Construction and profiling of a fast direct solver for distributed finite-element computations

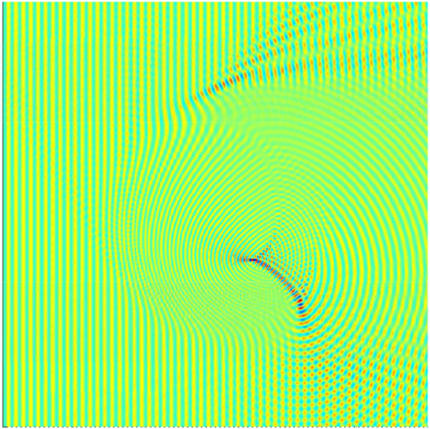
MS67, SIAM PP 2026

Peter Munch[†], Adrianna Gillman[‡]

[†]Institute of Mathematics, Technical University of Berlin, Germany
[‡]Department of Applied Mathematics, University of Colorado Boulder, USA

March 6, 2026

Motivation



Time-harmonic acoustic scattering

$$\Delta u + k^2 u = f.$$

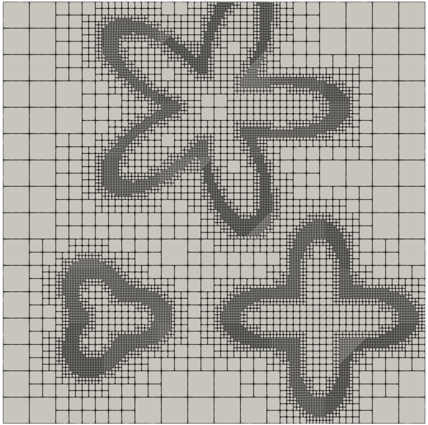
Until now:

- boundary integral methods
- composite spectral collocation methods

Now: finite element method $\rightarrow$ challenge: (direct) solver

Gillman, A., Barnett, A.H. and Martinsson, P.G., 2015. A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media. BIT Numerical Mathematics.
Gillman, A. and Martinsson, P.G., 2014. A direct solver with O(N) complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method. SISC.

Table of contents



Goal: direct solver for adaptive FEM

1. Finite-element computations
2. Direct solvers
3. Implementation details
4. Performance modeling/benchmarking
5. +6. Extensions
7. Outlook & conclusions

Part 1:

Finite-element computations

Poisson problem

Poisson problem:

$$-\text{div}(\nabla u) = f(\boldsymbol{x}) \qquad x \in \Omega,$$
$$u = g_{\text{D}}(\boldsymbol{x}) \qquad x \in \Gamma.$$

The corresponding weak form is: find $u_h \in V_\Omega^h$ such that

$$(\nabla v_h, \nabla u_h)_\Omega = (v_h, f)_\Omega$$

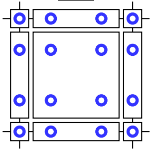
for all $v_h \in V_\Omega^h$. The bilinear form can be interpreted as a matrix. On cell level, we get element matrices (and vectors):

$$M_{ij} = \sum_q \nabla N_i(\tilde{x}_q) \nabla N_j(\tilde{x}_q) |J_q| \times w_q,$$

which are assembled (local to global):

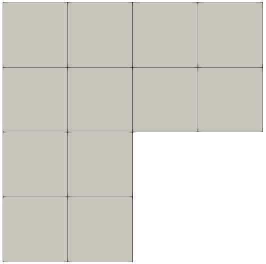
$$M = \sum_c R_c^\top M_c R_c.$$

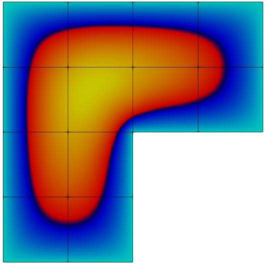
$\mathcal{Q}_3$



dpo: [1, 2, 4]

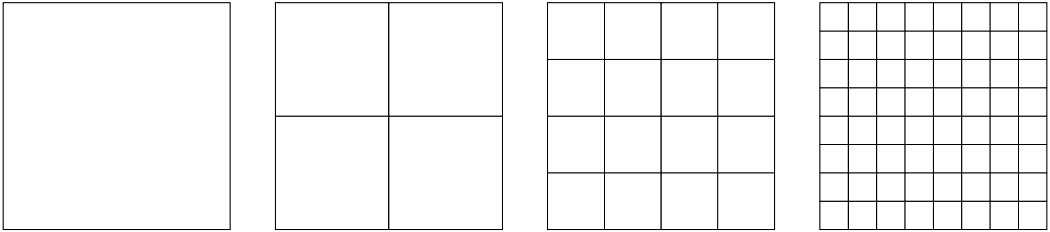
domain/mesh





solution

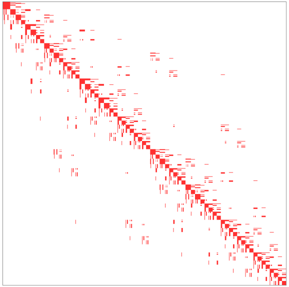
Poisson problem (cont.)



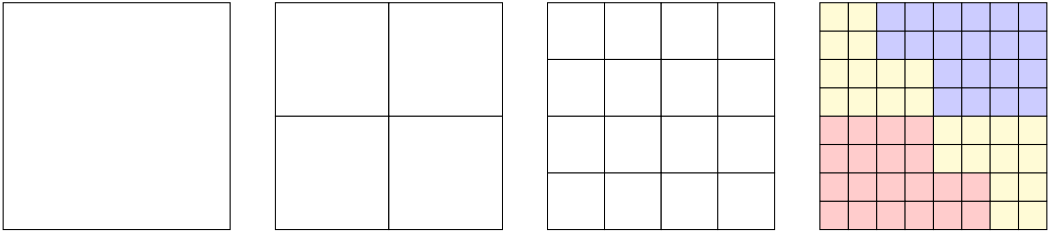
$l = 0$ $l = 1$ $l = 2$ $l = 3$

- forest of trees allows more complicated domains

sparsity pattern (Q_3) $\rightarrow$



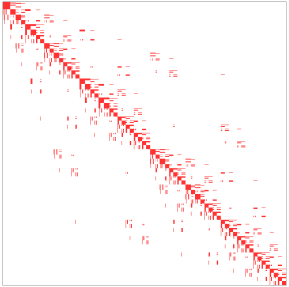
Poisson problem (cont.)



