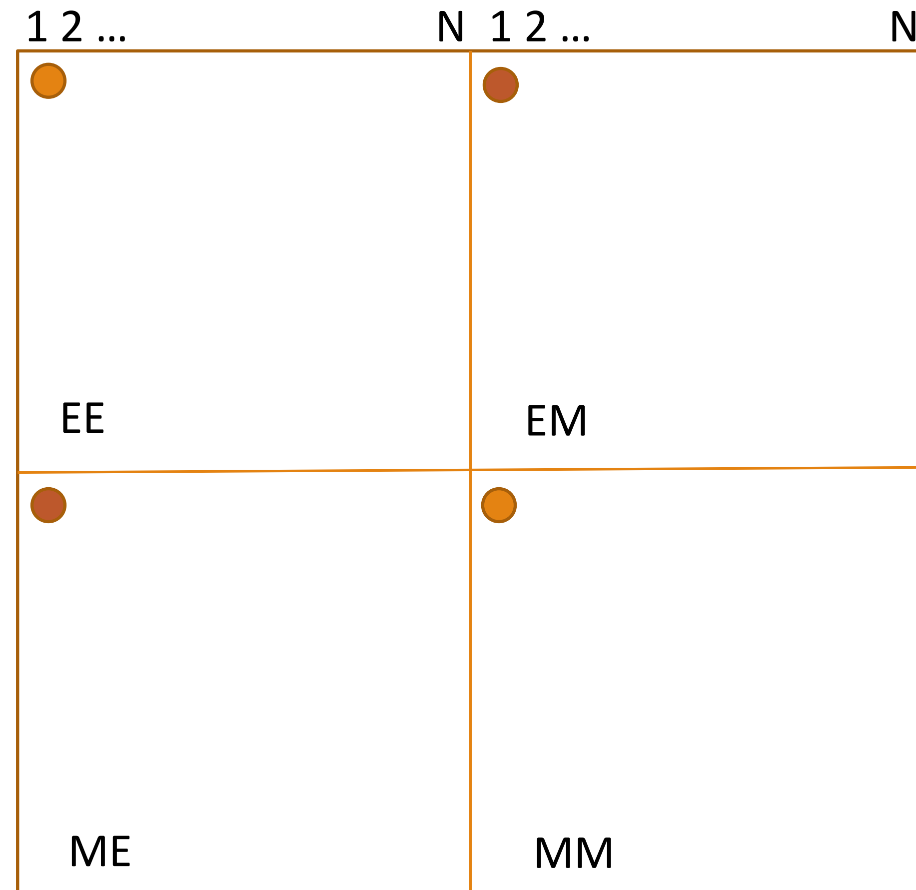
25,56s for static k=7 =>
23,80s for dynamic x=10% (« FILL3 »)



Dynamic MPI scheduling reduces the workload imbalance, but adds management overhead.
Still, by chosing workload chunks of 10%-20%, it is typically faster than static scheduling (even with best k).

