Elmer

Parallel Scalability & Bottle-necks

ElmerTeam

CSC – IT Center for Science Ltd.

PARA2012 Tutorial
Finlandia Hall, June 2012

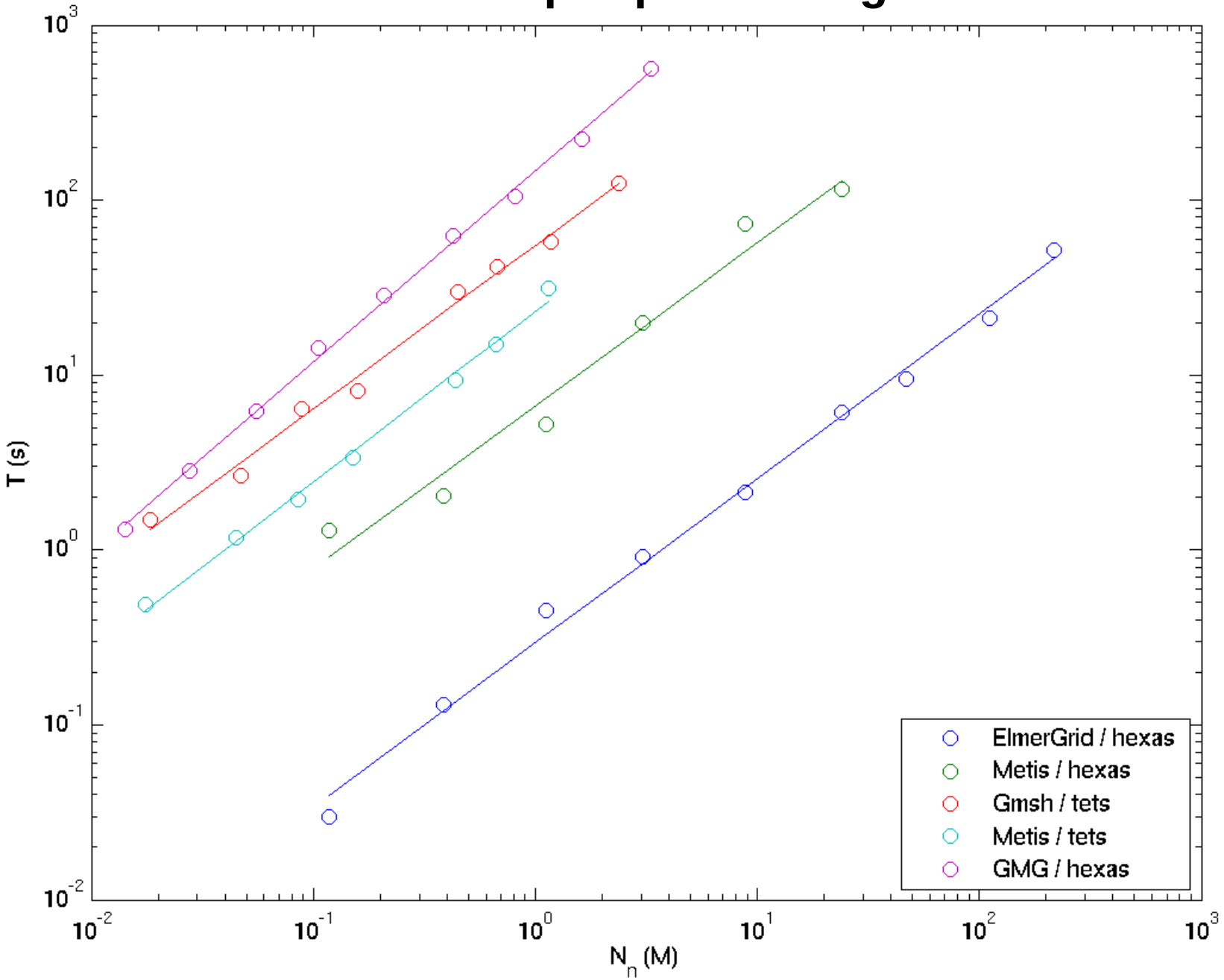
Requirements of parallel scalability

- All steps in the workflow must be considered
- Preprocessing
  - Lack of parallel tools
    -> bottle-necks in {memory, time, I/O}
  - Some possible remedies
- Computation
  - Algorithmic scalability with the problem size
    (weak scaling)
  - Effective parallel implementation (strong scaling)
- Postprocessing
  - Excellent parallel tools
  - Some possible ways to reduce the data