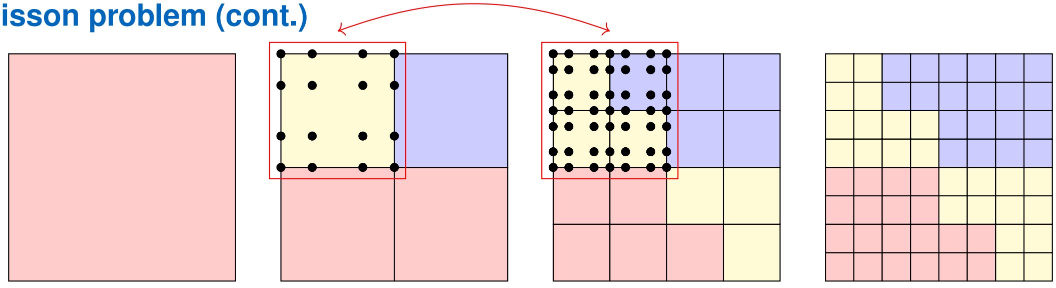
$l = 0$ $l = 1$ $l = 2$ $l = 3$

- forest of trees allows more complicated domains
- parallelization: partitioning of cells

sparsity pattern (Q_3) $\rightarrow$



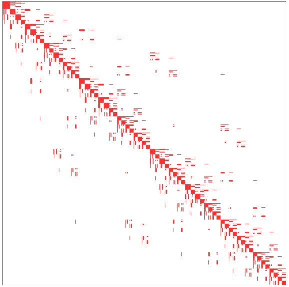
Poisson problem (cont.)



- forest of trees allows more complicated domains
- parallelization: partitioning of cells
- multigrid: partitioning of cells on levels & transfer with

  dpo: $\underbrace{4 \times [1, 2, 4]}_{\text{fine}} \rightarrow \underbrace{[1, 5, 25] \rightarrow [1, 2, 4]}_{\text{coarse}}$

sparsity pattern (Q_3) $\rightarrow$



Part 2:

Direct solvers

Block LU factorization

Direct solver: LU factorization $\to$ forward/backward substitution.
A 2×2 **block matrix** can be decomposed as (Schur complement: $S := D - CA^{-1}B$):

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \underbrace{\begin{bmatrix} I & \\ CA^{-1} & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} I & \\ & S \end{bmatrix} \begin{bmatrix} A & B \\ & I \end{bmatrix}}_{U}.$$

Inverse of M is explicitly given by:

$$M^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}B \\ & I \end{bmatrix} \begin{bmatrix} I & \\ & S^{-1} \end{bmatrix} \begin{bmatrix} I & \\ -CA^{-1} & I \end{bmatrix}.$$

Algorithm:
① (block) forward substitution, ② apply inverse of S, ③ (block) backward substitution.

Block LU factorization

Direct solver: LU factorization $\rightarrow$ forward/backward substitution.
A 2×2 **block matrix** can be decomposed as (Schur complement: $S := D - CA^{-1}B$):

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \underbrace{\begin{bmatrix} I & \\ CA^{-1} & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} I & \\ & S \end{bmatrix} \begin{bmatrix} A & B \\ & I \end{bmatrix}}_{U}.$$

Fast direct solvers for sparse matrices obtained from PDEs,

$$M^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}B \\ & I \end{bmatrix} \begin{bmatrix} I & \\ & S^{-1} \end{bmatrix} \begin{bmatrix} I & \\ -CA^{-1} & I \end{bmatrix},$$

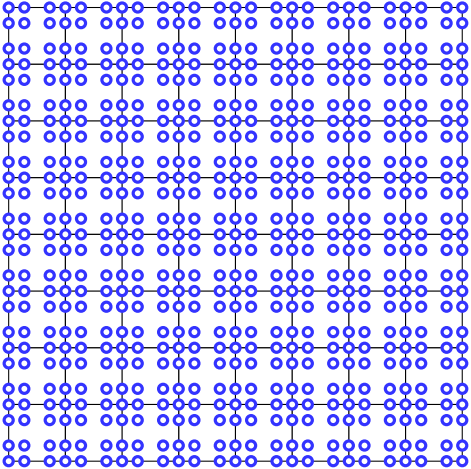
exploit an enumeration of unknowns appropriate for blocking:

- S^{-1} can be applied recursively,
- A^{-1}, CA^{-1}, $A^{-1}B$ are sparse (block) matrices $\Rightarrow$ blocks can be applied independently.

Q: what is an appropriate blocking/pivoting?

Example

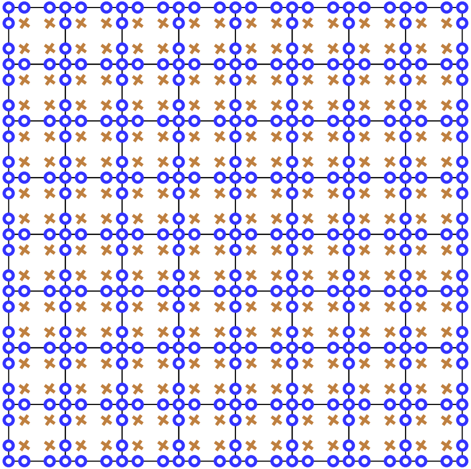
Strategy: nested dissection for 3 refinements and Q_3



l:	3
dofs:	625
dofs (boundary):	–
dofs (inner):	–

Example

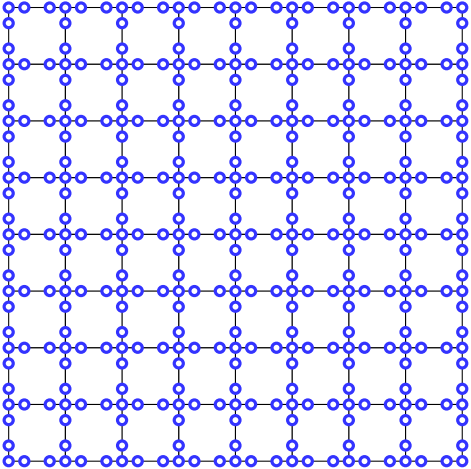
Strategy: nested dissection for 3 refinements and Q_3



