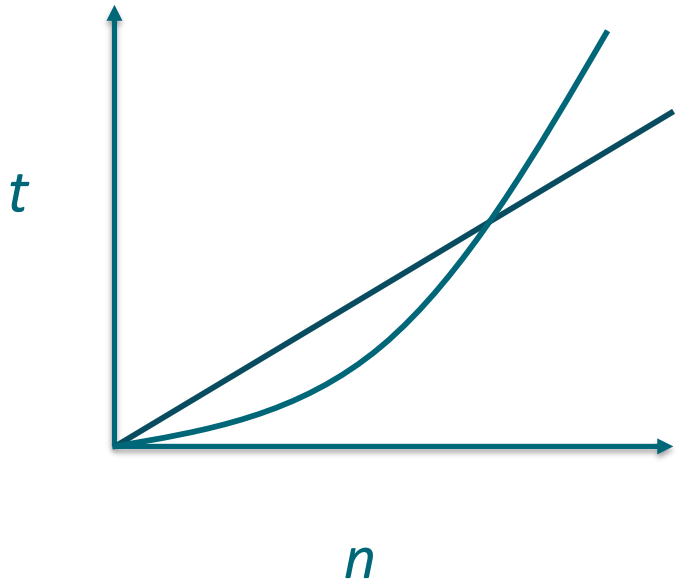
Parallel computing with Elmer

ElmerTeam
CSC – IT Center for Science, Finland

CSC, 2018

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!



Poisson equation at "Winkel"

- Mesh generation is cheapest
- 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 differe significantly in scaling

Mesh generation	alpha	beta
Gmsh	21.4	1.18

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

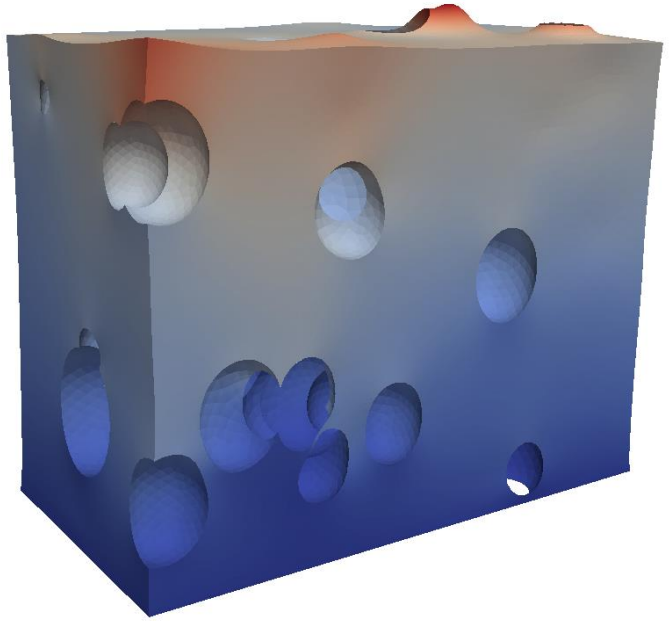
Motivation for using optimal linear solvers

- 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



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

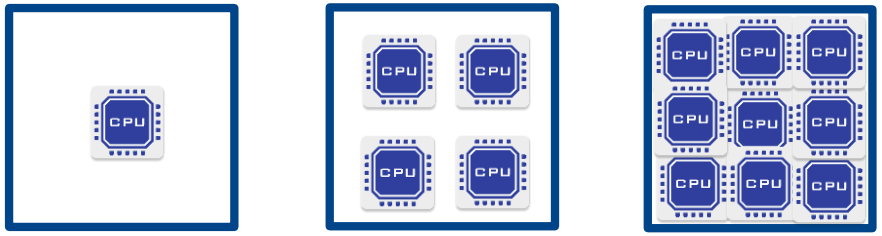
Programming models

- 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
  - As does Elmer!

Weak vs. strong parallel scaling

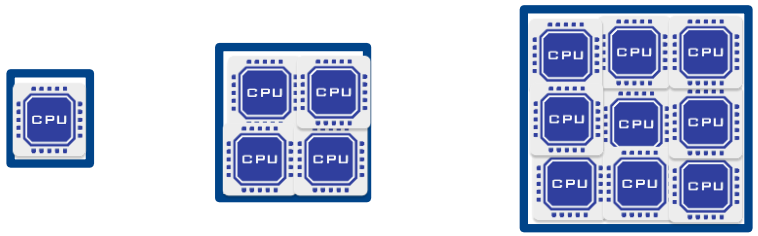
Strong scaling

- 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

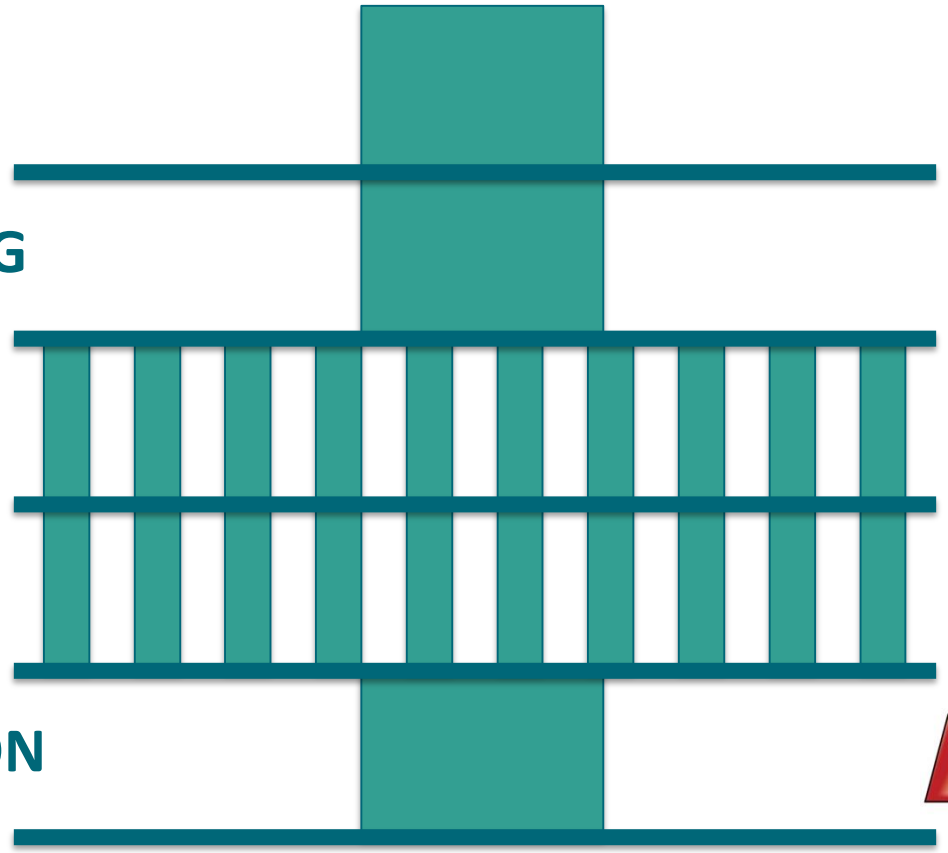


Serial workflow of Elmer

- All steps in the workflow are serial
- Typically solution of the linear system is the main bottle-neck
- For larger problems bottle-necks starts to appear in all phases of the serial workflow

