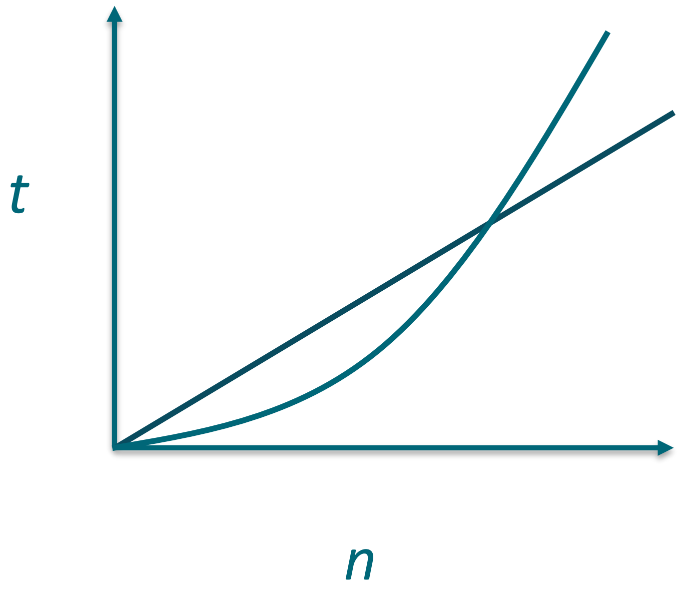
Linear Solvers of Elmer in serial & parallel

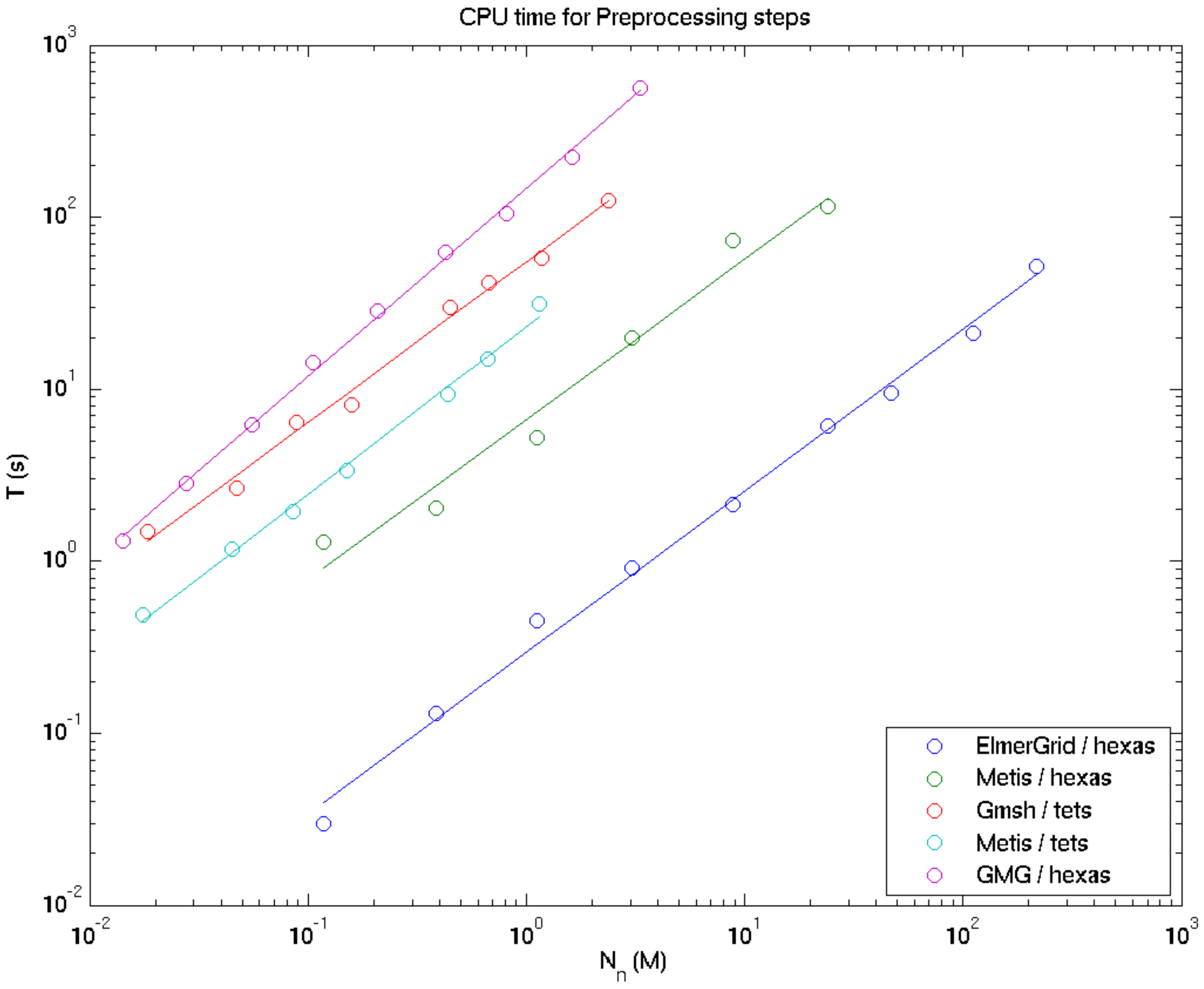
ElmerTeam
CSC

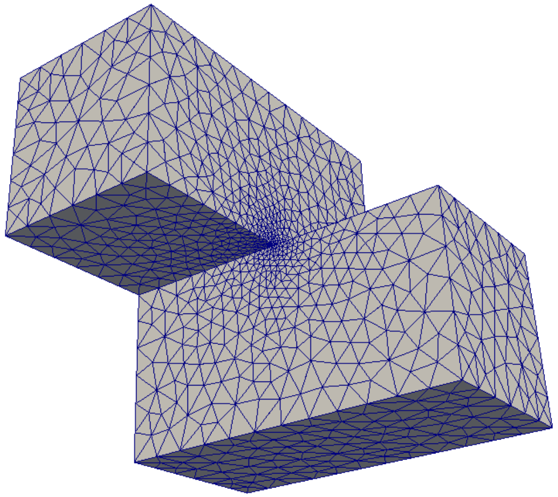
Algorithm scalability

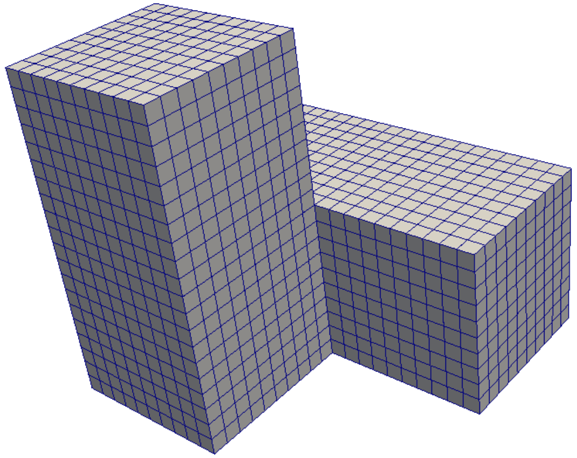
- Before going into parallel computation let's study where the bottle-necks will appear in the serial system
- Each algorithm/procedure has a characteristic scaling law that sets the lower limit to how the solution time *t* increases with problem size *n*
    - The parallel implementation cannot hope to beat this limit systematically
- Targeting very large problems the starting point should be nearly optimal (=linear) algorithm!