l:	3
dofs:	625
dofs (boundary):	369
dofs (inner):	256

Example

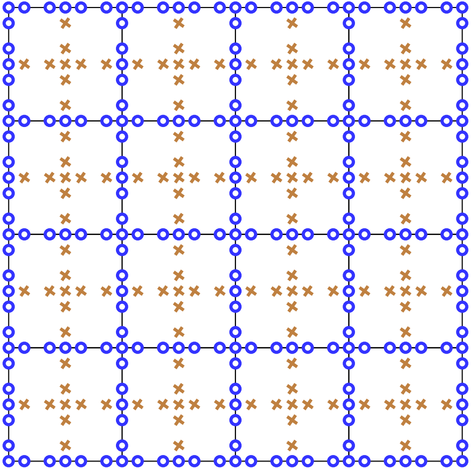
Strategy: nested dissection for 3 refinements and Q_3



l:	3
dofs:	369
dofs (boundary):	369
dofs (inner):	0

Example

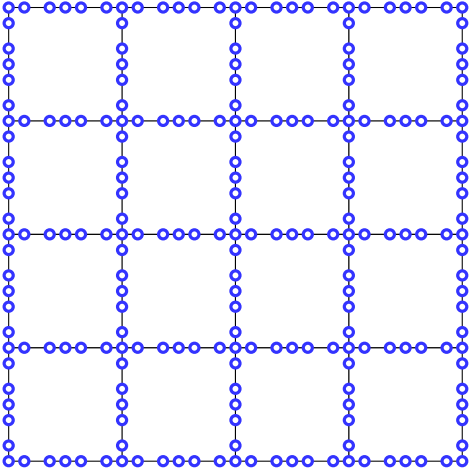
Strategy: nested dissection for 3 refinements and Q_3



l:	2
dofs:	369
dofs (boundary):	225
dofs (inner):	144

Example

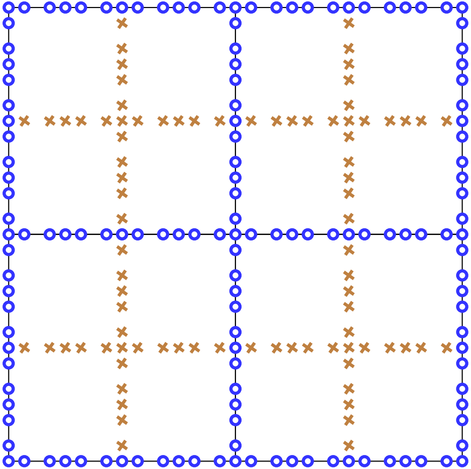
Strategy: nested dissection for 3 refinements and Q_3



l:	2
dofs:	225
dofs (boundary):	225
dofs (inner):	0

Example

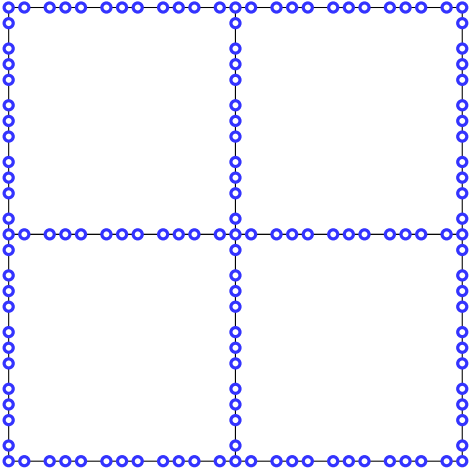
Strategy: nested dissection for 3 refinements and Q_3



l:	1
dofs:	225
dofs (boundary):	141
dofs (inner):	84

Example

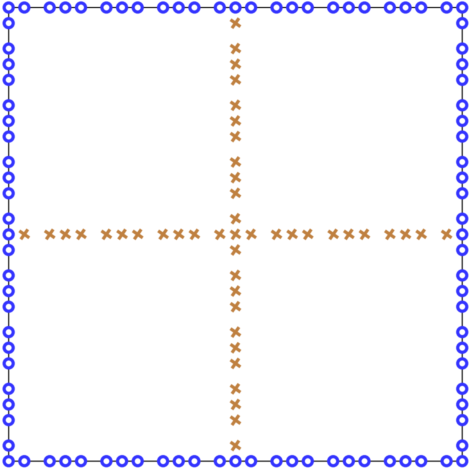
Strategy: nested dissection for 3 refinements and Q_3



l:	1
dofs:	141
dofs (boundary):	141
dofs (inner):	0

Example

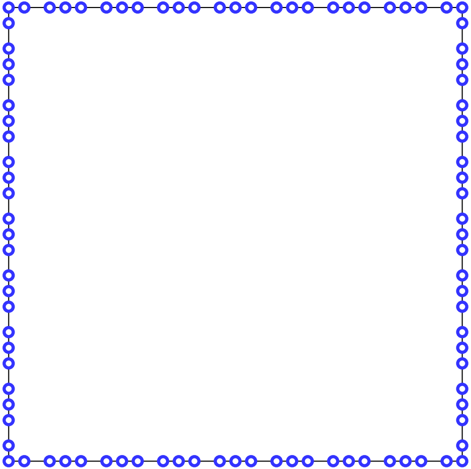
Strategy: nested dissection for 3 refinements and Q_3