$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

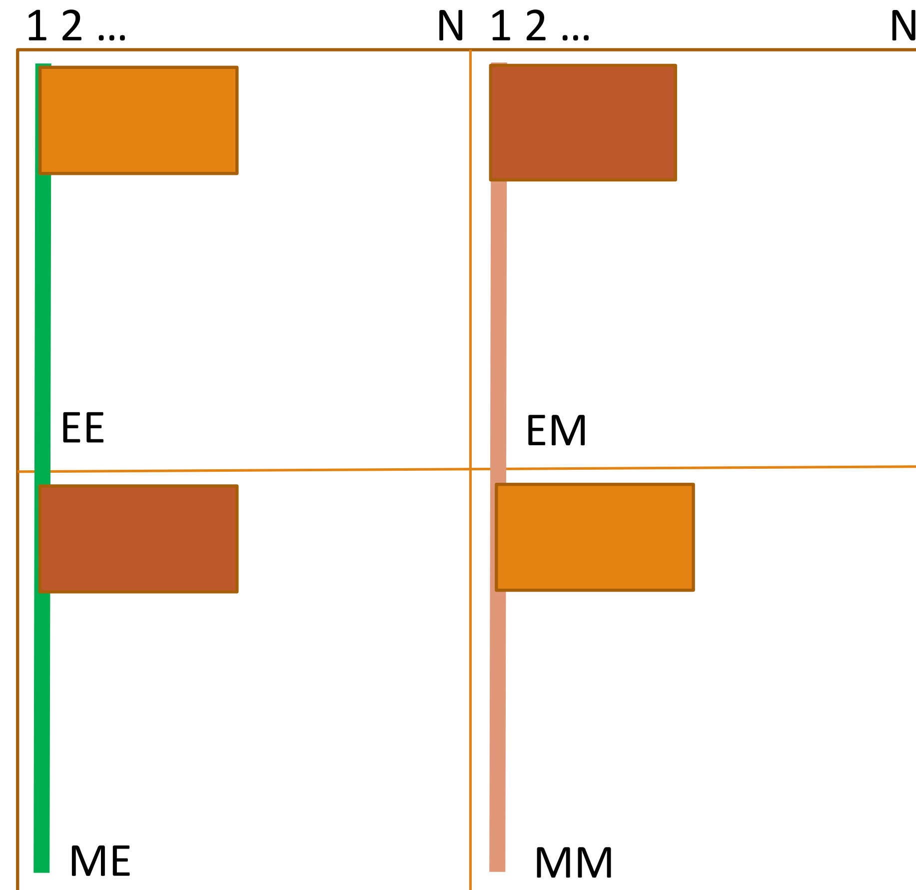
Inter-matrix symmetry



$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Inter-matrix symmetry

However these blocks are not necessarily on the same MPI node to be computed together.



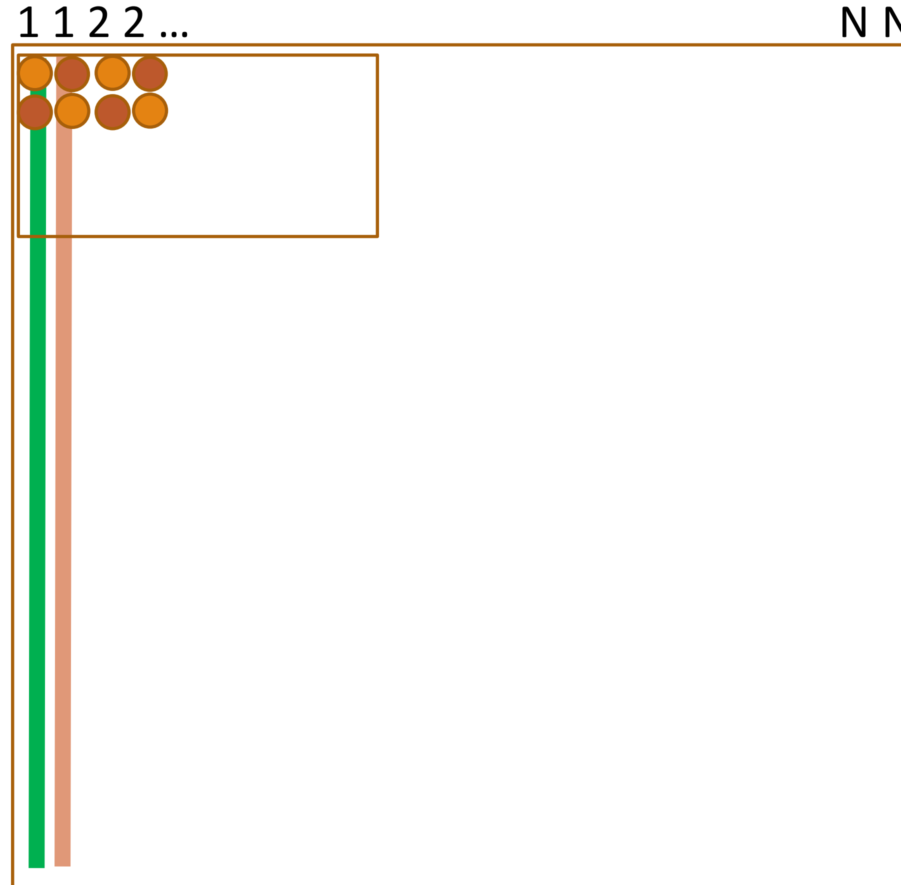
$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Inter-matrix symmetry

Reorder the columns such that the four values are stored next to each other.