Basic Parallel workflow of Elmer

- Addiational partition step using ElmerGrid
- Both assembly and solution is done in parallel using MPI
- Assembly is trivially parallel



MESHING

PARTITIONING

ASSEMBLY

SOLUTION

VISUALIZATION

Gmsh

NETGEN

ElmerGrid

METIS

Elmer

ParaView

ElmerGrid partitioning commands

Basic volume mesh partitioning options
(geometric partitioning and Metis graph partitiong)

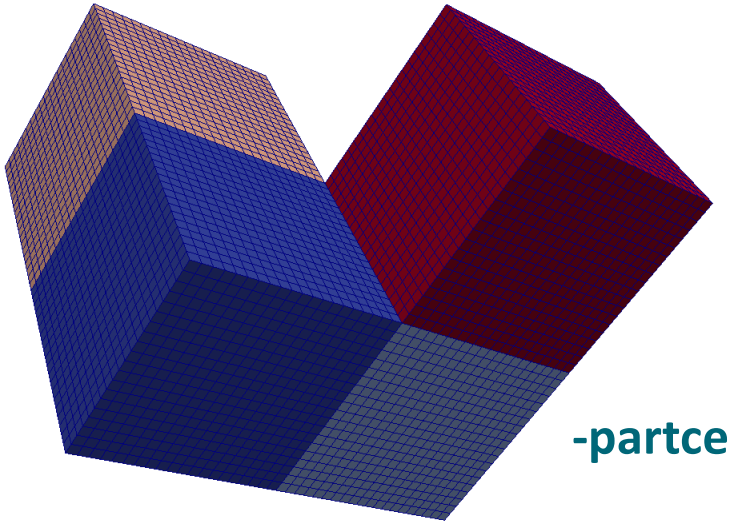
```
-partition int[3]     : the mesh will be partitioned in cartesian main directions
-partorder real[3]    : in the 'partition' method set the direction of the ordering
-partcell int[3]      : the mesh will be partitioned in cells of fixed sizes
-partcyl int[3]       : the mesh will be partitioned in cylindrical main directions
-metis int            : mesh will be partitioned with Metis using mesh routines
-metiskway int        : mesh will be partitioned with Metis using Kway routine
-metisrec int         : mesh will be partitioned with Metis using Recursive routine
-metiscontig          : enforce that the metis partitions are contiguous
-partdual             : use the dual graph in partition method (when available)
```

There are additional flags to control the partitioning of contact boundaries and halo elements.

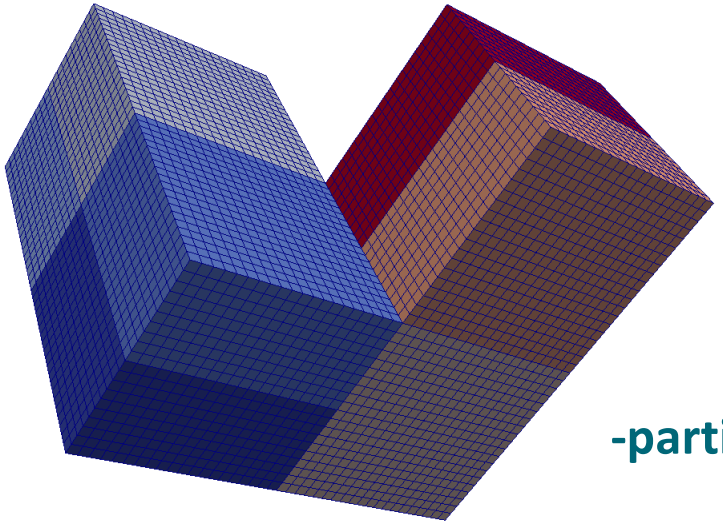
ElmerGrid partitioning examples

- ElmerGrid 2 2 mesh –**partcell n_x n_y n_z**
  - Partition elements in a uniform grid based on the bounding box
  - Number of partitions may be lower than the product if there are empty cells
  - Does not quarantee that partitions are of same size
- ElmerGrid 2 2 mesh –**partition n_x n_y n_z**
  - Partition elements recursively in the main coordinate directions
  - Partitions are of same size
  - Goodness depends heavily on the geometry
- ElmerGrid 2 2 mesh **–metisrec n**
  - Partition elements using a recursive routine of Metis
  - Cannot beat the geometric strategy for some ideal shapes
  - Robust in that partitioning is always reasonable

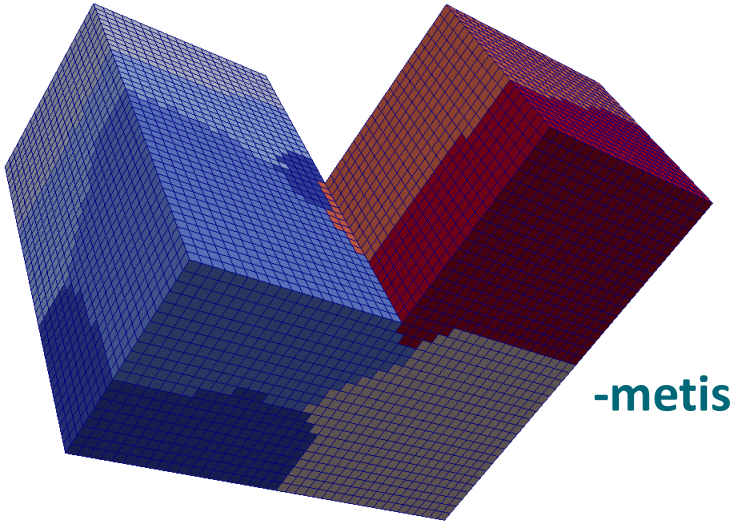
Mesh partitioning with ElmerGrid – structured mesh