l:	0
dofs:	141
dofs (boundary):	96
dofs (inner):	45

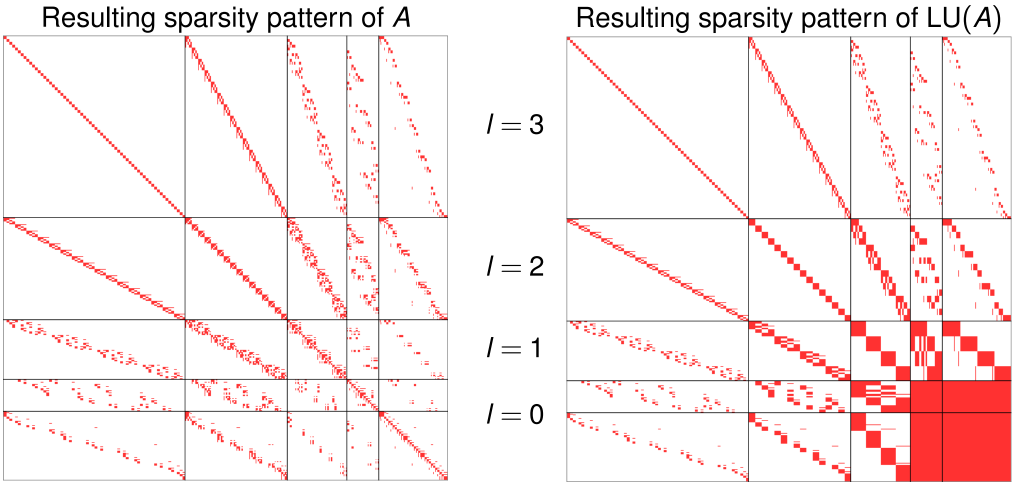
Example

Strategy: nested dissection for 3 refinements and Q_3

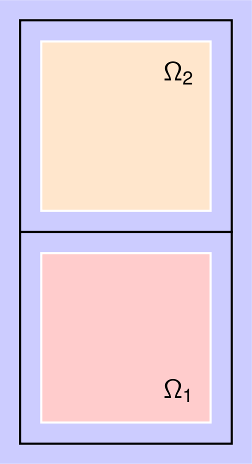


l:	0
dofs:	96
dofs (boundary):	96
dofs (inner):	0

Example (cont.)



Computation of Schur complement: two-cell example



The system gives

$$M = \begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ C_1 & C_2 & D \end{bmatrix}$$

$$\mathrm{LU}(M) = \begin{bmatrix} A_1^{-1} & 0 & -A_1^{-1}B_1 \\ 0 & A_2^{-1} & -A_2^{-1}B_2 \\ -C_1A_1^{-1} & -C_2A_1^{-1} & \mathcal{S}(M)^{-1} \end{bmatrix}$$

with the Schur complement

$$\mathcal{S}(M) = D - C_1A_1^{-1}B_1 - C_2A_2^{-1}B_2.$$

Observation: many terms can be computed independently.

Computation of Schur complement: two-cell example (cont.)

Now, let's assume that $D := D_1 + D_2$ and

$$M_i = \begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix}, \quad \mathrm{LU}(M_i) = \begin{bmatrix} A_i^{-1} & -A_i^{-1} B_i \\ -C_i A_i^{-1} & \text{"}\mathcal{S}(M_i)^{-1}\text{"} \end{bmatrix}, \quad \mathcal{S}(M_i) = D_i - C_i A_i^{-1} B_i.$$

We get:

$$M = \begin{bmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ C_1 & C_2 & D_1 + D_2 \end{bmatrix}, \quad \mathrm{LU}(M) = \begin{bmatrix} A_1^{-1} & 0 & -A_1^{-1} B_1 \\ 0 & A_2^{-1} & -A_2^{-1} B_2 \\ -C_1 A_1^{-1} & -C_2 A_1^{-1} & \mathcal{S}(M)^{-1} \end{bmatrix}$$

with the Schur complement

$$\begin{aligned} \mathcal{S}(M) &= D_1 + D_2 - C_1 A_1^{-1} B_1 - C_2 A_2^{-1} B_2 \\ &= (D_1 - C_1 A_1^{-1} B_1) + (D_2 - C_2 A_2^{-1} B_2) = \mathcal{S}(M_1) + \mathcal{S}(M_2). \end{aligned}$$

Algorithm: for each cell, compute the Schur complement, merge the Schur complements, and invert the sum.

Algorithm: setup process

Subroutine `setupBlockLU` to setup solver to solve $Mx = b$ on $l + 1$ levels:

for $l \leftarrow L$ to 0
  if $l = 0$ **then**
    set up coarse-grid solver based on $\hat{S} \leftarrow \sum_c R_{c,b}^\top S_c R_{c,b}$ ▷ coarsest level (⇔)
  else
    if $l = L$ **then**
      $(I_{c,i}, I_{c,b}) \leftarrow \text{partition}(I_c)$ ▷ finest level
      $(A_c, B_c, C_c, D_c) \leftarrow \text{partition}(M_c)$
    else
      $(I_i, I_b) \leftarrow \text{merge}(I_{c\to 1}, \ldots, I_{c\to n})$ ▷ intermediate levels (⇕)
      $(A_c, B_c, C_c, D_c) \leftarrow \text{merge}(S_{c\to 1}, \ldots, S_{c\to n})$
    end if
    $(A_c^{-1}, -A_c^{-1}B_c, -C_cA_c^{-1}, \underbrace{D_c - C_cA_c^{-1}B_c}_{S_c}) \leftarrow \text{computeBlockLU}(A_c, B_c, C_c, D_c)$
  end if
end for

Algorithm: solution process

Subroutine `applyBlockLU` to solve $Mx = b$ on $l+1$ levels:

if $l = 0$ **then**

$\boldsymbol{x} \leftarrow \hat{S}^{-1}\boldsymbol{b}$ ▷ coarse-grid solver (⇔)

else

$\boldsymbol{b} \leftarrow \boldsymbol{b} + \sum_c R_{c,b}^\top(-C_c A_c^{-1}) R_{c,i}\boldsymbol{b}$ ▷ forward substitution (⇔)

$\boldsymbol{x} \leftarrow \texttt{applyBlockLDU}(l-1, \boldsymbol{x}, \boldsymbol{b})$ ▷ solve Schur complement

$\boldsymbol{x} \leftarrow \boldsymbol{x} + \sum_c R_{c,i}^\top \left[A_c^{-1} \;\; (-A_c^{-1}B_c)\right] \begin{bmatrix} R_{c,i}\boldsymbol{b} \\ R_{c,b}\boldsymbol{x} \end{bmatrix}$ ▷ backward substitution (⇔)

end if

Comments:

- loops over cells/blocks on a level can be parallelized (no dependencies)
- similarity to V-cycle of multigrid

Part 3:

Implementation details

Reference implementation in deal.II

Interface of standard direct solver:

```
TrilinosWrappers::SolverDirect solver;
solver.initialize(matrix);
solver.solve(solution, rhs);
```