Analysis of serial workflow

- A simple Poisson problem was solved with one core
- Time consumed for each step was analyzed as a function of problem size *N*
- Observations were fitted to the model: $T=aN^b$
- solution time (GMG)
  - > unstructured meshing time (gmsh)
    - > partitioning time (Metis)
      - > structured meshing time (ElmerGrid)
- Scalability of each step almost linear (b ≈ 1-1.1)

CPU time for serial pre-processing and solution



Scalability model

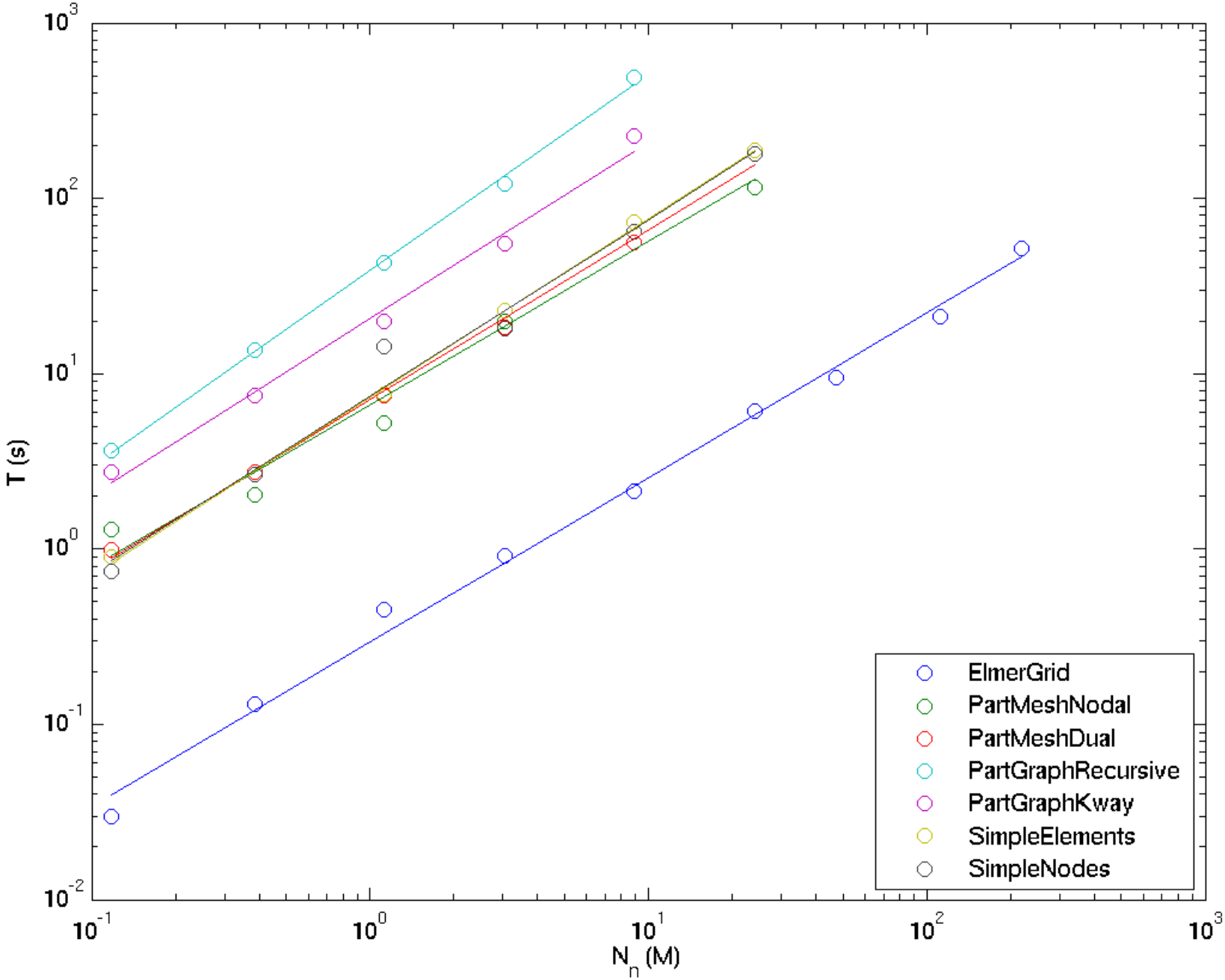
Table 9.1: Serial performance of different tools and algorithms in terms of CPU time and memory consumption