CPU time for serial pre-processing and solution

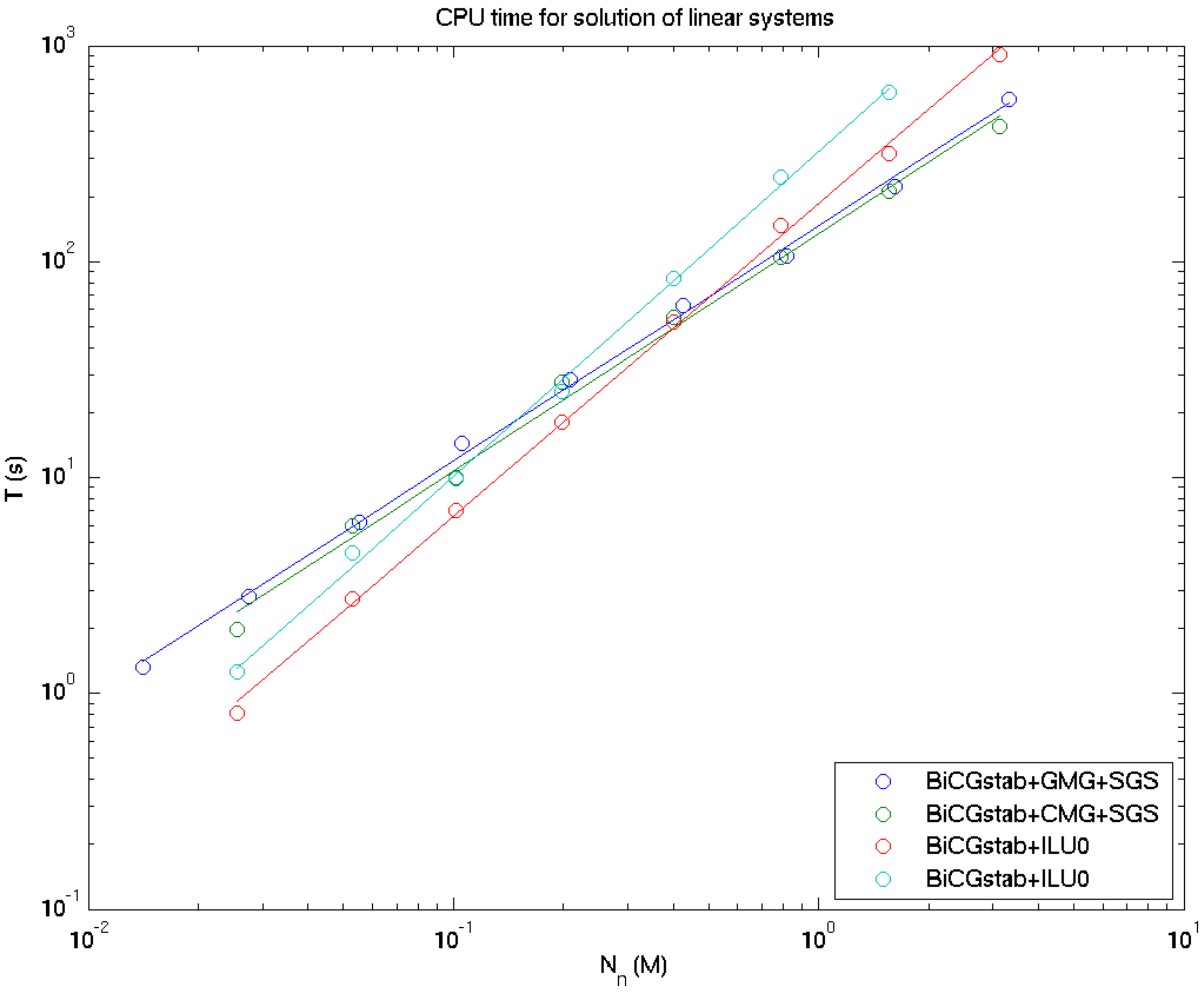


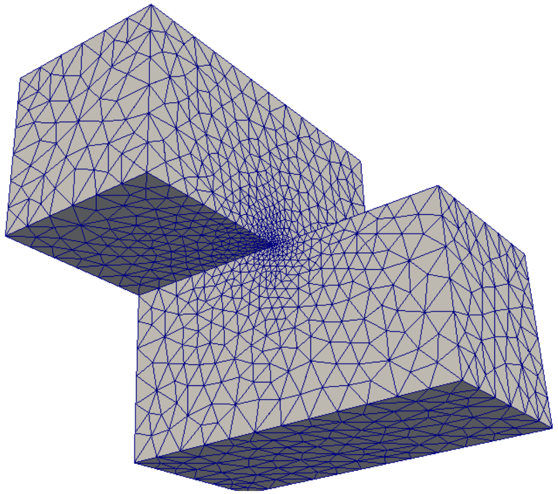


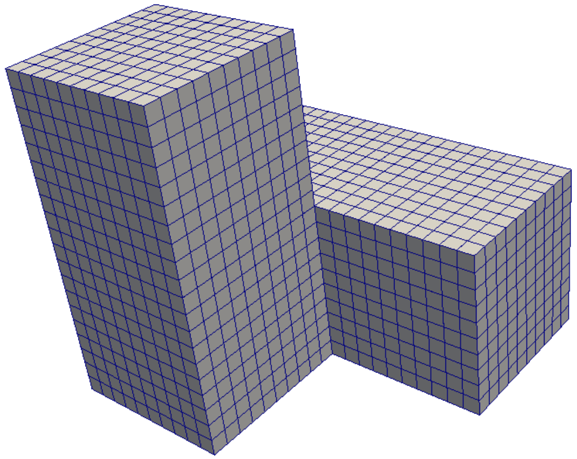


”winkel”

CPU time for serial solution – one level vs. multilevel





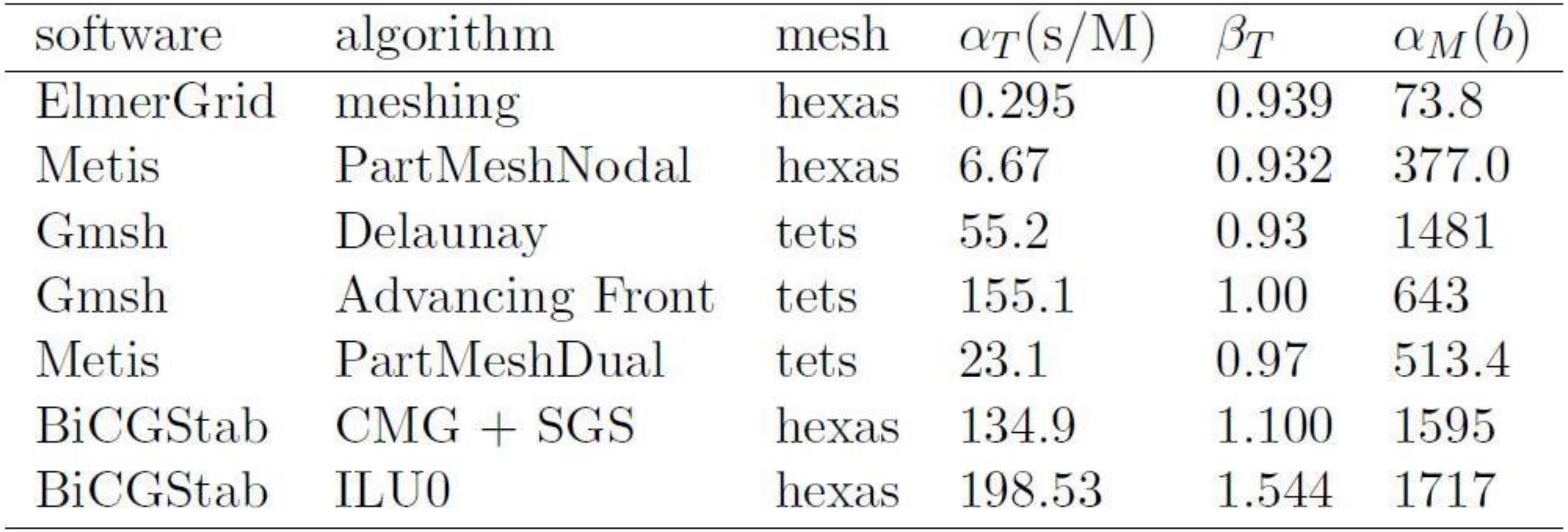


”winkel”

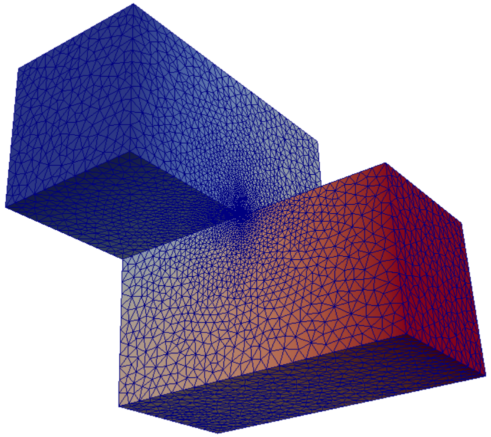
Algorithmic scalability results (old)

Serial performance of different tools and algorithms in terms of CPU time and memory consumption for Poisson equation.

$$t = \alpha n^{\beta}$$



software	algorithm	mesh	α_T(s/M)	β_T	$\alpha_M(b)$
ElmerGrid	meshing	hexas	0.295	0.939	73.8
Metis	PartMeshNodal	hexas	6.67	0.932	377.0
Gmsh	Delaunay	tets	55.2	0.93	1481
Gmsh	Advancing Front	tets	155.1	1.00	643
Metis	PartMeshDual	tets	23.1	0.97	513.4
BiCGStab	CMG + SGS	hexas	134.9	1.100	1595
BiCGStab	ILU0	hexas	198.53	1.544	1717



T(solution) > T(tet meshing) > T(partitioning) > T(hex meshing)

The solution is the first bottleneck even for simple equations, for complex equations and transient problems even more so!