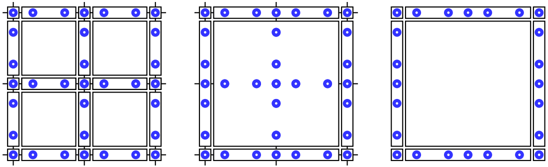
Interface of presented direct solver:

```
NestedDissection solver(/*...*/);
solver.reinit(dof_handler, constraints, compute_element_stiffness_matrix);
solver.vmult(solution, rhs);
```

Features:

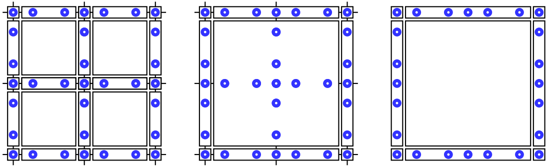
- distributed mesh hierarchy (p4est, multigrid) ⇒ sequence of merging
- coarse-grid solver: direct solver (one cell) vs. sparse direct solver (multiple cells)
- dense-matrix algorithms ⇒ `LAPACK`; sparse-matrix algorithms ⇒ `MUMPS`
- work on distributed vectors
- setup of communication patterns via dynamic sparse communication algorithms
- optimizations: exploiting symmetry, reusing of matrices on levels ⇒ Cartesian mesh, ...
- total: 900 lines of code ⇒ lightweight!

Bookkeeping: merging of indices and matrices



dpo: $4 \times [1, 2, 0] \longrightarrow [1, 5, 9] \longrightarrow [1, 5, 0]$

Bookkeeping: merging of indices and matrices



dpo: $4 \times [1, 2, 0] \longrightarrow [1, 5, 9] \longrightarrow [1, 5, 0]$

We need to merge (and partition) during setup:

- matrices

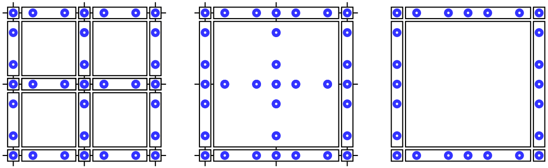
$$(A_c, B_c, C_c, D_c) \leftarrow \text{merge}(S_{c\to1}, ..., S_{c\to n})$$

- indices of child cells

$$(I_i, I_b) \leftarrow \text{merge}(I_{c\to1}, ..., I_{c\to n}).$$

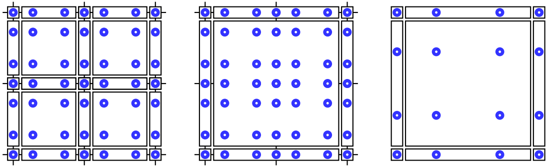
Problem: cells might be distributed in parallel.

Bookkeeping: merging of indices and matrices



dpo: 4 × [1, 2, 0] ⟶ [1, 5, 9] ⟶ [1, 5, 0]

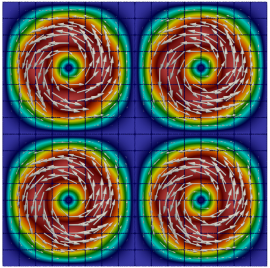
Adaptation of multigrid bookkeeping:

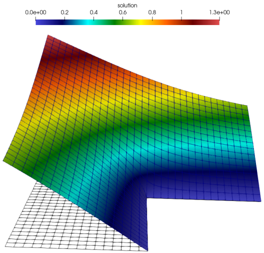


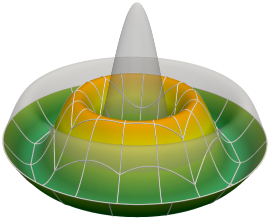
dpo: 4 × [1, 2, 4] ⟶ [1, 5, 25] ⟶ [1, 2, 4]

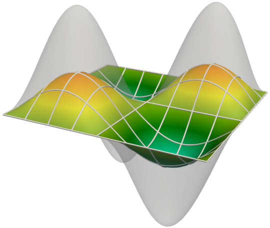
Validation: overview

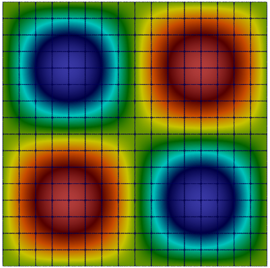
- Poisson problem on L-shaped domain & Kershaw mesh
- Wave equation with RK4
- Heat equation with IRK
- Stokes problem
- CutFEM











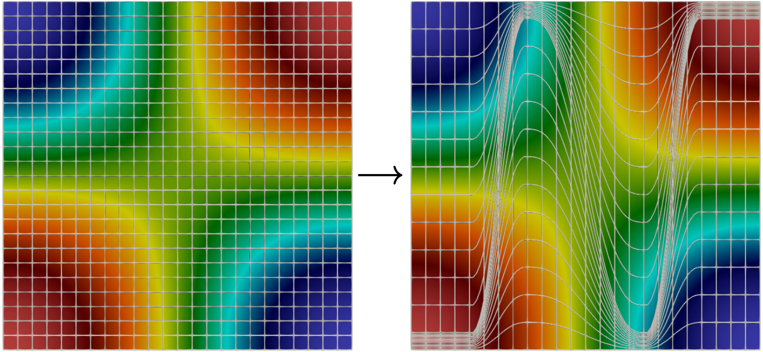
Validation: Poisson problem

Solve Poisson problem on 2D Kershaw meshes $\Omega \in [-0.5, 0.5]^2$. We seek the solution

$$u(x,\ y) = \sin(\pi x)\sin(\pi y),$$

which determines g_D and requires the right-hand-side function $f(x) = 2\pi^2 \sin(\pi x)\sin(\pi y)$.

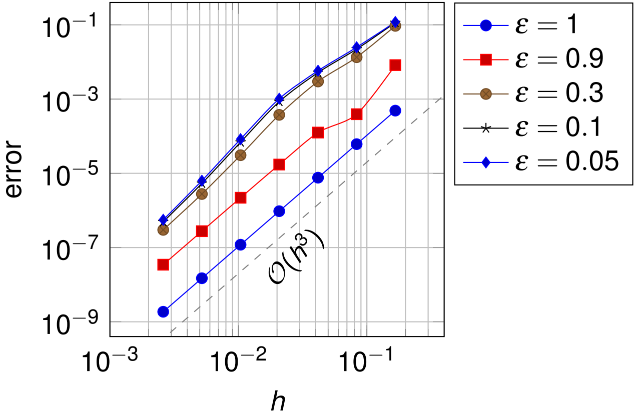
mesh parameter $\varepsilon = 1$ $\longrightarrow$ $\varepsilon = 0.1$



highly anisotropic

Validation: Poisson problem (cont.)