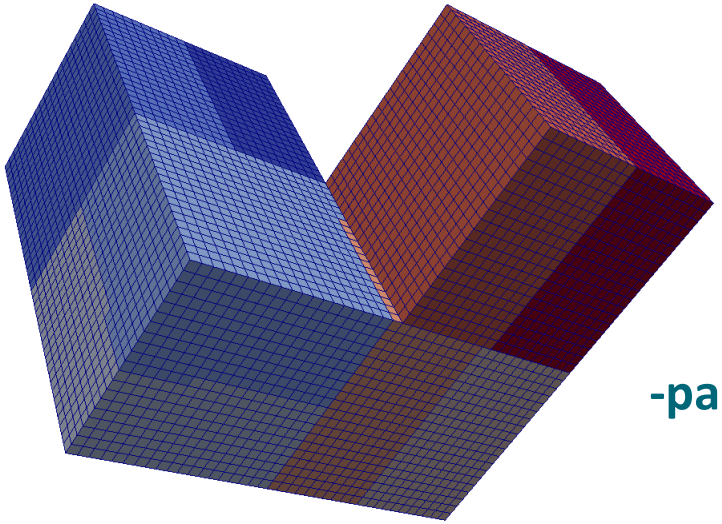
-partcell 2 2 2



-partition 2 2 2

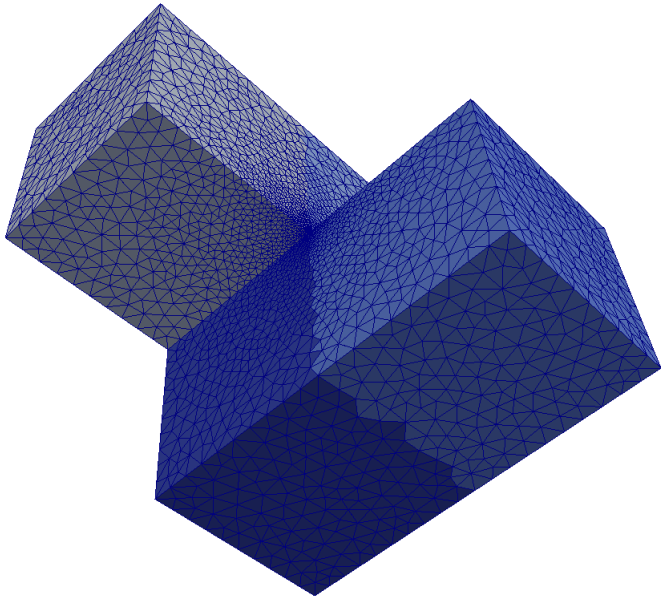


-metis 8

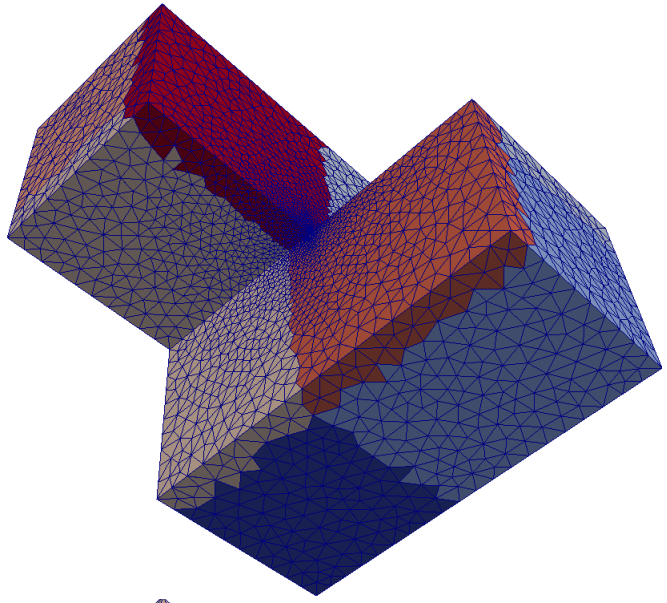


-partdual -metisrec 8

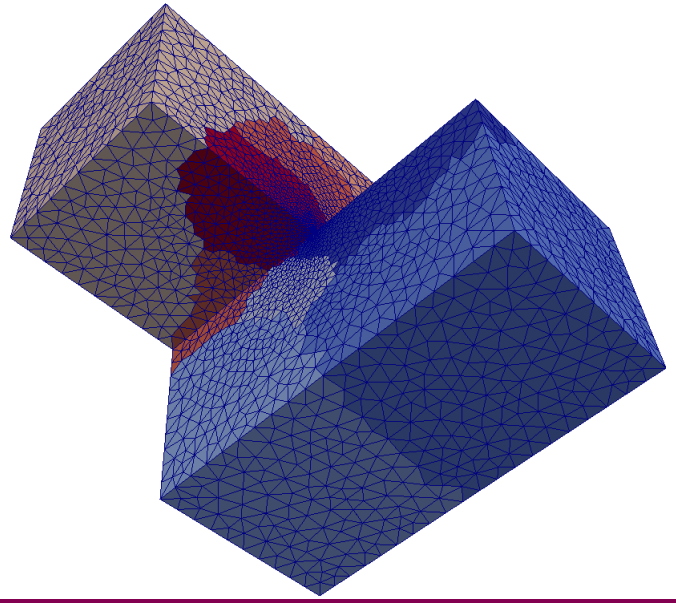
Mesh partitioning with ElmerGrid – unstructured mesh



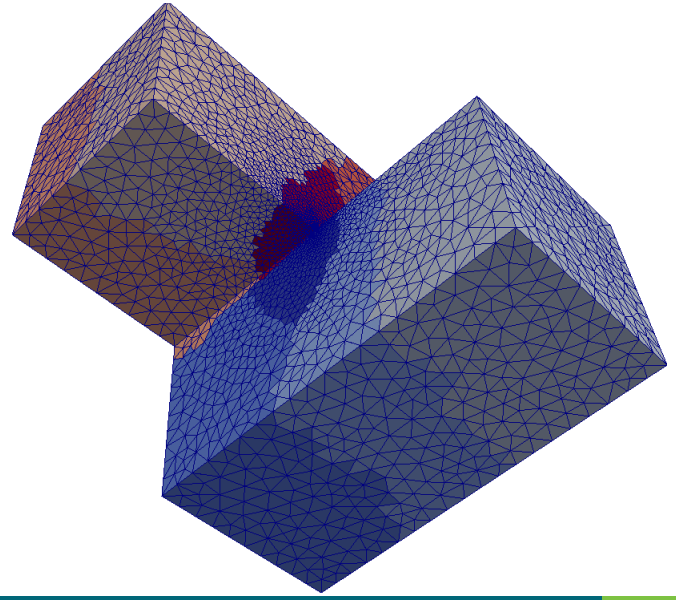
-partcell 2 2 2



-partition 2 2 2



-metis 8



-partdual -metisrec 8

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]

Parallel linear solvers in Elmer

Iterative

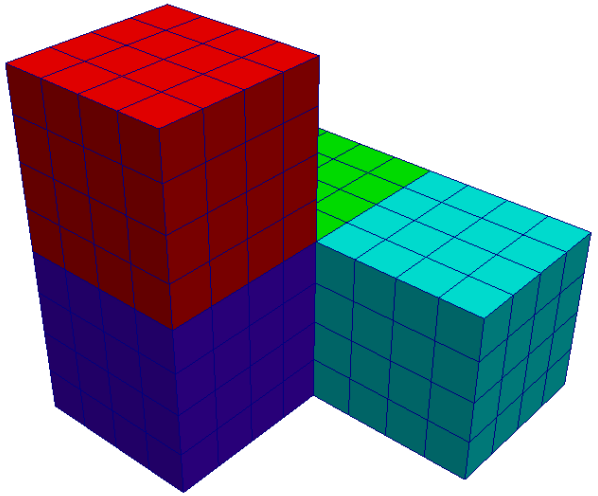
- HUTITER
  - Krylov methods initially coded at HUT
- Hypre
  - Krylov solvers
  - Algebraic multigrid: BoomerAMG
  - Truly parallel ILU and Parasails preconditioning
- Trilinos
  - Krylov solvers
  - Algebraic multigrid: ML
  - …
- ESPRESO
  - FETI library of IT4I
    http://espreso.it4i.cz/