Poisson equation at "Winkel"

- Success of various iterative methods determined mainly by preconditioning strategy
- Best preconditioner is clustering multigrid method (CMG)
- For simple Poisson almost all preconditioners work reasonable well
- Direct solvers differ significantly in scaling
- For vector valued problems number of possible strategies increases due to various splitting techniques
  - ○ Monolithic vs. segregated methods

Linear solver	alpha	beta
BiCGStab+CMG0(SGS1)	178.30	1.09
GCR+CMG0(SGS2)	180.22	1.10
Idrs+CMG0(SGS1)	175.20	1.10
…		
BiCgStab + ILU0	192.50	1.13
…		
CG + vanka	282.07	1.16
Idrs(4) + vanka	295.18	1.16
…		
CG + diag	257.98	1.17
BiCgStab(4) + diag	290.11	1.19
…		
MUMPS(PosDef)	4753.99	1.77
MUMPS	12088.74	1.93
umfpack	74098.48	2.29

Serial linear solvers used with Elmer

We must solve large sparse linear systems: $Ax = b$

Iterative methods

- Internal Krylov methods
  - HUT library: CG, BiCGStab, BiCGStabl, GMRes, TMQMR, QMR
  - Recent additions: GCR, Idrs, BiCGStabl
- Internal Algebraic multigrid
  - Serial AMG and CMG methods (alpha version)
- Hypre
  - Linear solvers
  - Both Krylov methods & BoomerAMG
- Trilinos
- AMGx

Direct methods

- Banded (serial only)
- Umfpack (serial only)
- MUMPS (serial and parallel)
- MKL Pardiso (parallel, not free)

Preconditioning of linear systems

- Instead of solving the original linear system, one may solve the (left) preconditioned system:

$$PAx = Pb$$

  where P is an approximation of the inverse if A
  - ILUn, Incomplete LU depomposition with fill pattern defined by A^n
  - Diagonal precondtioner, *P=1/diag(A)*
  - No strict guidelines on construction, experimental numerics
- *P* may also be considered to an operator
  - Multigrid as precondioner
- The goal of this preconditioned system is to reduce the condition number
  - Results to more robust and faster convergence of linear system
- Typically iterative solution: Krylov method + preconditioner

- Preconditioners in Elmer
  - ILUn, n=0,1,2,3,…
  - ILUt, specific tolerance
  - Diagonal
  - Vanka
  - AMG and AMG

Linear solvers, example

```
Linear System Solver = Iterative
Linear System Iterative Method = "GCR” ! BiCGStab, BiCGStabl, GMRes, Idrs, …
Linear System Max Iterations = 500
Linear System Convergence Tolerance = 1.0E-08
Linear System Abort Not Converged = False
Linear System Preconditioning = "ILU0”  ! ILU0, ILU1, ILU2, ILUT
Linear System ILUt Tolerance = 1.0e-3
Linear System Residual Output = 10
!Idrs Parameter = 4
!BiCGStabl Polynomial Degree = 6

!Linear System Residual Mode = Logical True
!Linear System Robust = Logical True ! Works with GCR and BiCGStabl

! Direct alternative
!Linear System Solver = Direct
Linear System Direct Method = MUMPS ! umfpack
```

Parallel computing concepts

Computer architectures

- Shared memory
  - All cores can access the whole memory
- Distributed memory
  - All cores have their own memory
  - Communication between cores is needed in order to access the memory of other cores
- Current supercomputers **combine** the distributed and shared memory (within nodes) approaches

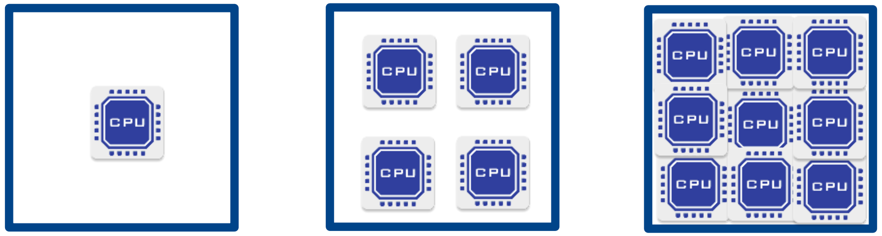
Programming models

- Threads (pthreads, OpenMP)
  - Can be used only in shared memory computer
  - Limited parallel scalability
  - Simpler or less explicit programming
- Message passing (MPI)
  - Can be used both in distributed and shared memory computers
  - Programming model allows good parallel scalability
  - Programming is quite explicit
- Massively parallel FEM codes use typically MPI as the main parallelization strategy

Weak vs. strong parallel scaling

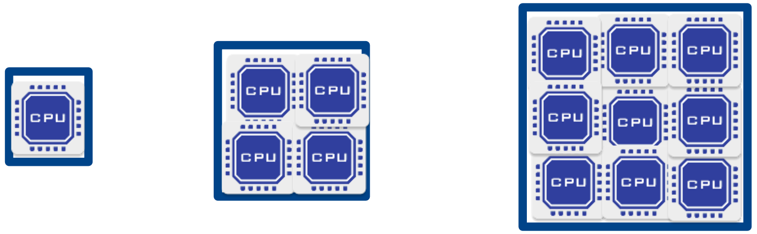
Strong scaling

- How the solution time *T* varies with the number of processors *P* for a fixed total problem size.
- Optimal case: *P* x*T* = *const.*
- A bad algorithm may have excellent strong scaling
- Typically 10^4-10^5 dofs needed in FEM for good strong scaling



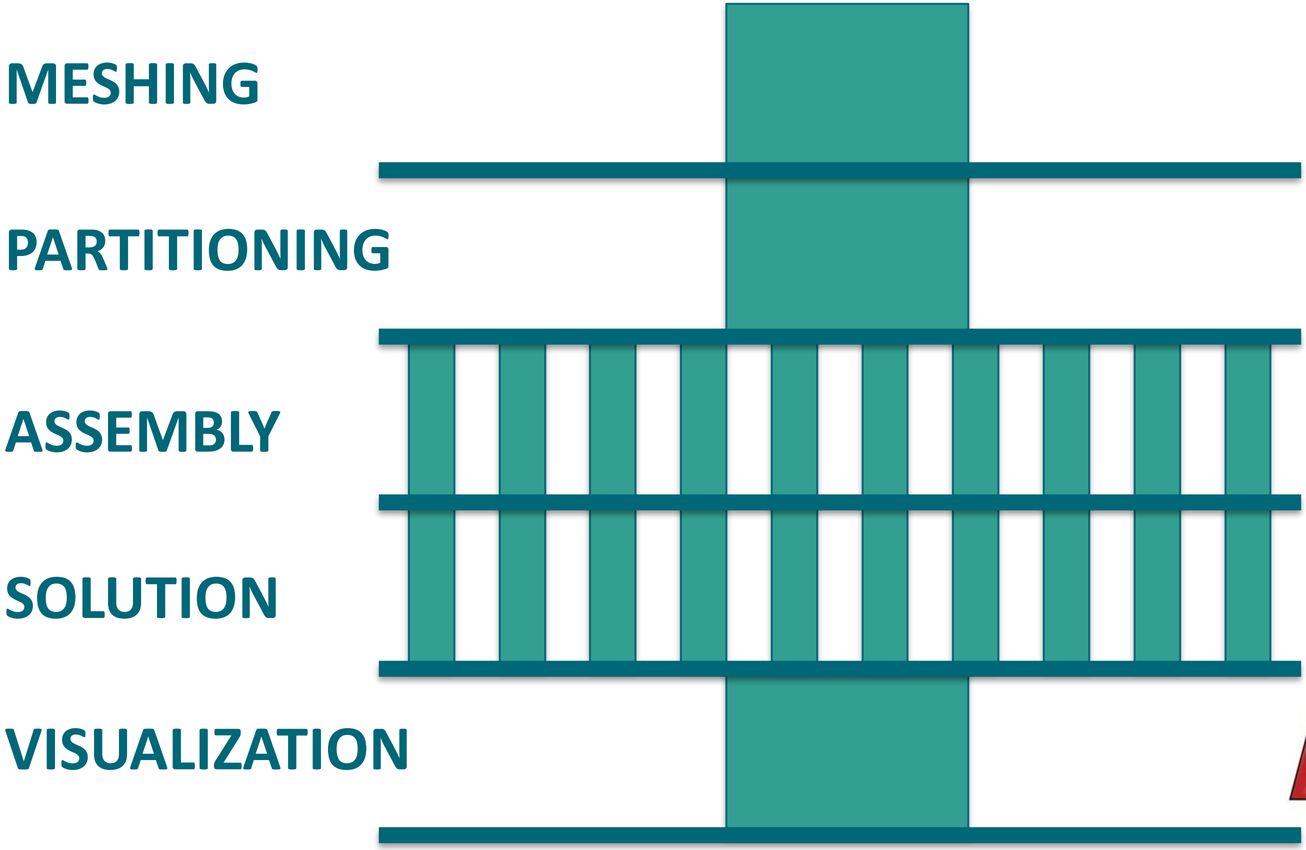
Weak scaling

- How the solution time *T* varies with the number of processors *P* for a fixed problem size per processor.
- Optimal case: *T=const.*
- Weak scaling is limited by algorithmic scaling