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

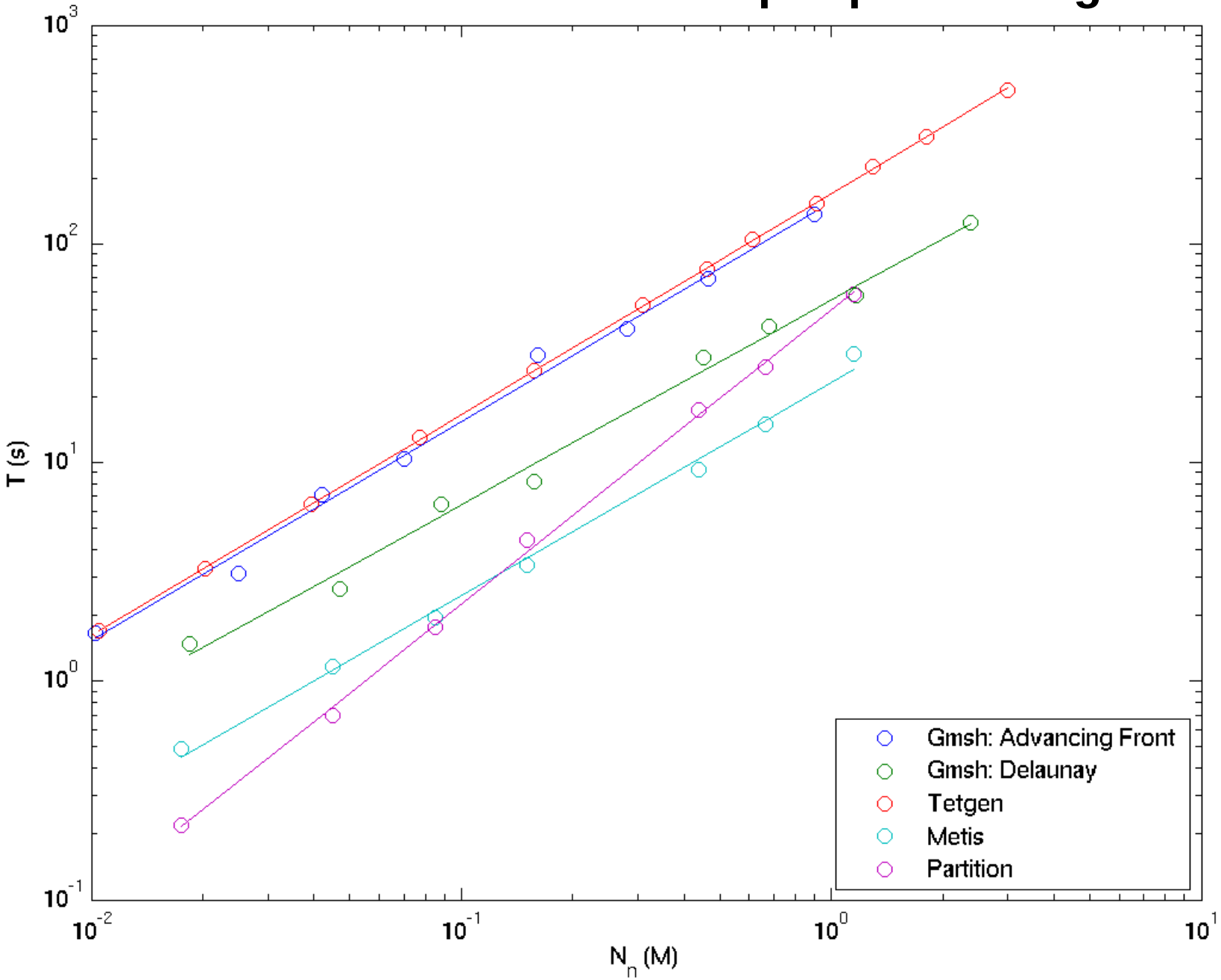
Overcoming bottle-necks in preprocessing