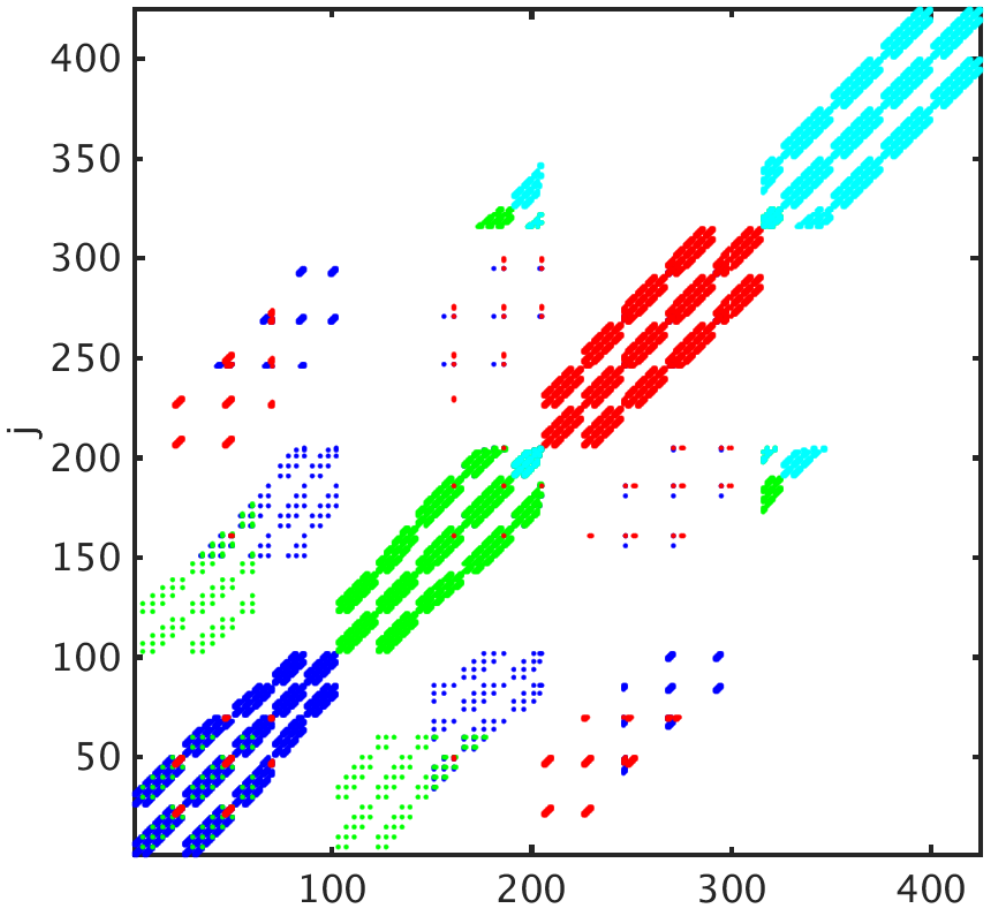
Direct

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

MUMPS

Partitioning and matrix structure

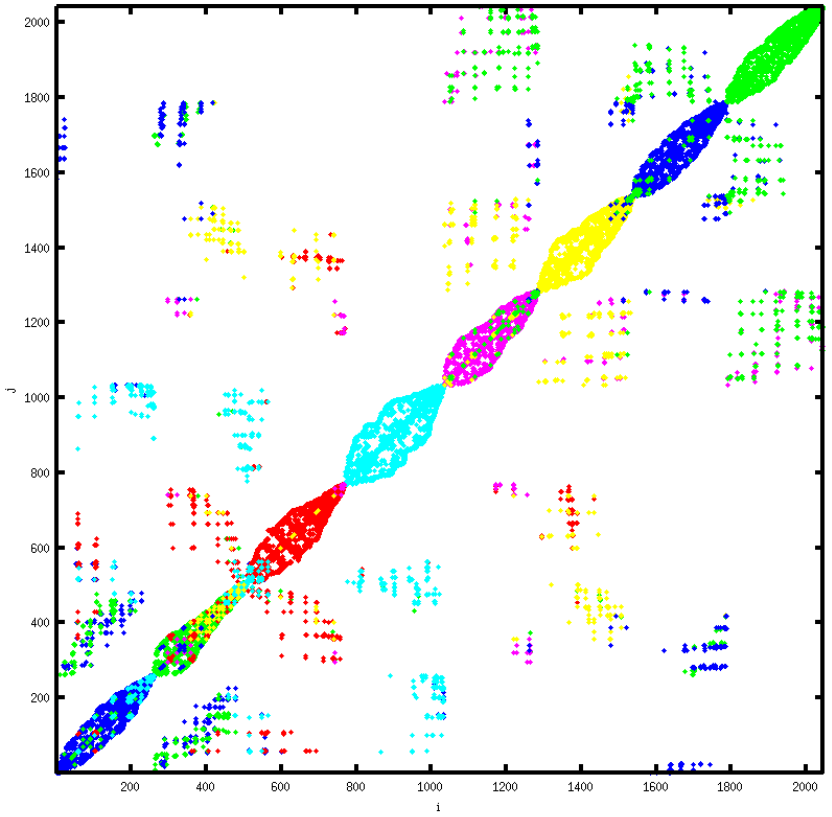


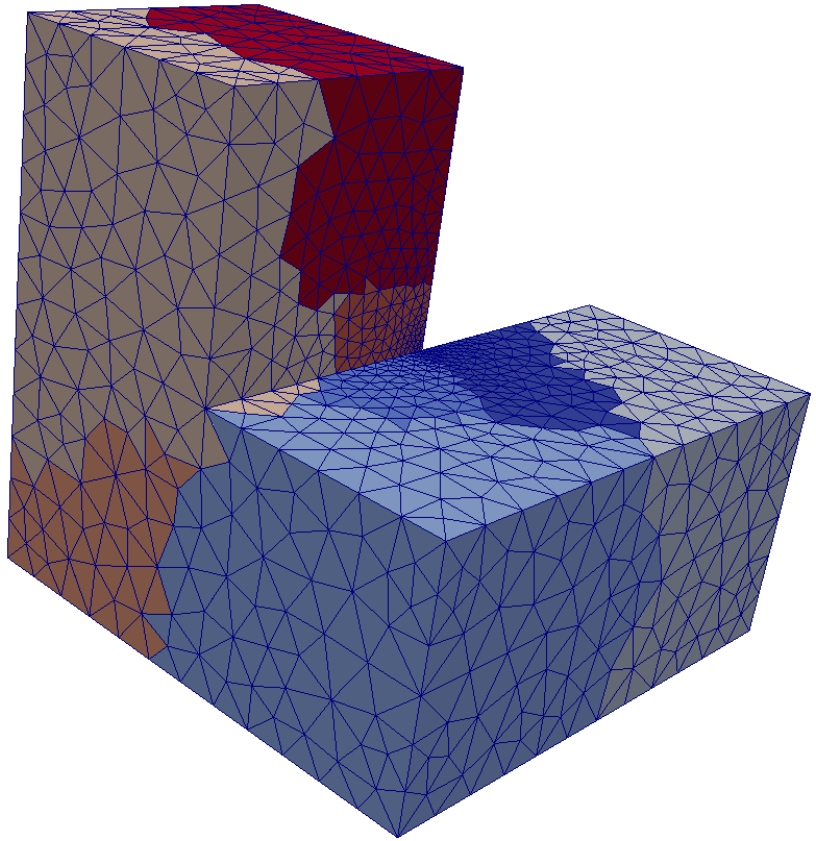


Contiguous parallel numbering used

- 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



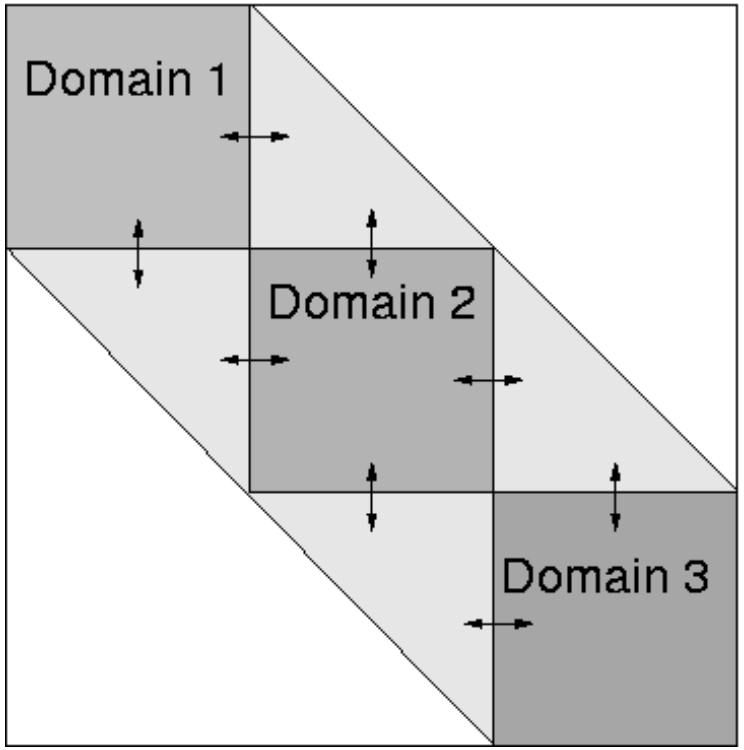


Metis partitioning into 8

- 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

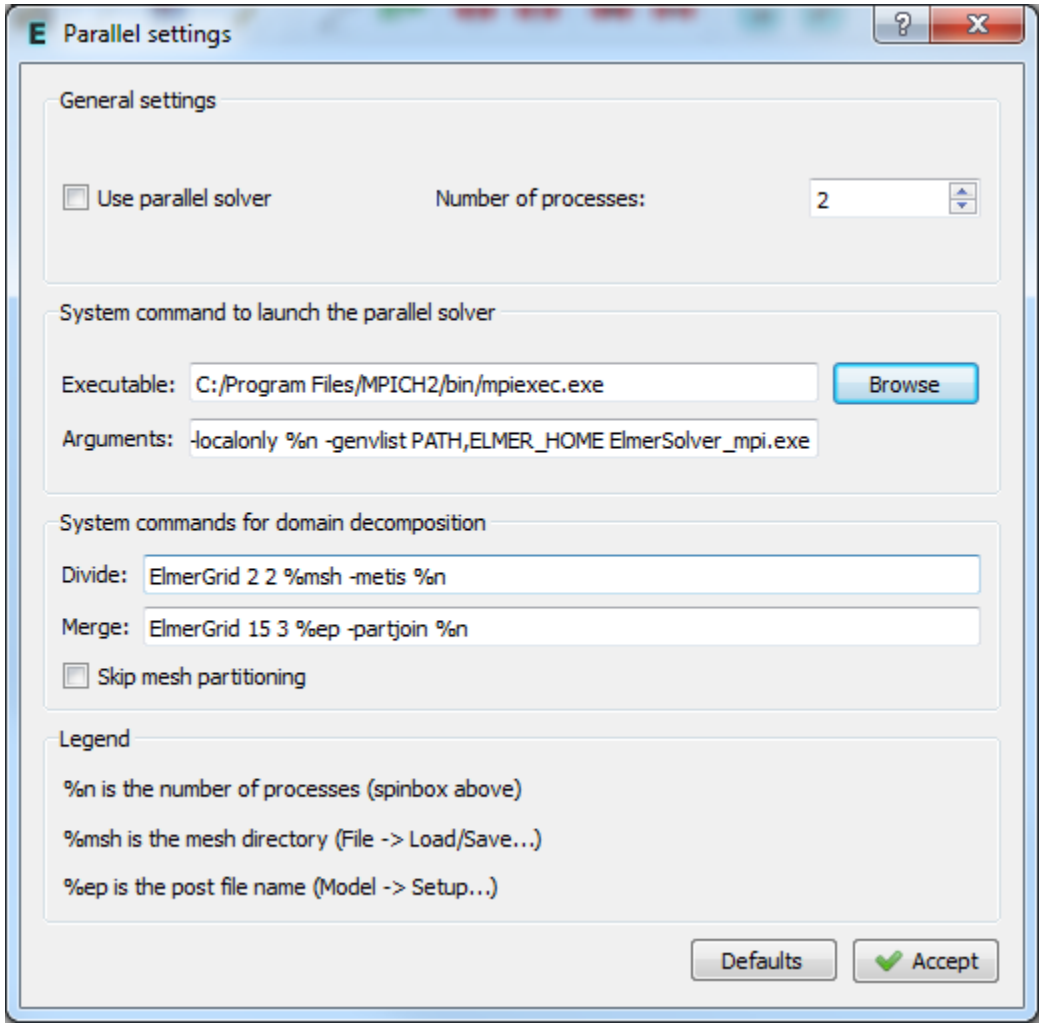
Differences in serial and parallel algorithms

- Some algorithms are slightly different in parallel
- ILU in ElmerSolver library is performed only blockwise which may result to inferior convergence
- Diagonal and vanka preconditions are exactly the same in parallel



Parallel computation in ElmerGUI

- If you have parallel environment it can also be used interactively via **ElmerGUI**
- Calls **ElmerGrid** automatically for partiotioning (and fusing)



Parallel postprocessing using Paraview

- Use ResultOutputSolver to save data to `.vtu` files
- The operation is almost the same for parallel data as for serial data
- There is a extra file `.pvtu` that holds is a wrapper for the parallel `.vtu` data of each partition

Summary: Files in serial vs. parallel solution

Serial

- Serial mesh files
- Command file (.sif) may be given as an inline parameter
- Execution with
  `ElmerSolver [case.sif]`
- Writes results to one file

Parallel

- Partitioned mesh files
- ELMERSOLVER_STARTINFO is always needed to define the command file (.sif)
- Execution with
  `mpirun -np N ElmerSolver_mpi`
- Calling convention is platform dependent
- Writes results to *N* files + 1 wrapper file