Basic Parallel workflow (of Elmer)

- Both assembly and solution is done in parallel using MPI
- Assembly is trivially parallel
- This is the most common parallel workflow



MESHING

PARTITIONING

ASSEMBLY

SOLUTION

VISUALIZATION

Gmsh

NETGEN

ElmerGrid

METIS

Elmer

ParaView

Mesh partitioning with ElmerGrid

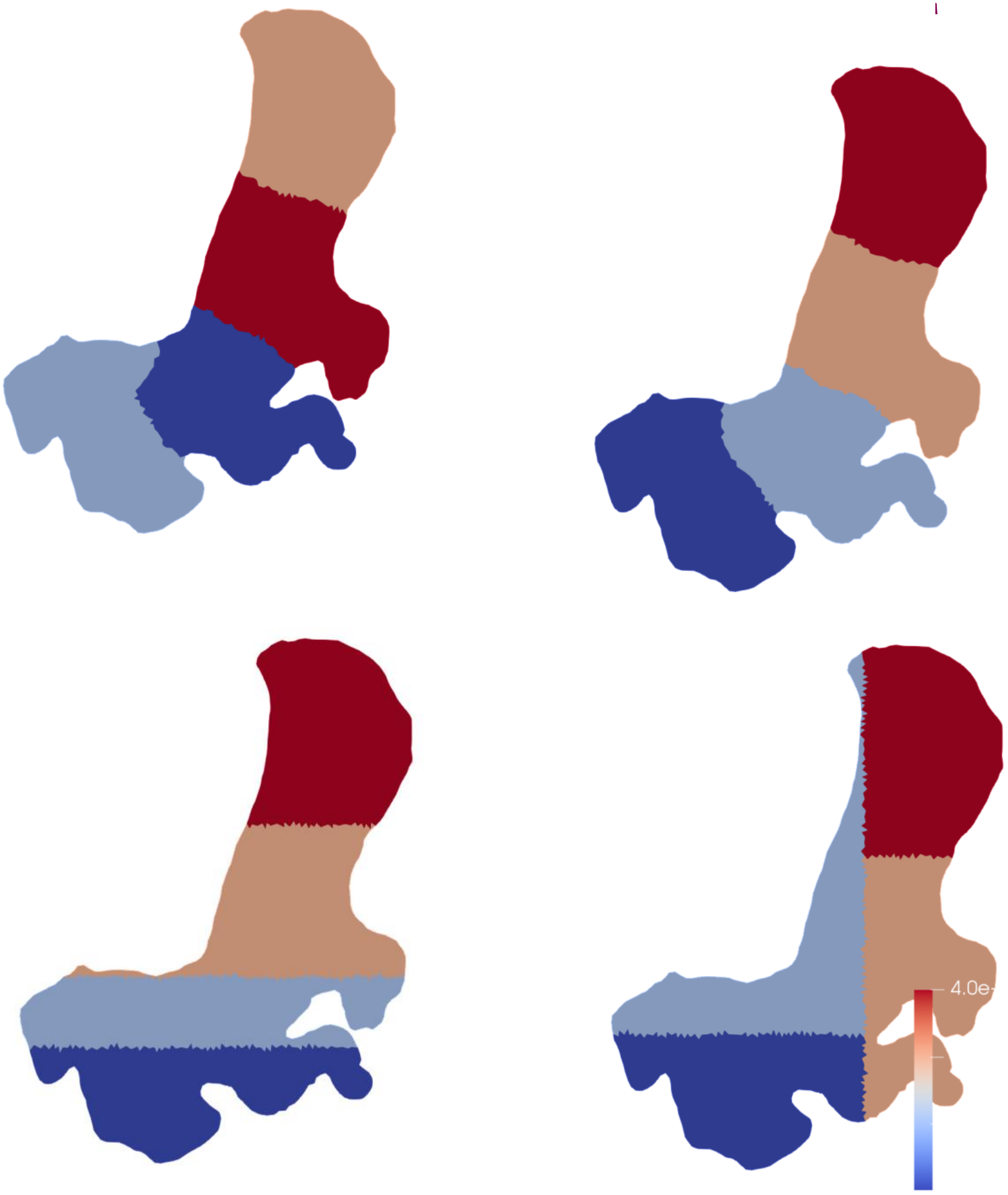
- Two main strategies for mesh partitioning
- **Metis** graph partitioning library:

  `-metiskway #np & -metisrec #np`

  - Generic strategy
  - Includes five different graph partitioning routines fı Metis
- Recursive division by cartesian directions:

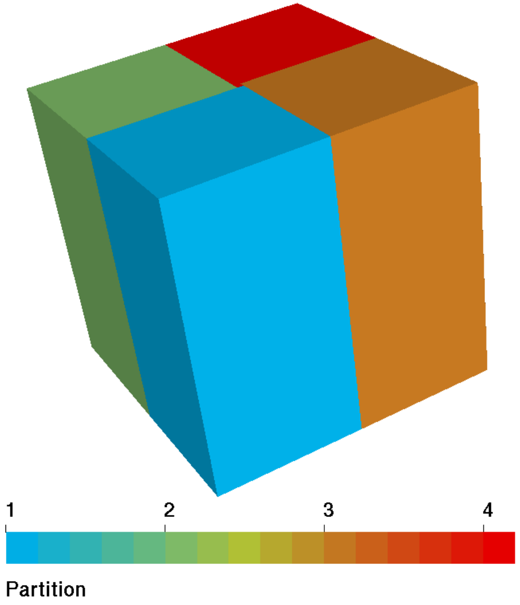
  `-partition nx ny nz`

  - Simple shapes (ideal for quads and hexas)
  - Choice between partitioning of nodes or elements fı

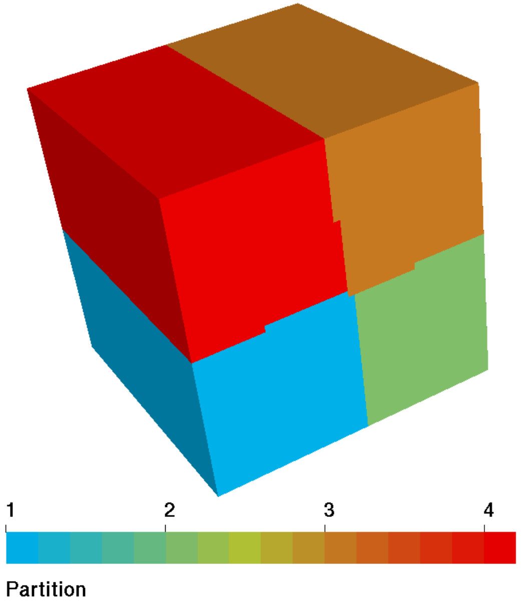


Mesh partitioning with ElmerGrid

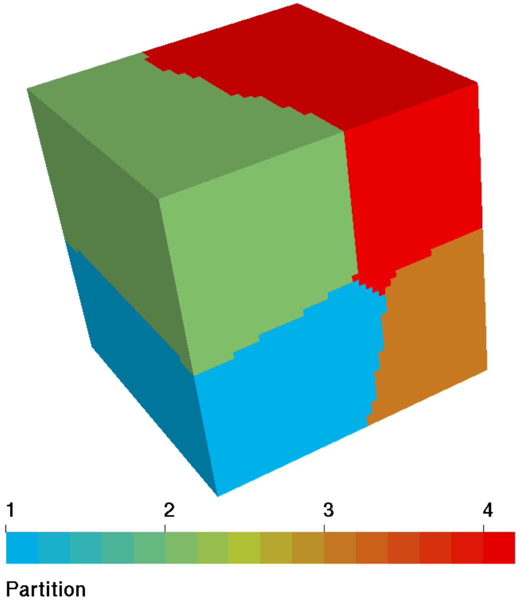
- Optimal partitioning depends on geometry
- To find the best partitioning is a non-trivial task



```
-partition 2 2 1
```



```
-partdual -metisrec 4
```



```
-partdual -metiskway 4
```