Apply symmetry calculus to uncompressed blocks:

When vij_{EE} is computed, we compute vij_{MM}, vij_{EM} and vij_{ME} at the same time and benefit from mutual intermediate values.

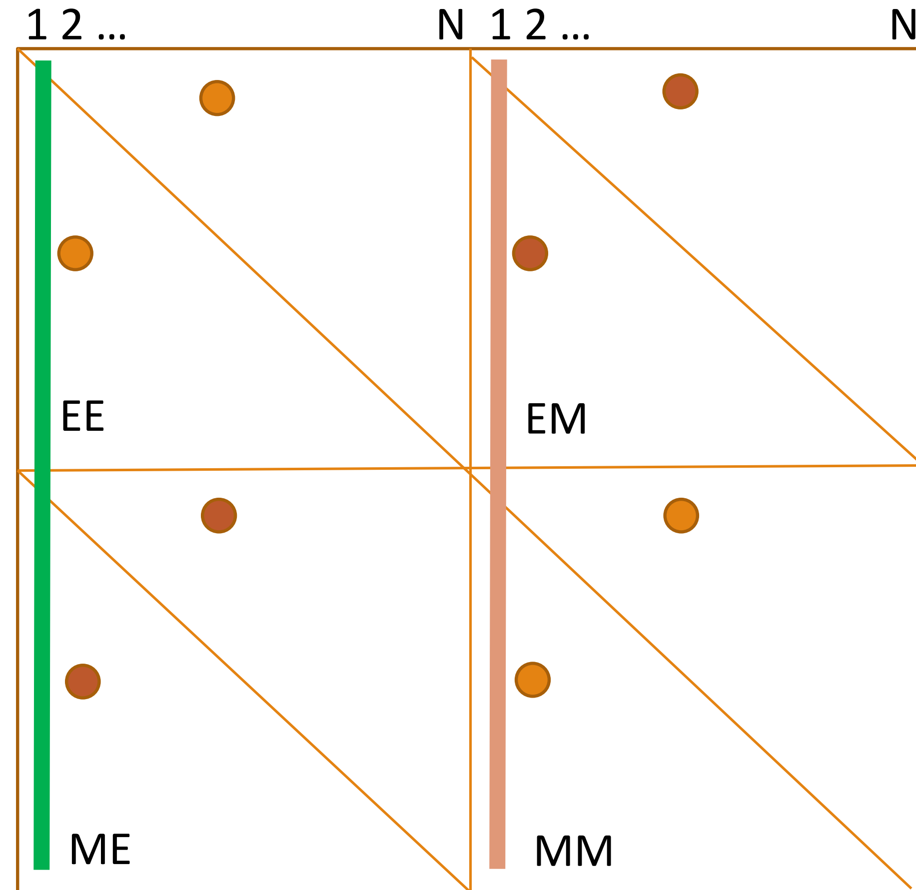


23,80s (« FILL3 »)

16,01s (« FILL9 »)