- Meshing is often the most difficult bottle-neck
  - Serial tools used to create up to ~1-10 M nodes
- Options for larger problems
  - Parallel mesh generation
  - Finalizing the mesh in parallel level
- Mesh partitioning is almost always less laborious than meshing
  - Serial partititioning is seldom a problem
  - There are parallel versions of partitioning tools: ParMetis

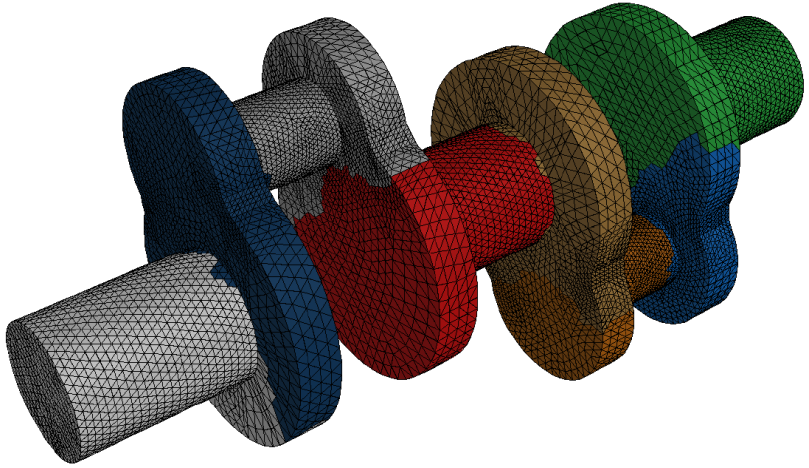
CPU time for structured pre-processing



CPU time for unstructured pre-processing

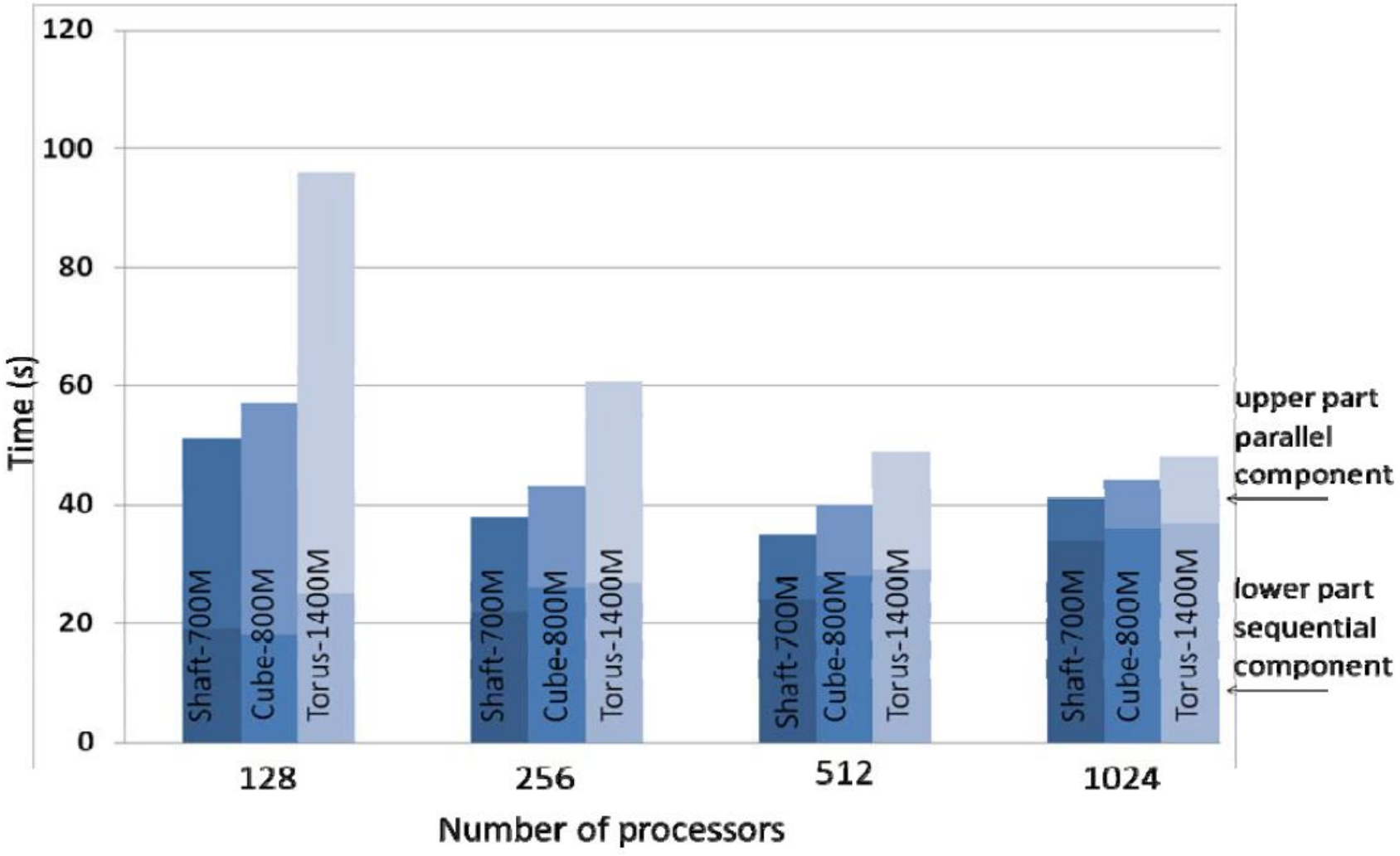


Parallel mesh generation



- Parallel mesh generation is still in its infancy
- No freely available established tools (?)
- Preliminary work for Elmer performed within PRACE in Bogazigi Univ., Istanbul
  - Y. Yılmaz, C. Özturan*, O. Tosun, A. H. Özer, S. Soner *“Parallel Mesh Generation, Migration and Partitioning for the Elmer Application”*
  - Based on netgen serial mesh generation
  - Generate coarse mesh -> partition -> mesh refinement
  - ”mesh with size 1.4 billion could be generated in under a minute”
  - Still experimental, writes mesh into disk for Elmer to read
    -> Introduces a possible I/O bottle-neck
- Ultimately parallel mesh generation should be integrated with an API rather than disk I/O

Parallel mesh generation: performance



Y. Yılmaz et. al.
"Parallel Mesh Generation, Migration and Partitioning for the Elmer Application"