ElmerGrid command in parallel

```
Keywords are related to mesh partitioning for parallel ElmerSolver runs:
-partition int[3]     : the mesh will be partitioned in cartesian main directions
-partorder real[3]    : in the 'partition' method set the direction of the ordering
-partcell int[3]      : the mesh will be partitioned in cells of fixed sizes
-partcyl int[3]       : the mesh will be partitioned in cylindrical main directions
-metis int            : mesh will be partitioned with Metis using mesh routines
-metiskway int        : mesh will be partitioned with Metis using graph Kway routine
-metisrec int         : mesh will be partitioned with Metis using graph Recursive routine
-metiscontig          : enforce that the metis partitions are contiguous
-metisseed            : random number generator seed for Metis algorithms
-partdual             : use the dual graph in partition method (when available)
-halo                 : create halo for the partitioning for DG
-halobc               : create halo for the partitioning at boundaries only
-haloz / -halor       : create halo for the the special z- or r-partitioning-halogreedy
…
```

Mesh structure of Elmer

Serial

`meshdir/`

- `mesh.header`
  size info of the mesh
- `mesh.nodes`
  node coordinates
- `mesh.elements`
  bulk element defs
- `mesh.boundary`
  boundary element defs with reference to parents

Parallel

`meshdir/partitioning.N/`

- `mesh.n.header`
- `mesh.n.nodes`
- `mesh.n.elements`
- `mesh.n.boundary`
- `mesh.n.shared`
  information on shared nodes

for each i in [0,N-1]

Serial vs. parallel solution

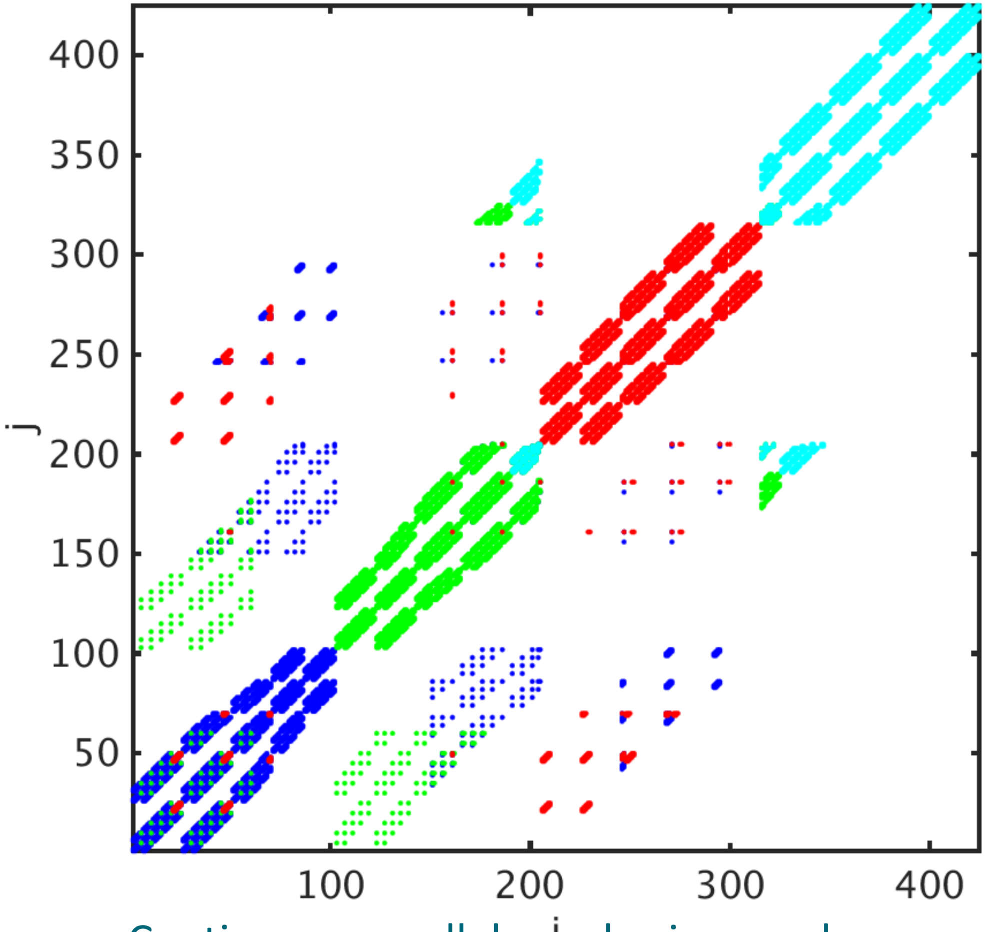
Serial

- Serial mesh files
- Execution with `ElmerSolver case.sif`
- Writes results to one file: `vtu` files

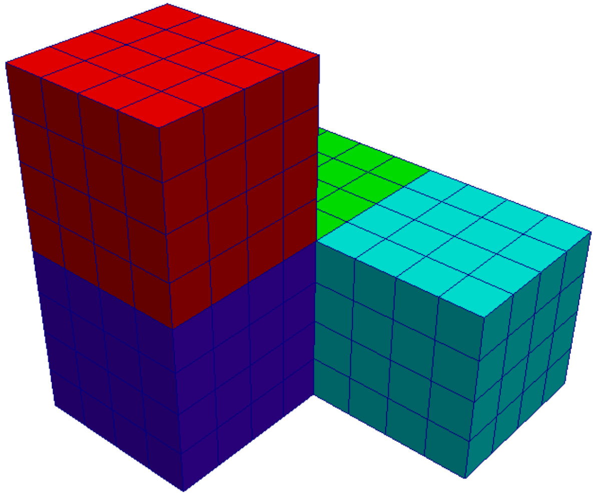
Parallel

- Partitioned mesh files
- Execution with `mpirun -np N ElmerSolver_mpi case.sif`
- Calling convention is platform dependent
- Writes results to *N* `vtu` files + one `pvtu` file