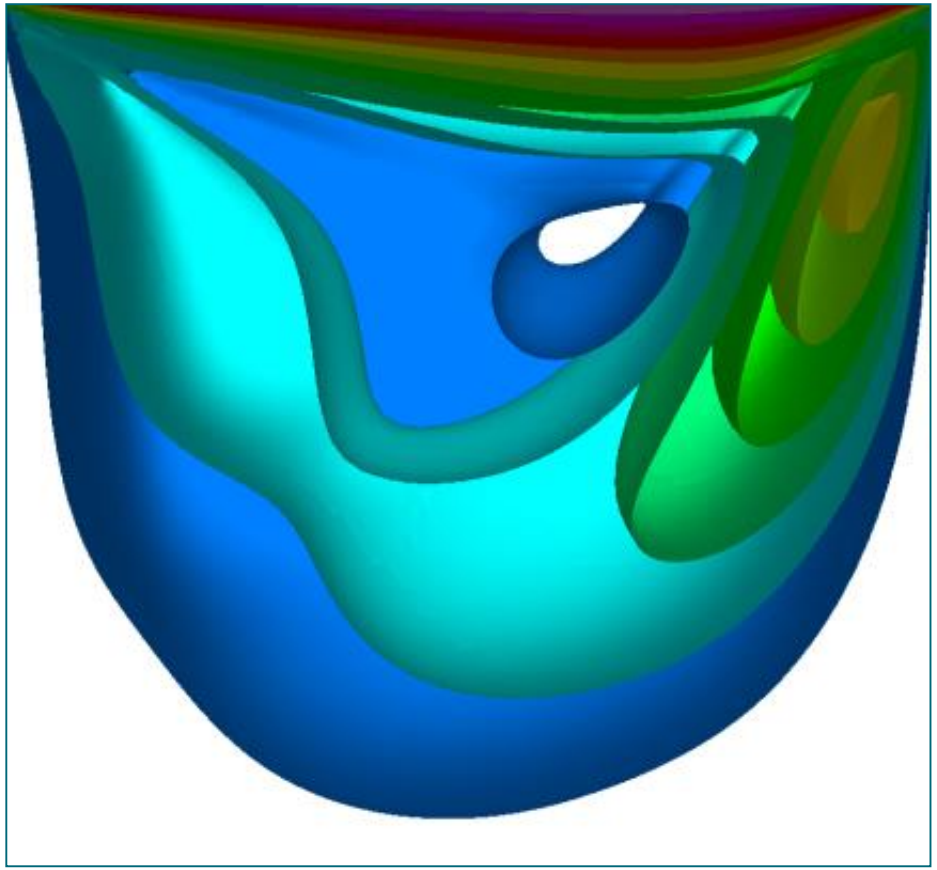
Example: Weak scaling of Elmer (FETI)

#Procs	Dofs	Time (s)	Efficiency
8	0.8	47.80	-
64	6.3M	51.53	0.93
125	12.2M	51.98	0.92
343	33.7M	53.84	0.89
512	50.3M	53.90	0.89
1000	98.3M	54.54	0.88
1331	131M	55.32	0.87
1728	170M	55.87	0.86
2197	216M	56.43	0.85
2744	270M	56.38	0.85
3375	332M	57.24	0.84

Solution of Poisson equation with FETI method where local problem (of size 32^3=32,768 nodes) and coarse problem (distributed to 10 partitions) is solved with MUMPS. Simulation with Cray XC (Sisu) by Juha Ruokolainen, CSC, 2013.

Block preconditioner: Weak scaling of 3D driven-cavity

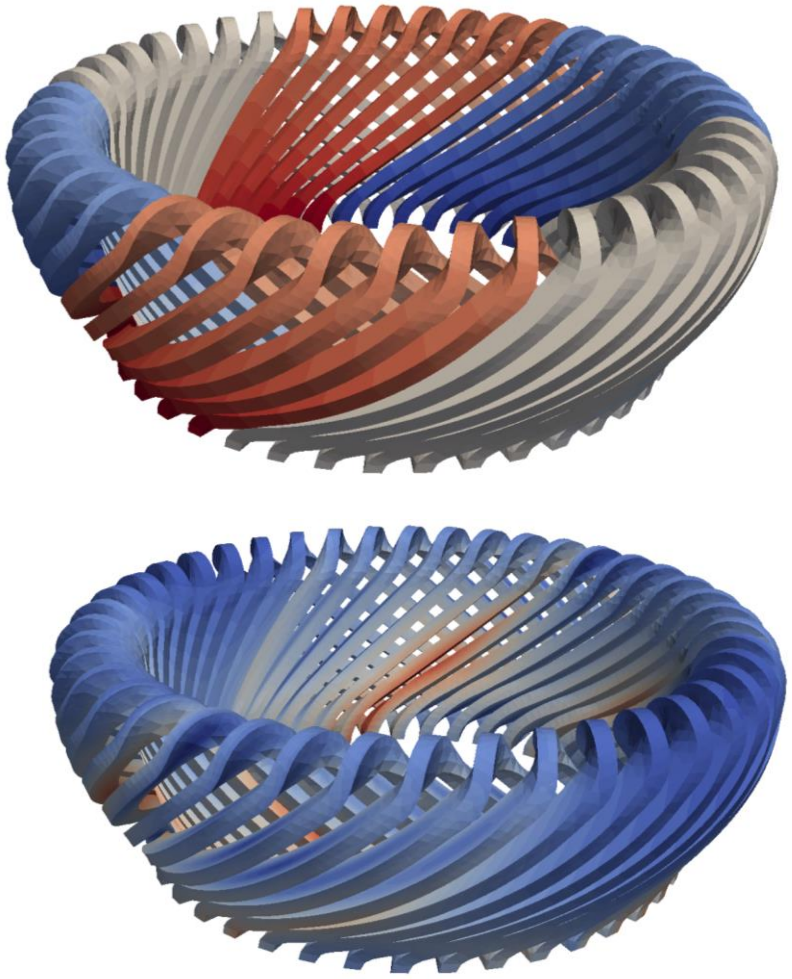
Elems	Dofs	#procs	Time (s)
34^3	171,500	16	44.2
43^3	340,736	32	60.3
54^3	665,500	64	66.7
68^3	1,314,036	128	73.6
86^3	2,634,012	256	83.5
108^3	5,180,116	512	102.0
132^3	9,410,548	1024	106.8



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)

Scalability of edge element AV solver for end-windings



#Procs	Time(s)	T_{2P}/T_P
4	1366	-
8	906	1.5
16	260	3.5
32	122	2.1
64	58.1	2.1
128	38.2	1.8
256	18.1	2.1

Magnetic field strength (left) and electric potential (right) of an electrical engine end-windings. Meshing M. Lyly, ABB. Simulation (Cray XC, Sisu) J. Ruokolainen, CSC.