Results ($k = 2$):



Iterative solvers:[1] the number of iterations normally increases with $\varepsilon \downarrow$.

[1] Munch, P. and Kronbichler, M., 2024. Cache-optimized and low-overhead implementations of additive Schwarz methods for high-order FEM multigrid computations. IJHPCA.

Part 4:

Performance modeling & benchmarking

Dense linear-algebra building blocks

Setup and solution consist of dense linear-algebra building blocks with the costs:

- setup: $\sim n^3$
  - LU decomposition of A $\sim 2n^3/3$
  - matrix inversion of A
    1. LU decomposition $A = LU$ $\sim 2n^3/3$
    2. invert U $\sim n^3/3$
    3. invert L $\sim n^3/3$
  - matrix-matrix multiplication BC $\sim 2mnp$
- solution: matrix-vector multiplications $\sim n^2$

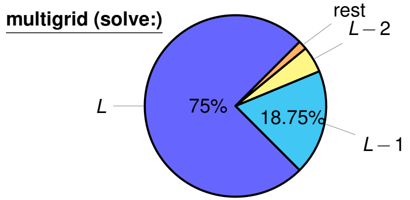
... with $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{m \times n}$, $C \in \mathbb{R}^{n \times p}$

By merging 4 cells in 2D:

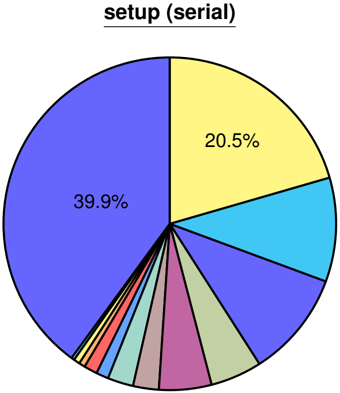
- the number of cells is reduced by factor 4 level by level
- the matrix sizes increase by 2 level by level

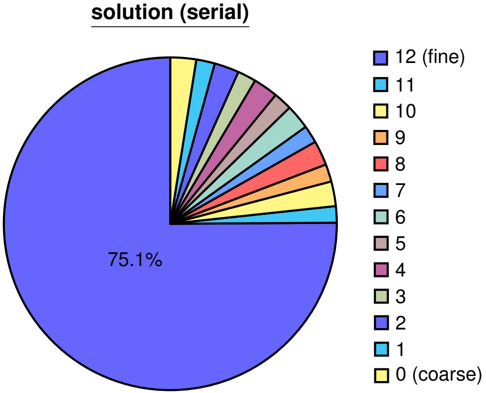
implying constant costs per level during solution.

Serial cost estimates



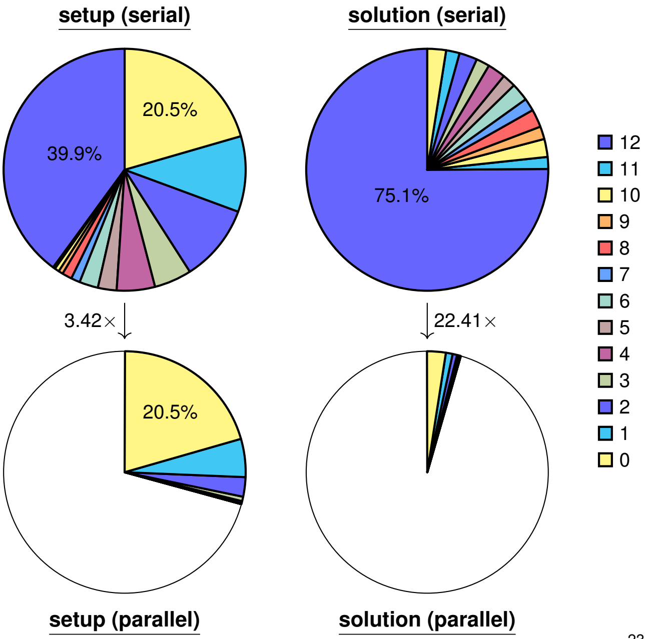
Details: Q_{15}, $4096 = 4^6$ cells with a total of 923,521 DoFs, 2-to-1 merge, DBC