Partitioning and matrix structure

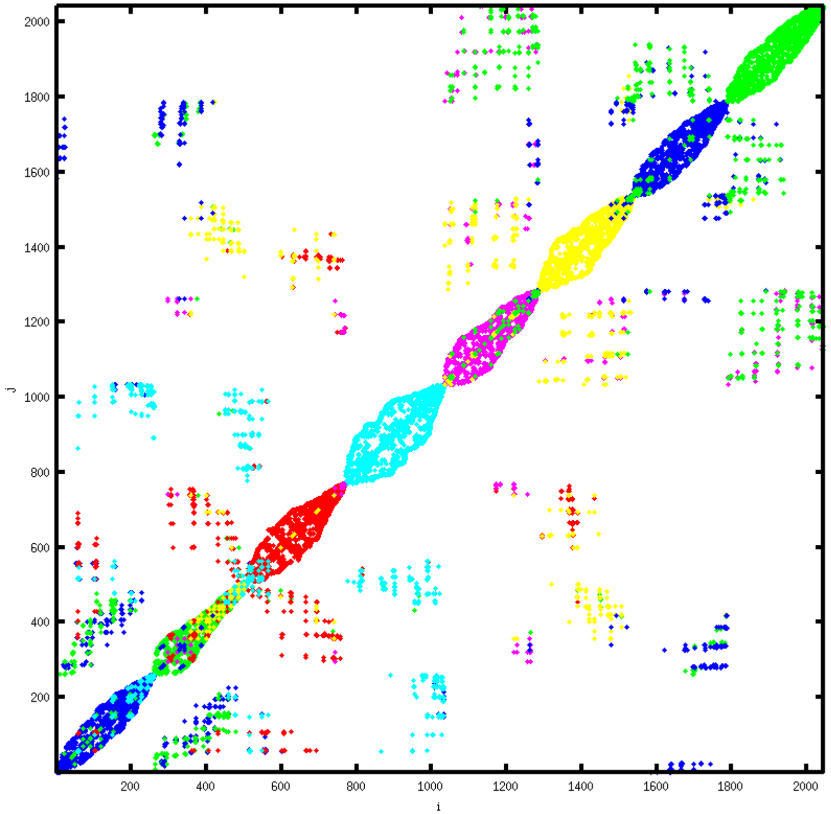


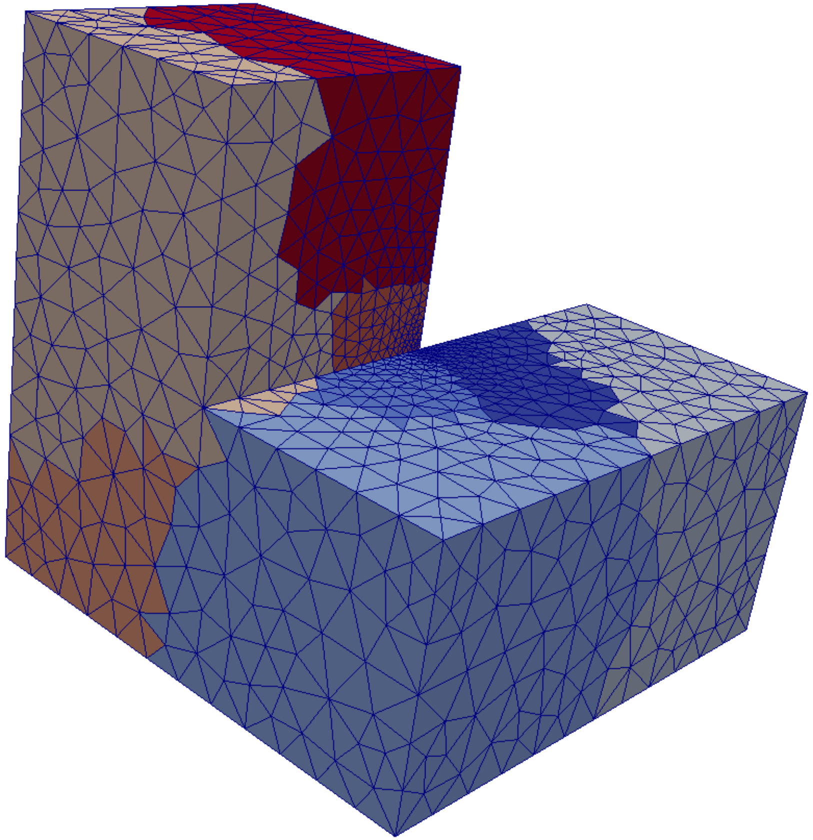
Contiguous parallel numbering used



- Shared nodes result to need for communication.
  - Each dof has just one owner partiotion and we know the neighbours for
  - Owner partition usually handles the full row
  - Results to **point-to-point communication** in MPI
- Matrix structure sets challenges to efficient preconditioners in parallel
  - It is more difficult to implement algorithms that are sequential in nature, e.g. ILU
  - Krylov methods require just matrix vector product, easy!
- Communication cannot be eliminated. It reflects the local interactions of the underlying PDE

Partitioning and matrix structure – unstructured mesh





Metis partitioning into 8

- Partitioning should try to minimize communication
- Relative fraction of shared nodes goes as N^(-1/DIM)
- For vector valued and high order problems more communication with same dof count

Parallel linear solvers used with Elmer

Iterative

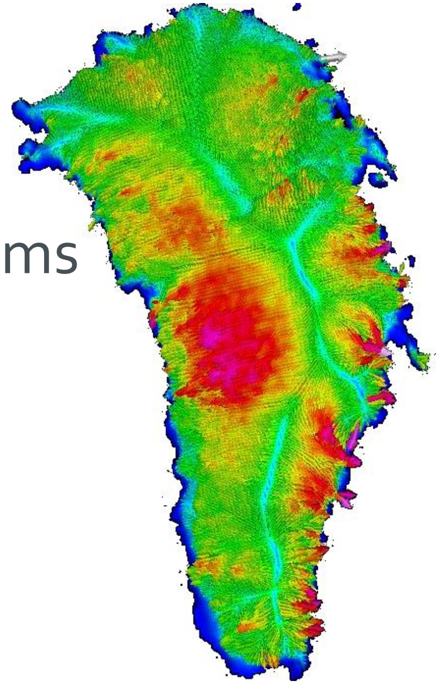
- Internal Krylov methods
  - Usable as in serial
  - ILUn done only partitionwise
- Hypre
  - Krylov solvers
  - Algebraic multigrid: BoomerAMG
  - Truly parallel ILU and Parasails preconditioning
- Trilinos
  - Krylov solvers
  - Algebraic multigrid: ML
  - ...
- FETI
  - Uses MUMPS for local problem

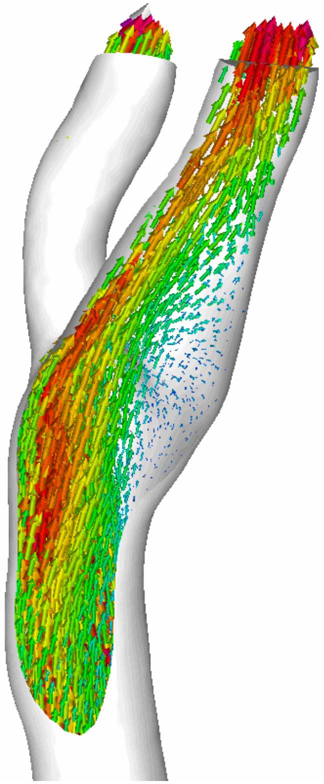
Direct

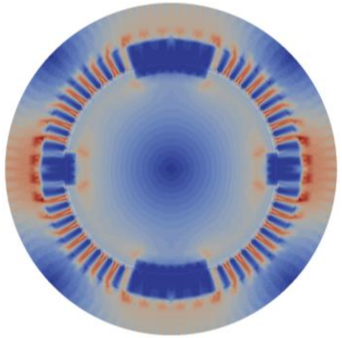
- MUMPS
  - Direct solver that may work when averything else fails
- MKL Pardiso
  - Comes with the Intel MKL library
  - Multihreaded

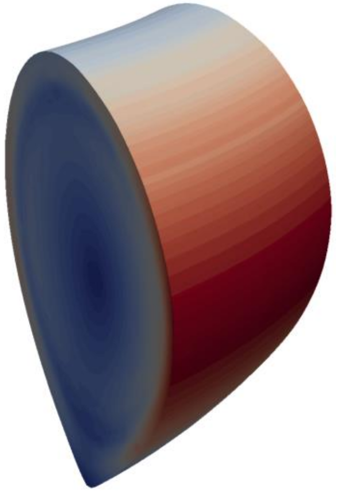
Challenge of real-world problems

- Linear solver libraries work great for many standard problems
  - Scalability demonstrated up to 1000's of cores
- Unfortunately many of the real world cases are
  - Unsymmetric
  - Constrained
  - Compromized in mesh quality (aspect ratio)
  - Etc.
- Often the target number of cores is often rather modest
  - 100's of cores
  - But direct solvers are still too slow or memory intensive
- We look on strategies that split the complex problems into more simple ones where standard libraries excel
  => block precontioning









Block preconditioning

- In parallel runs a central challenge is to have good **parallel preconditioners**
- This problem is increasingly difficult for PDEs with vector fields
  - Navier-Stokes, elasticity, acoustics,…
  - Strongly coupled multiphysics problems
- Preconditioner need not to be just a matrix, it can be a procedure!
- **Idea**: Use as preconditioner a procedure where the components are solved one-by-one and the solution is used as a **search direction** in an outer Krylov method
- Number of outer iterations may be shown to be bounded
- Individual blocks may be solved with optimally scaling methods
  - Multilevel methods

Block precontioning

- Given a block system

$$\begin{bmatrix} \mathbf{K}_{11} & \cdots & \mathbf{K}_{1N} \\ & \cdots & \\ \mathbf{K}_{N1} & \cdots & \mathbf{K}_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \end{bmatrix}$$

- Block Gauss-Seidel

$$\mathrm{P} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{K}_{21} & \mathbf{K}_{22} & \mathbf{0} & \cdots \\ \cdots & & & \end{bmatrix}$$

Block Jacobi

$$\mathrm{P} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{K}_{22} & \mathbf{0} & \cdots \\ \cdots & & & \end{bmatrix}$$

- Preconditioner is the operator which produces the new search direction $s^{(k)}$
- Use GCR to minimize the residual $||\mathbf{b} - \mathbf{K}\mathbf{x}^{(k)}||$ over the space $\mathcal{V}_k = \mathbf{x}^{(0)} + \mathrm{span}\{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(k)}\}$

GCR with general search directions to solve Ku = f

```
$k = 0$
$\mathbf{r}^{(k)} = \mathbf{f} - \mathbf{K}\mathbf{u}^{(k)}$
while $(\|\mathbf{r}^{(k)}\| < TOL\|\mathbf{f}\|$ and $k < m)$
        Generate the search direction $\mathbf{s}^{(k+1)}$
        $\mathbf{v}^{(k+1)} = \mathbf{K}\mathbf{s}^{(k+1)}$
        do $j = 1, k$
            $\mathbf{v}^{(k+1)} = \mathbf{v}^{(k+1)} - \langle \mathbf{v}^{(j)}, \mathbf{v}^{(k+1)} \rangle \mathbf{v}^{(j)}$
            $\mathbf{s}^{(k+1)} = \mathbf{s}^{(k+1)} - \langle \mathbf{v}^{(j)}, \mathbf{v}^{(k+1)} \rangle \mathbf{s}^{(j)}$
        end do
        $\mathbf{v}^{(k+1)} = \mathbf{v}^{(k+1)} / \|\mathbf{v}^{(k+1)}\|$
        $\mathbf{s}^{(k+1)} = \mathbf{s}^{(k+1)} / \|\mathbf{v}^{(k+1)}\|$
        $\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \langle \mathbf{v}^{(k+1)}, \mathbf{r}^{(k)} \rangle \mathbf{s}^{(k+1)}$
        $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \langle \mathbf{v}^{(k+1)}, \mathbf{r}^{(k)} \rangle \mathbf{v}^{(k+1)}$
        $k = k + 1$
end while
```