Coupled model for electrical machines

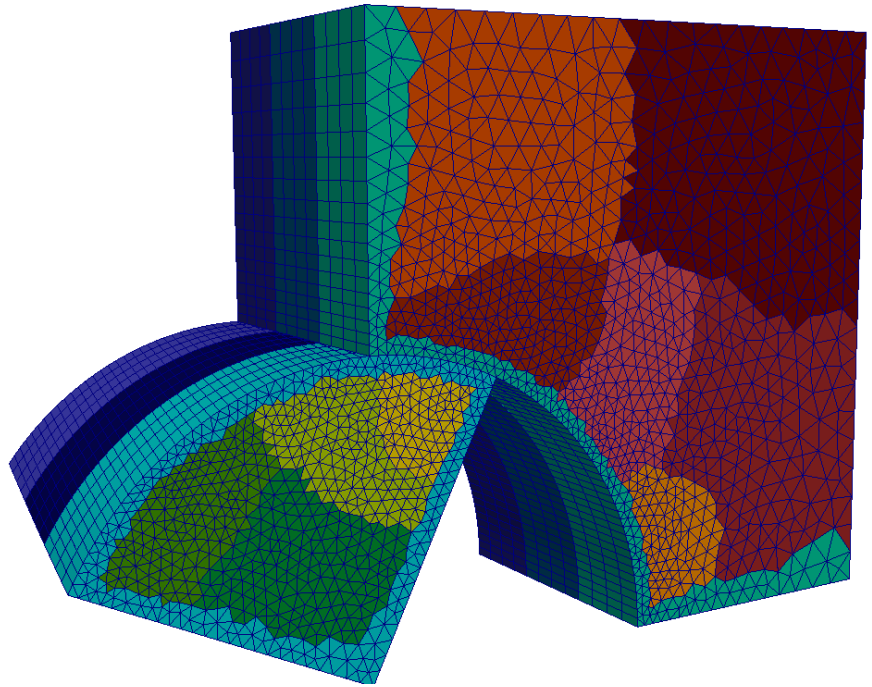
- Monolithic parallel linear system including
  - Electric scalar potential using nodal elements
  - Magnetic vector potential using edge elements (in 3D)
  - Mortar projector for the nodal dofs P_v (for conductors)
  - Mortar projector for the edge dofs P_a (in 3D)
  - Current conditions for case driven by external circuit (few rather dense rows)

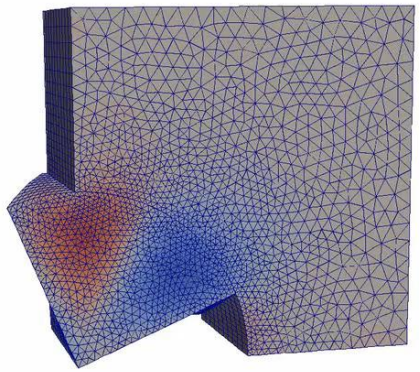
$$\begin{pmatrix} V_v & V_a & P_v^T & 0 & 0 \\ A_v & A_a & 0 & P_a^T & N \\ P_v & 0 & 0 & 0 & 0 \\ 0 & P_a & 0 & 0 & 0 \\ 0 & S & 0 & 0 & R \end{pmatrix} \begin{pmatrix} v \\ a \\ \lambda_v \\ \lambda_a \\ i \end{pmatrix} = \begin{pmatrix} f_a \\ f_v \\ 0 \\ 0 \\ V_{ext} \end{pmatrix}$$

- Solved with Krylov method, e.g. GCR or BiCGStab(l)
- Hybrid preconditioning strategy
  - Vector potential with diagonal
  - Scalar potential & mortar projectors with ILU
  - Electrical circuits either with ILU or MUMPS
- Still some challenges on robustness!

Hybrid partitioning scheme

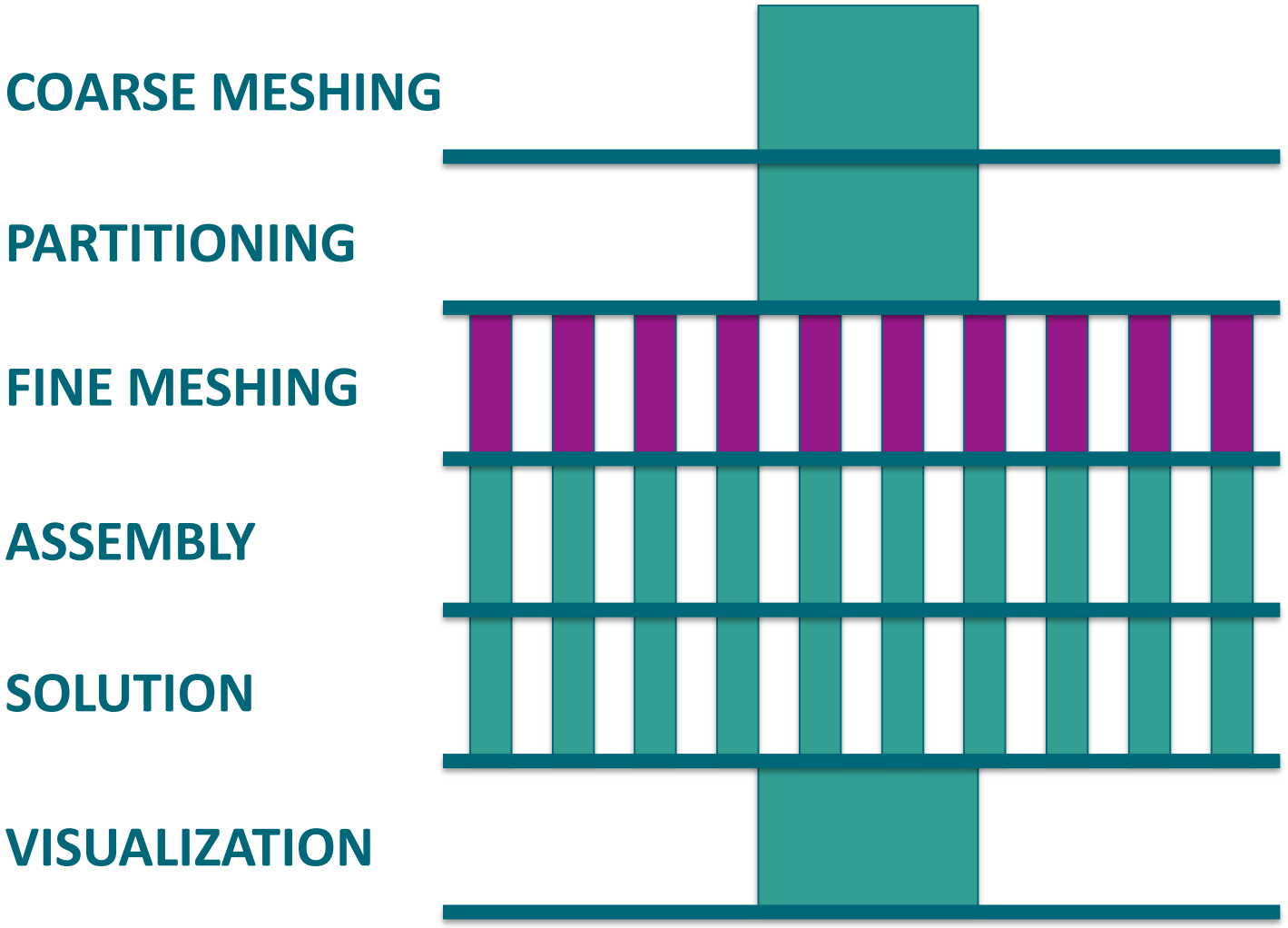
- The linear system arising from the electromagentic problem must be solved together with the continuity constraints
- To minimize communication (and coding) effort we partition the mesh cleverly
- Electrical machines have always rotating interface: Partition the interface elements so that opposing element layers on the cylinder are always within the same partition
  - Unstructured surface meshes are treated similarly except **halo** elements are also saved on the boundary
- Other elements are partitioned with Metis
- Local mortar conditions much easier to deal with!