Finalizing the mesh in parallel level

- First make a coarse mesh and partition it
- Division of existing elements (*2^DIM^n* -fold problem-size)
  - Known as ”Mesh Multiplication”
  - In Simulation block set ”`Mesh Levels = N`”
  - There is a geometric multigrid that utilizes the mesh hierarchy
  - Simple inheritance of mesh grading
- Increase of element order (p-elements)
  - There is also a p-multigrid in Elmer
- Extrusion of 2D layer into 3D for special cases
  - Example: Greenland Ice-sheet
- For complex geometries this is often not an option
  - Optimal mesh grading difficult to maintain
  - Geometric accuracy cannot be increased

Mesh Multiplication, example

- Mesh multiplication was applied to two meshes
  - Mesh A: structured, 62500 hexahedrons
  - Mesh B: unstructured, 65689 tetrahedrons
- The CPU time used is negligible

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	

Analysis of algorithmic scalability

- Scalability of algorithms may be studied also in serial simulation
  - Observations of relevant as long as the parallel algorithm has at best the same properties
- Experiments show that only multigrid methods provide almost linear scaling
  - Power 1..1.2
  - Krylov methods ~1.5 in 2D and ~1.4 in 3D

Platform: Lenovo T60 running Elmer under mingw
Equation: Poisson **TempDist.sif**
Geometry: L-shaped domain **angle.grd**
Elementtype: 404