$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

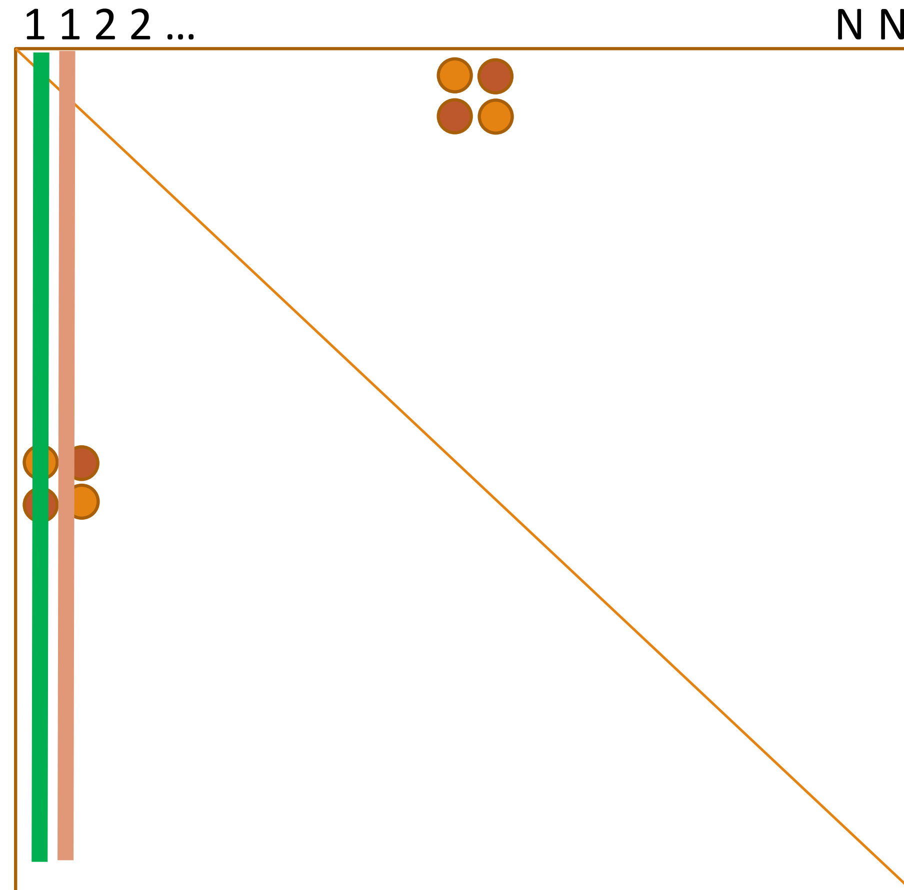
Intra-matrix symmetry



$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Intra-matrix symmetry

The same symmetry for the reordered columns:



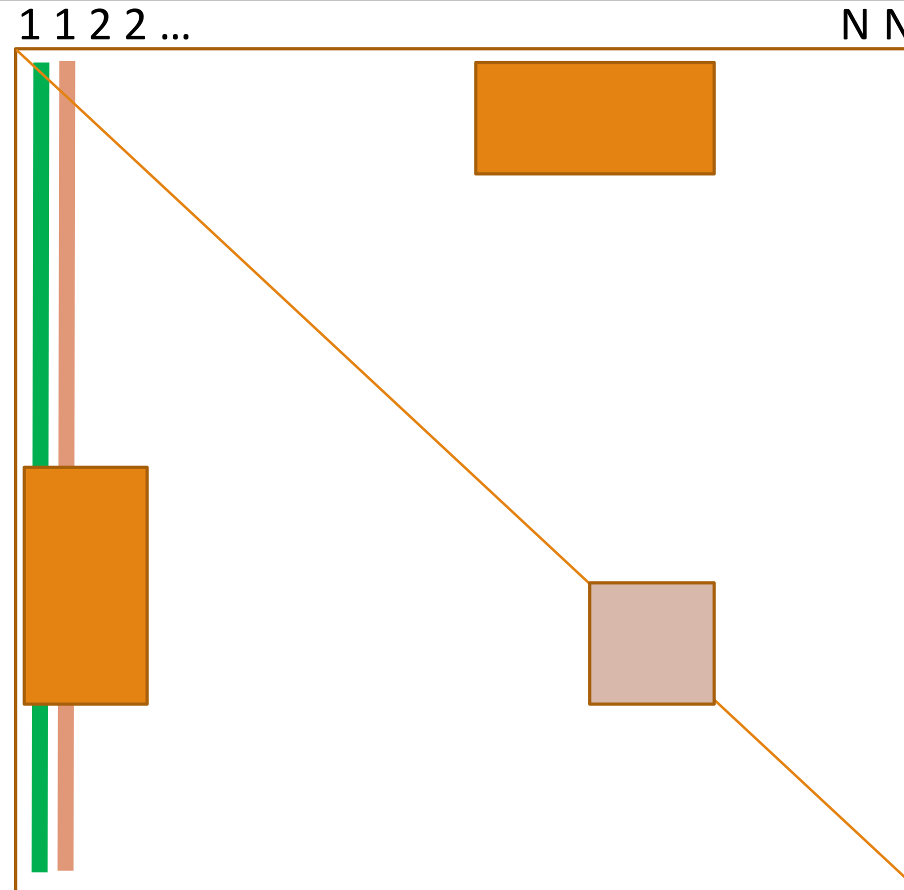
$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Intra-matrix symmetry

Again these blocks are not necessarily on the same MPI node to be computed together.

Assign a couple of symmetric blocks to the same MPI node (diagonal blocks do not have an opposite).

Apply symmetry to <u>uncompressed</u> blocks: When block1 is filled, we fill block2 at the same time and benefit from mutual intermediate values.