Parallel workflow for meshing bottle-necks

- Large meshes may be finilized at the parallel level

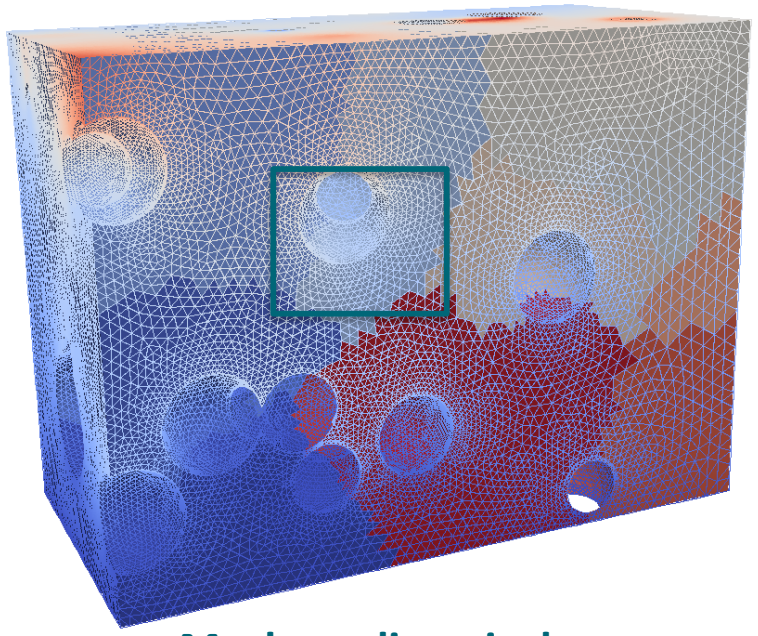


Mesh Multiplication

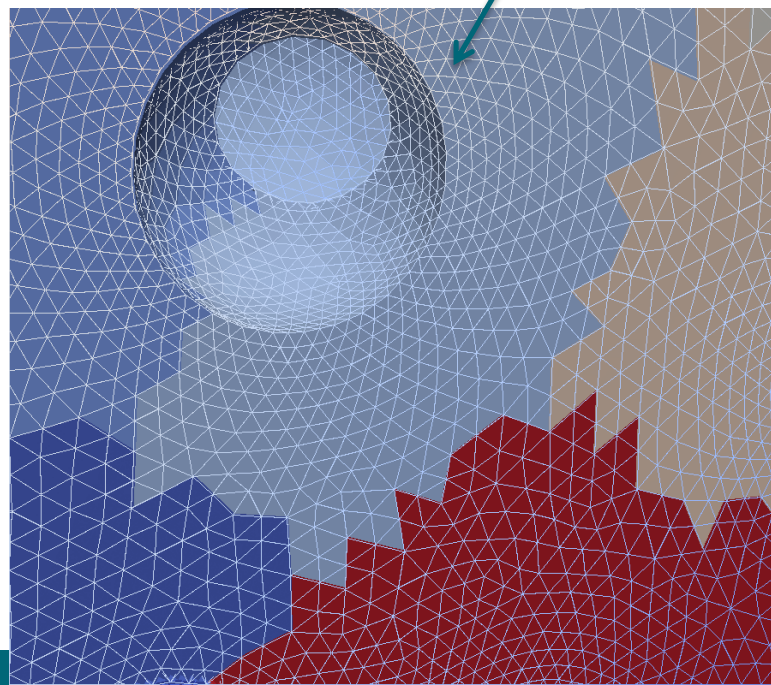
- Split elements edges after partitioning **at parallel level**
  - effectively eliminating memory and I/O bottle-necks
  - Each multiplication creates 2^DIM-fold number of elements
  - Does not increase accuracy of geometry presentation
  - May inherit mesh grading
  - CPU time used in neglible

Mesh	#splits	#elems	#procs	T_center (s)	T_graded (s)
A	2	4 M	12	0.469	0.769
	2	4 M	128	0.039	0.069
	3	32 M	128	0.310	0.549
B	2	4.20 M	12	0.369	
	2	4.20 M	128	0.019	
	3	33.63 M	128	0.201	

Mesh A: structured, 62500 hexahedrons
Mesh B: unstructured, 65689 tetrahedrons

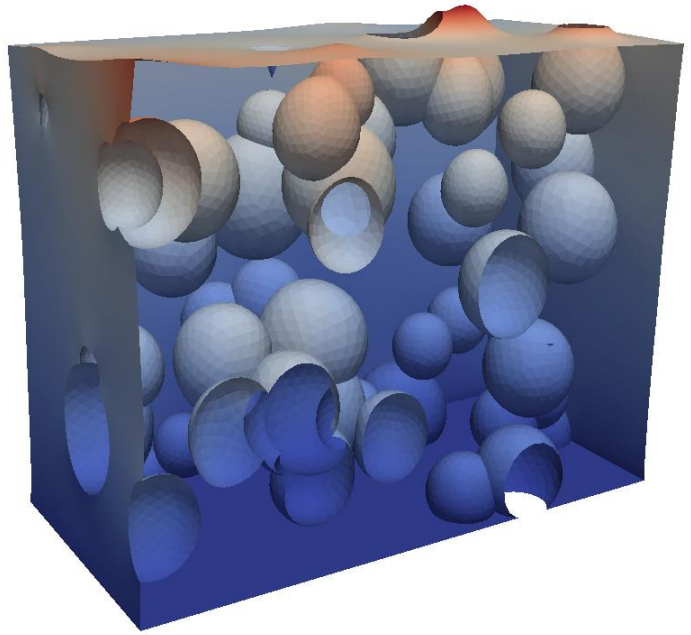


Mesh grading nicely preserved



Overcoming bottle-necks in postprocessing

- Visualization
  - Paraview and Visit excellent tools for parallel visualization
  - Access to all data is often an overkill
- Reducing data
  - Saving only boundaries
  - Uniform point clouds
  - A priori defined isosurfaces
  - Using coarser meshes for output when hierarchy of meshes exist
- Extracting data
  - Dimensional reduction (3D -> 2D)
  - Averaging over time
  - Integrals over BCs & bodies
- More robust I/O
  - Not all cores should write to disk in massively parallel simulations
  - HDF5+XDML output available for Elmer, mixed experiences



Binary output	Single Prec.	Only bound.	Bytes/ node
-	X	-	376.0
X	-	-	236.5
X	X	-	184.5
X	-	X	67.2
X	X	X	38.5

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.

Recipes for resolving scalability bottle-necks

- Finalize mesh on a parallel level (no I/O)
  - Mesh multiplication or parallel mesh generation
- Use algorithms that scale well
  - E.g. Multigrid methods
- If the initial problem is difficult to solve effectively divide it into simpler sub-problems
  - One component at a time -> block preconditioners
    - GCR + Block Gauss-Seidel + AMG + SGS
  - One domain at a time -> FETI
  - Splitting schemes (e.g. Pressure correction in CFD)
- Analyze results on-the-fly and reduce the amount of data for visualization

Future outlook

- Deeper integration of the workflow
  - Heavy pre- and postprocessing internally or via API
- Cheaper flops from new multicore environments
  - Interesting now also for the finite element solvers
  - Usable via reasonable programming effort;
    attention to algorithms and implementation
- Complex physics introduces always new bottle-necks
  - Rotating boundary conditions in parallel…