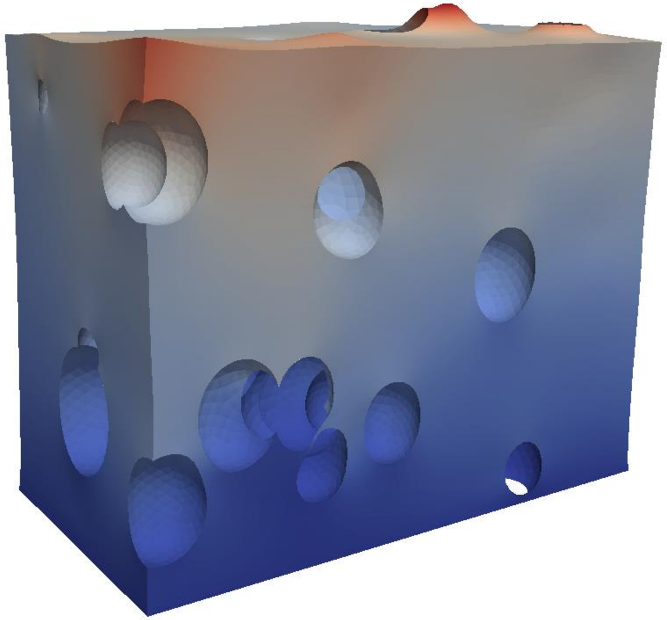
Motivation for using block preconditioner

- Comparison of algorithm **scaling** in linear elasticity between different preconditioners
  - ILU1 vs. block preconditioning (Gauss-Seidel) with agglomeration multigrid for each component
- At smallest system performance about the same
- Increasing size with 8^3=512 gives the block solver scalability of ***O(~1.03)*** while ILU1 fails to converge

	BiCGstab(4)+ILU1		GCR+BP(AMG)	
#dofs	T(s)	#iters	T(s)	#iters
7,662	1.12	36	1.19	34
40,890	11.77	76	6.90	45
300,129	168.72	215	70.68	82
2,303,472	>21,244*	>5000*	756.45	116

Simulation Peter Råback, CSC.

* No convergence was obtained



Stokes problem in computational glaciology

- Stokes equation

$$
\begin{aligned}
-\operatorname{div}[2\eta(\mathbf{D})\mathbf{D}(\mathbf{v})] + \nabla p &= \rho\mathbf{g}, \\
-\operatorname{div}\mathbf{v} &= 0
\end{aligned}
$$

where the strain rate tensor is

$$\mathbf{D} = \mathbf{D}(\mathbf{v}) = 1/2(\nabla\mathbf{v} + \nabla\mathbf{v}^T).$$

- Ice is a shear-thinning fluid that follows the Glen’s flow law

$$\eta = 1/2A^{-k}[I_2(\mathbf{D})]^{(k-1)/2}$$

- Resulting system is very challenging to solve
  - The viscosity variations may be of order 10^5
  - The aspect ratio of the ice may be of order 10^3

Block preconditioner for the Stokes problem

- Each nonlinear step requires solving the Stokes problem

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{P} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{G} \end{bmatrix}$$

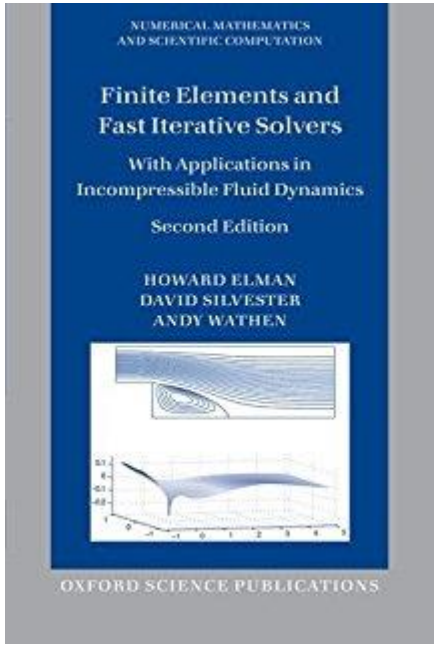
- Note that here C is result of stabilization, with suitable choice of basis vectors it can also be zero. The preconditioner is of the form

$$\mathsf{P} = \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{0} & \mathbf{Q} \end{bmatrix}$$

- An optimal choice of Q corresponds to the Schur complement. Usual choice is

$$\mathbf{Q} = \varepsilon^{-1}\mathbf{M},$$

where M is the mass matrix and ε is the viscosity from previous iteration.



NUMERICAL MATHEMATICS AND SCIENTIFIC COMPUTATION

Finite Elements and Fast Iterative Solvers

With Applications in Incompressible Fluid Dynamics

Second Edition

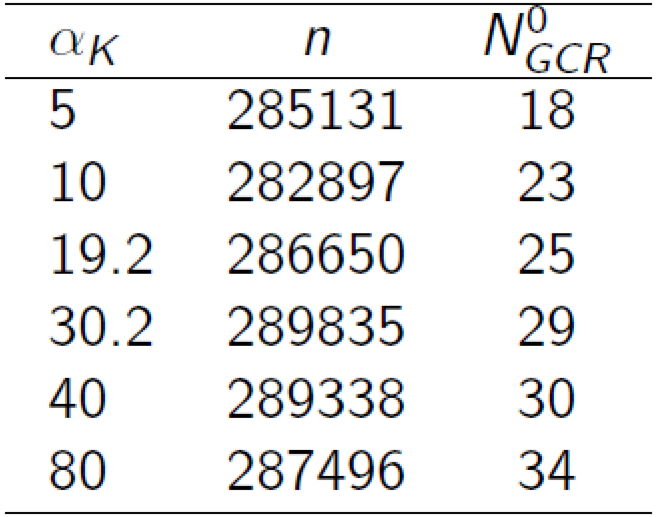
HOWARD ELMAN
DAVID SILVESTER
ANDY WATHEN

OXFORD SCIENCE PUBLICATIONS

H. Elman, D. Silvester, A. Wathen, ***Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*****, OUP Oxford, 2005.**

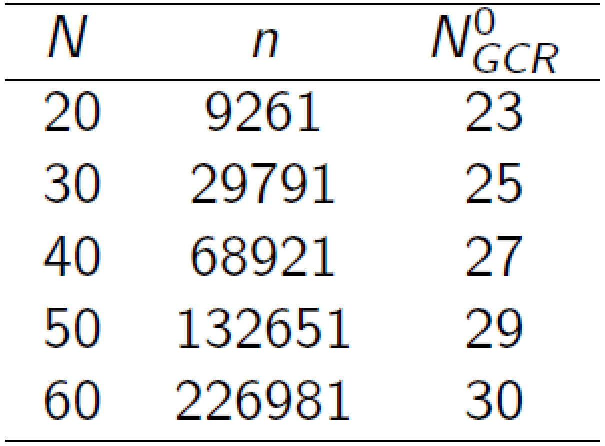
Block preconditioner robustness

- Tested on Midtre Lovenbreen glacier test case



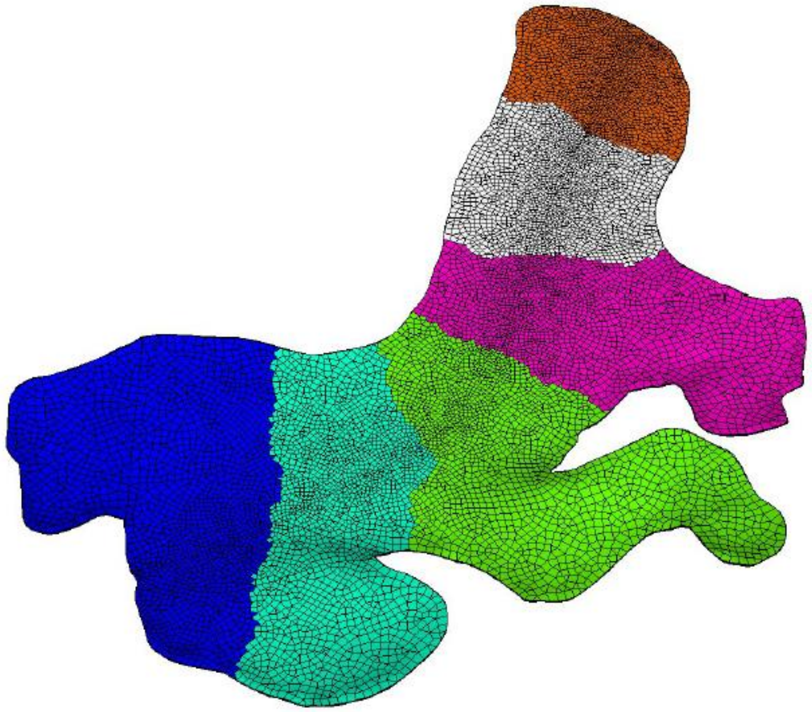
α_K	n	N^0_{GCR}
5	285131	18
10	282897	23
19.2	286650	25
30.2	289835	29
40	289338	30
80	287496	34