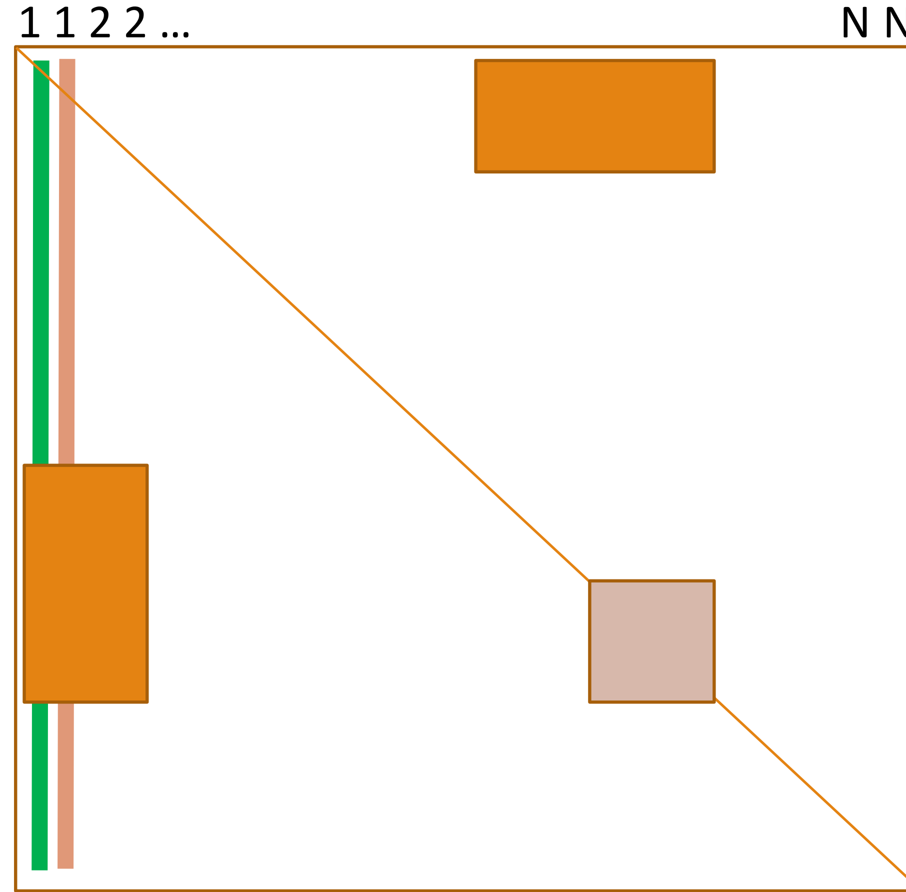


23,80s (« FILL3 »)
↓
16,01s (« FILL9 »)
↓
13,93s (« FILL12 »)

$$\begin{bmatrix} [A1] - \frac{a_2}{2}[C_{KH}]^T[D][C_{KH}] & [Q] + \frac{b_1}{2}[D][C_{KH}] + \frac{b_2}{2}[C_{KH}]^T[D] \\ [Q]^T + \frac{a_2}{2a_0}[D][C_{KH}] - \frac{a_1}{2a_0}[C_{KH}]^T[D] & [A2] - \frac{b_1}{2a_0}[C_{KH}]^T[D][C_{KH}] \end{bmatrix} \begin{pmatrix} \overline{J} \\ \overline{M} \end{pmatrix} = \begin{pmatrix} \overline{E} \\ \overline{H} \end{pmatrix}$$

Double ACA

When block1 is ACA compressed, we compress block2 at the same time and benefit from mutual intermediate values.



23,80s (« FILL3 »)

↓

16,01s (« FILL9 »)

↓

13,93s (« FILL12 »)

↓

6,17s (« FILL15 »)

Conclusion

- Adaptation of HACApK to our use case
- Universal and specific optimizations
  - Dynamic scheduling (universal), approx. 2x faster than before
  - Exploiting symmetries (use case specific), approx. 4x faster than before
- Experiment:

  454,68s (sequential)=> 23,80s (parallel with dynamic scheduling) => 6,17s (symmetries)

Perspectives

- PEC-only version of bem-hoibc
- Parallelize IPO ?
- GPU-clusters instead of CPU-clusters ?