Parallel cost estimates

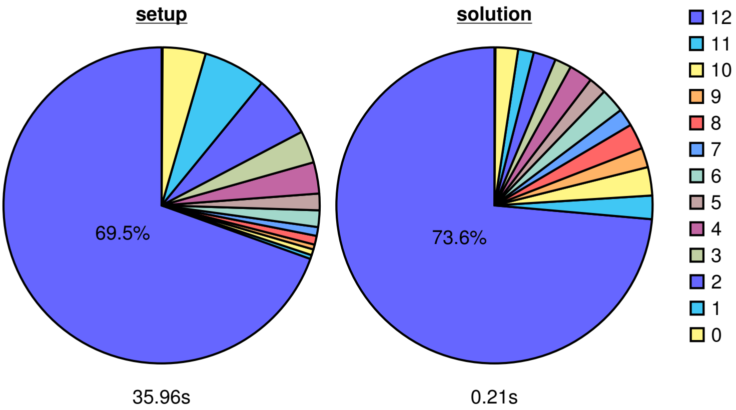
Strategy: distribute cells/blocks



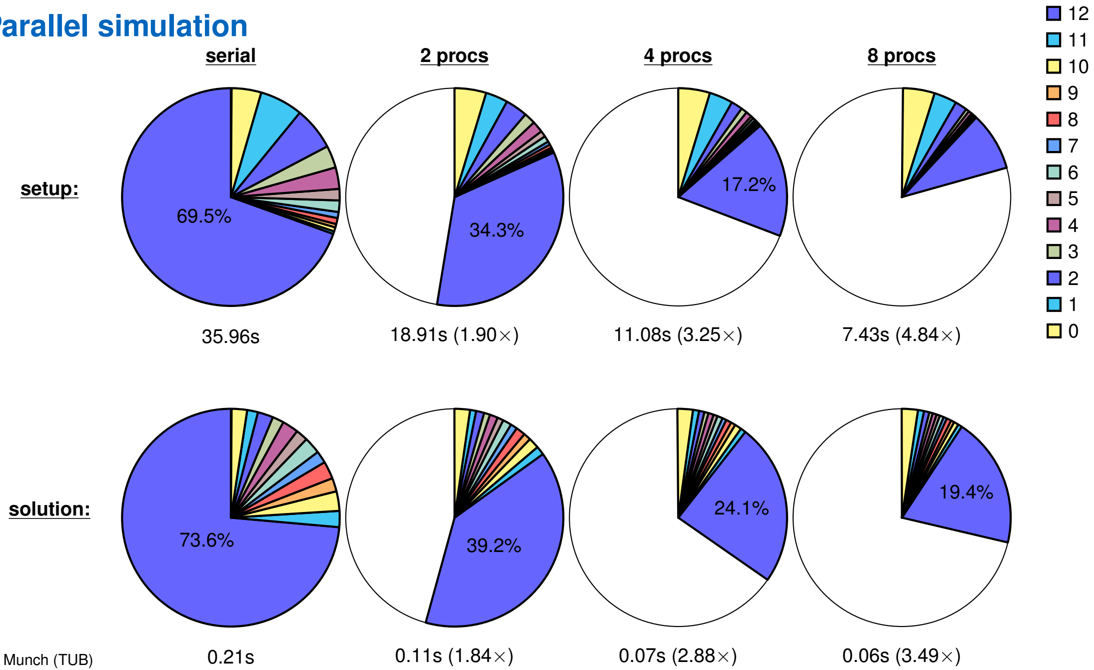
Q: parallelization within blocks?

Serial simulation

workstation: 24 cores, Intel(R) Core(TM) i9-14900



Parallel simulation



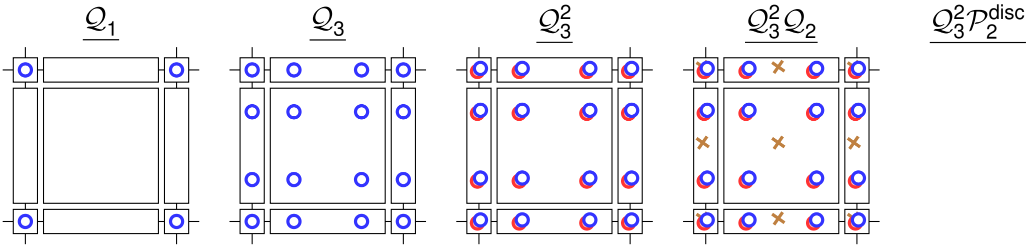
Part 5:

Extension 1+2: different elements and reference cells

Extension 1: different elements

Observation: Merging dpos is extendable to other element types.

Examples:



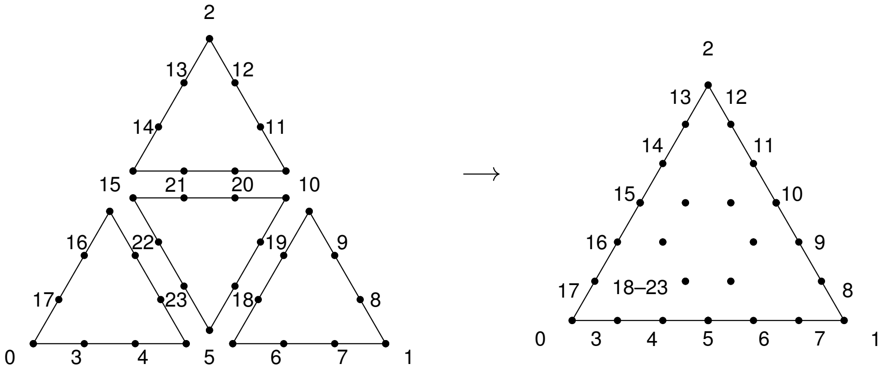
dpo: [1, 0, 0] [1, 2, 4] [2, 4, 8] [3, 5, 9] [2, 4, 14]

Reason: Even though they lead to different sparsity patterns, they result in the same block sparsity pattern determined by dpo.

Extension 2: simplices

Observation: Merging dpos is extendable to other cell shapes.

Example: P_3 on triangles



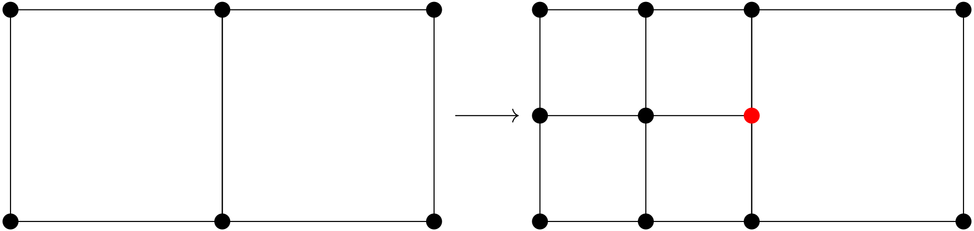
dpo: [1, 2, 0] [1, 5, 6]

Part 6:

Extension 3: adaptivity

Local mesh refinement

Example: 2 coarse cells; scalar, linear Lagrange elements ($k = 1$); non-conformal refinement



- task: guarantee H^1-conformity
- normally via constraint matrix[2]: $x_i = C_{ij} x_j + b_i$

[2] Shephard, M.S., 1984. Linear multipoint constraints applied via transformation as part of a direct stiffness assembly process. IJNME.

Local mesh refinement (cont.)