Robustness in respect to element aspect ratio α



N	n	N^0_{GCR}
20	9261	23
30	29791	25
40	68921	27
50	132651	29
60	226981	30

Robustness in respect to problem size

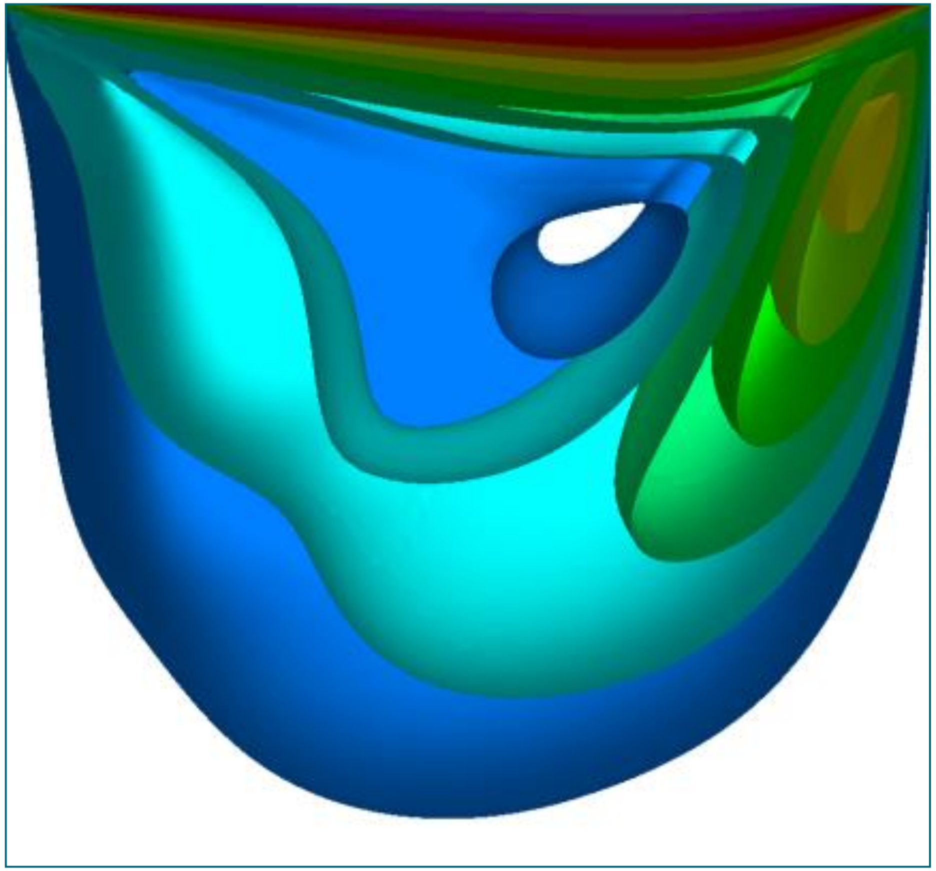


M. Malinen, J. Ruokolainen, P. Råback, J. Thies, T. Zwinger. *Parallel block preconditioning by using the solver of Elmer.* Applied Parallel and Scientific Computing, PARA 2012, Helsinki, Finland, Springer, Heidelberg, 2013; 545-547.

- Number of outer iterations is not too much affected by the problem size of mesh quality.
- Speed of computation determined by the strategy used for individual blocks

Block preconditioner: Weak scaling of 3D driven-cavity

Elems	Dofs	#procs	Time (s)
34^3	171,500	16	44.2
43^3	340,736	32	60.3
54^3	665,500	64	66.7
68^3	1,314,036	128	73.6
86^3	2,634,012	256	83.5
108^3	5,180,116	512	102.0
132^3	9,410,548	1024	106.8



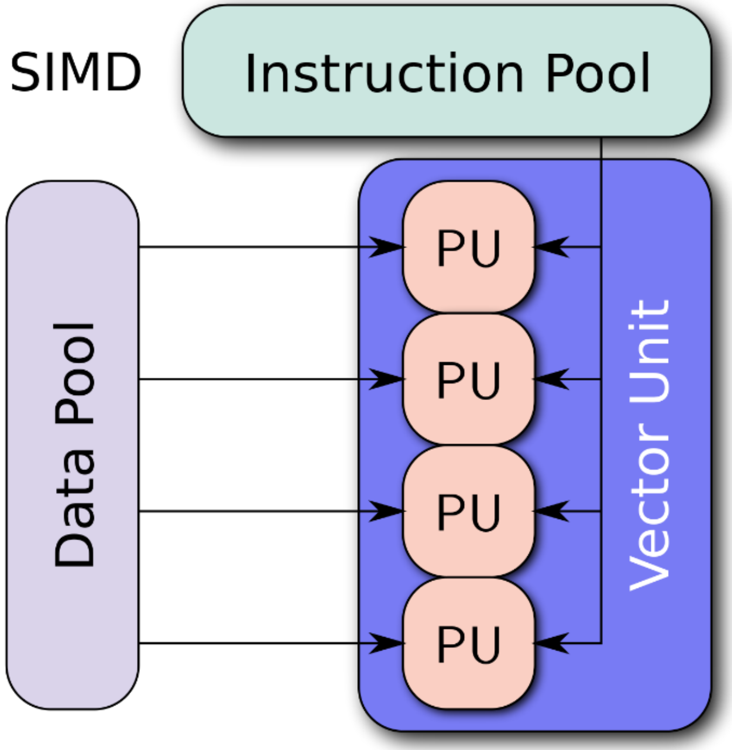
Velocity solves with Hypre: CG + BoomerAMG preconditioner for the 3D driven-cavity case (Re=100) on Cray XC (Sisu). Simulation Mika Malinen, CSC, 2013.

O(~1.14)

Motivation for vectorization

- New computer architectures use SIMD (=vector) units to do fast computations
- If you (on an Intel chip) don't utilize this, you a priori loose ¾ of your performance
- FEM: assembly = creating the matrix

  solution = solving it
- Until recently, assembly procedures in Elmer did not utilize SIMD
  - New Stokes solver does!
  - Gains depend on the number of integration points



By Vadikus - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=39715273

Hybridization of the Finite Element code

- The number of cores in CPUs keep increasing but the clock speed has stagnated
- Significant effort has been invested for the hybrization of Elmer
  - Assembly process has been multithreaded and vectorized
  - "Coloring" of element to avoid race conditions
- Speed-up of assembly for typical elements varies between 2 to 8.
- As an accompanion the multitreaded assembly requires multithreaded linear solvers.

Multicore speedup, P=2
128 threads on KNL, 24 threads on HSW

Element (#ndofs, #quadrature points)	Speedup		Optimized local matrix evaluations / s	
	KNL	HSW	KNL	HSW
Line (3, 4)	0.7	2.0	4.2 M	14.5 M
Triangle (6, 16)	2.5	3.9	2.6 M	6.5 M
Quadrilateral (8, 16)	2.8	4.0	2.6 M	6.6 M
Tetrahedron (10, 64)	7.9	6.3	1.0 M	1.5 M
Prism (15, 64)	8.3	5.8	0.8 M	0.9 M
Hexahedron (20, 64)	7.2	5.8	0.6 M	0.9 M

Speed-up assembly process for poisson equation using 2nd order p-elements. Juhani Kataja, CSC, IXPUG Annual Spring Conference 2017.

Tips for linear solvers

- Direct solvers
  - In 1D always
  - In 2D often very competitive
  - In 3D only if nothing else works
- Iterative solvers
  - BiCGStabl + ”BiCGStabl Polynomial Degree = 4..6”
    - Perhaps the most robust iterative solver without memory problems
  - IDRS + ”Idrs Parameter”
    - Very fast and quite robust
  - GCR
    - Very robust, but cost and memory consumption increases with iteration count
    - Best used when number of iterations is bounded (block preconditioner)
    - Does not require exact preconditioner
- Preconditioners
  - ILUn + ILUt
    - The standard strategy, mind that not the same in parallel
    - Balance higher ”n” with crappier iterative solver
  - Block preconditioner
    - When you aim massively parallel