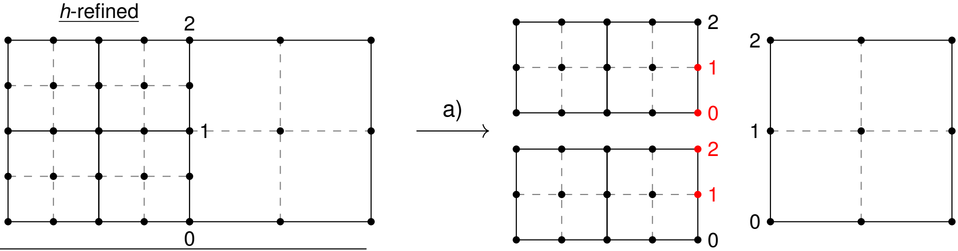
- Option 1: application of constraints as global postprocessing step:

$$A = C^\top \tilde{A} C, \; \tilde{A} = \sum_i R_i^\top A_i R_i$$

- Option 2: application of constraints on cell level[3]

$$A = \sum_i \tilde{R}_i^\top \underbrace{C_i^\top A_i C_i}_{\tilde{A}_i} \tilde{R}_i$$

▷ updated index map $\tilde{R}_i$ and new element matrix $\tilde{A}_i$



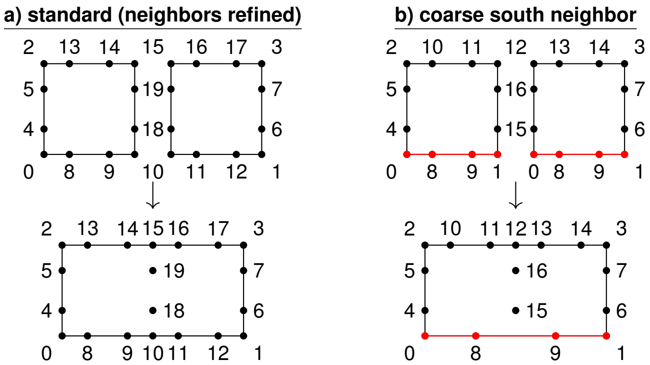
[3] Munch, P., Ljungkvist, K. and Kronbichler, M., 2022. Efficient application of hanging-node constraints for matrix-free high-order FEM computations on CPU and GPU. ISC.

Adapted algorithm

(1) update DoF map and treat $C_i^\top A_i C_i$ as element matrix

(2) loop refinement level by refinement level (similar to: local-smoothing multigrid)

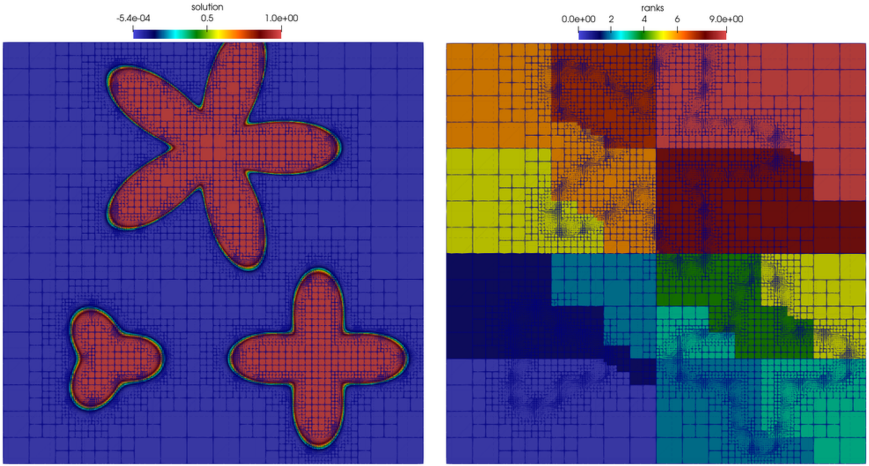
(3) merging of cells leads to different dpos:



extra bookkeeping!

(Preliminary) Results

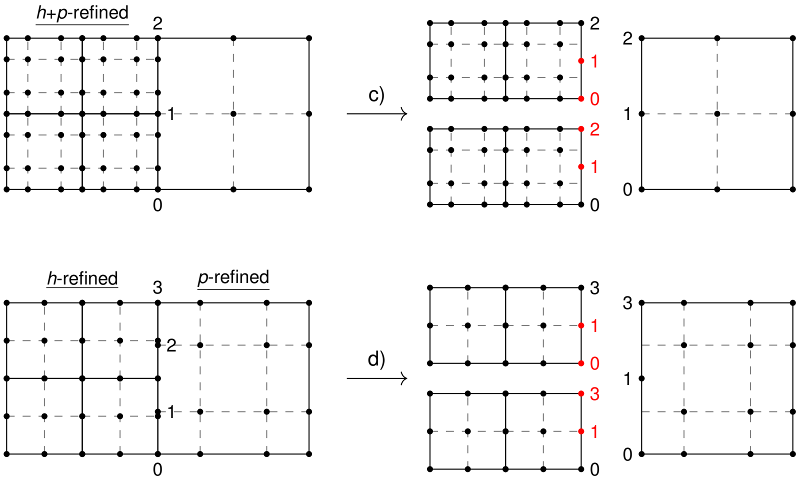
Poisson problem on polar-star geometry:[4]



[4]Chipman, D., Calhoun, D. and Burstedde, C., 2024. A fast direct solver for elliptic PDEs on a hierarchy of adaptively refined quadtrees. arXiv preprint arXiv:2402.14936.

Extension to hp-adaptivity

- *p*-refinement (not shown)
- 2 configurations of *hp*-refinement:



Literature

- Babb, T., Gillman, A., Hao, S. and Martinsson, P.G., 2018. An accelerated Poisson solver based on multidomain spectral discretization. BIT Numerical Mathematics, 58(4), pp. 851-879.
- Geldermans, P. and Gillman, A., 2019. An adaptive high order direct solution technique for elliptic boundary value problems. SIAM Journal on Scientific Computing, 41(1), pp. A292-A315.
- Chipman, D., Calhoun, D. and Burstedde, C., 2024. A fast direct solver for elliptic PDEs on a hierarchy of adaptively refined quadtrees. arXiv preprint arXiv:2402.14936.

Part 7:

Conclusions & outlook

A fast direct solver for finite-element computations

Conclusions:

- direct solver for finite-element computations
- can be fast if the structure of the problem is exploited
- multigrid: similarity in implementation, but different performance characteristics

Outlook:

- extension to 3D
- flux sparsity pattern $\rightarrow$ stabilization and DG
- use fast linear algebra on blocks and improve coarse-level scalability $\rightarrow$ threading?
- low-rank approximation of subblocks $\rightarrow$ preconditioning
- application to time-harmonic acoustic scattering