No elements		**10092**	**20008**	**40020**	**70277**	**100101**	**150080**	**200466**			**200466**	
No nodes		**10325**	**20236**	**40483**	**70840**	**100833**	**150976**	**201501**			**201501**	
File		**p1.dat**	**p2.dat**	**p3.dat**	**p4-dat**	**p5.dat**	**p6.dat**	**p7.dat**			**Mem(Mb)**	
Method	**Prec/Sm**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**		**cumt (s)**	**q**	**O(n)**
AMG+SGS		0.14	0.30	0.64	1.08	1.52	2.28	3.09		9.05	102	1.03
CG	AMG+SGS	0.14	0.31	0.66	1.09	1.52	2.30	3.13		9.14	107	1.02
QMR	AMG+SGS	0.20	0.47	1.03	1.66	2.33	3.52	4.66		13.86	120	1.04
BiCGstab	AMG+SGS	0.20	0.45	1.09	1.66	2.34	3.52	4.64		13.91	120	1.04
CG	CMG(4)+SGS	0.16	0.34	0.77	1.47	2.28	3.97	5.53		14.52	102	1.20
QMR	CMG(4)+SGS	0.16	0.38	0.78	1.58	2.42	4.28	5.81		15.41	108	1.21
BiCGstab	CMG(4)+SGS	0.17	0.36	0.83	1.58	2.44	4.27	5.78		15.42	107	1.19
CG	CMG(0)+SGS	0.16	0.39	0.89	1.81	2.83	4.47	6.61		17.16	97	1.25
BiCGstab	CMG(0)+SGS	0.20	0.47	1.16	2.17	3.42	5.49	8.08		20.99	102	1.23
QMR	CMG(0)+SGS	0.20	0.45	1.13	2.19	3.45	5.47	8.13		21.03	102	1.24
CMG(4)+SGS		0.20	0.52	1.20	2.34	3.91	6.75	10.24		25.16	95	1.30
Umfpack		0.20	0.47	1.13	2.63	3.73	6.33	14.80		29.28	374	1.38
CG	ILU1	0.17	0.49	1.39	3.25	5.42	9.92	15.38		36.01	121	1.51
CG	ILU2	0.17	0.52	1.45	3.34	5.53	9.94	15.42		36.38	135	1.50
BiCGStab	ILU1	0.20	0.52	1.64	3.66	5.95	9.75	17.88		39.60		1.49
QMR	ILU1	0.20	0.53	1.67	3.63	6.03	9.75	17.89		39.70		1.48
BiCGStab	ILU2	0.22	0.59	1.67	3.84	6.45	10.24	16.88		39.89		1.45
QMR	ILU2	0.22	0.59	1.67	3.89	6.47	10.20	16.86		39.91		1.45
CGS	ILU2	0.20	0.56	1.67	3.63	6.20	11.17	18.06	43%	41.50		1.50
CGS	ILU1	0.19	0.55	1.59	3.73	6.34	12.14	18.22		42.76		1.54
BiCGStab2	ILU1	0.19	0.55	1.72	3.75	6.58	11.36	19.23		43.38		1.54
CG	ILU0	0.17	0.55	1.67	3.89	6.67	12.31	19.02		44.28	103	1.57
BiCGStabL	ILU1	0.19	0.55	1.75	4.03	6.72	12.06	19.47		44.77		1.56
BiCGStab2	ILU2	0.22	0.61	1.80	3.97	6.55	12.58	19.14		44.86		1.50
BiCGStabL	ILU2	0.19	0.58	1.72	4.17	6.66	12.20	21.31		46.83		1.57
QMR	ILU0	0.22	0.61	1.86	4.67	7.36	13.55	21.77		50.03		1.55
CMG(0)+SGS		0.33	0.91	2.31	5.03	8.08	13.10	20.70		50.46		1.38
BiCGStab	ILU0	0.23	0.63	1.89	4.88	7.36	13.61	21.92		50.51		1.53
BiCGStabL	ILU0	0.25	0.63	2.05	4.67	7.92	15.41	22.19		53.11		1.53
CGS	ILU0	0.25	0.64	1.91	4.56	8.19	15.22	23.30		54.06		1.54
BiCGStab2	ILU0	0.22	0.64	2.17	4.69	8.19	15.28	23.19		54.37		1.57

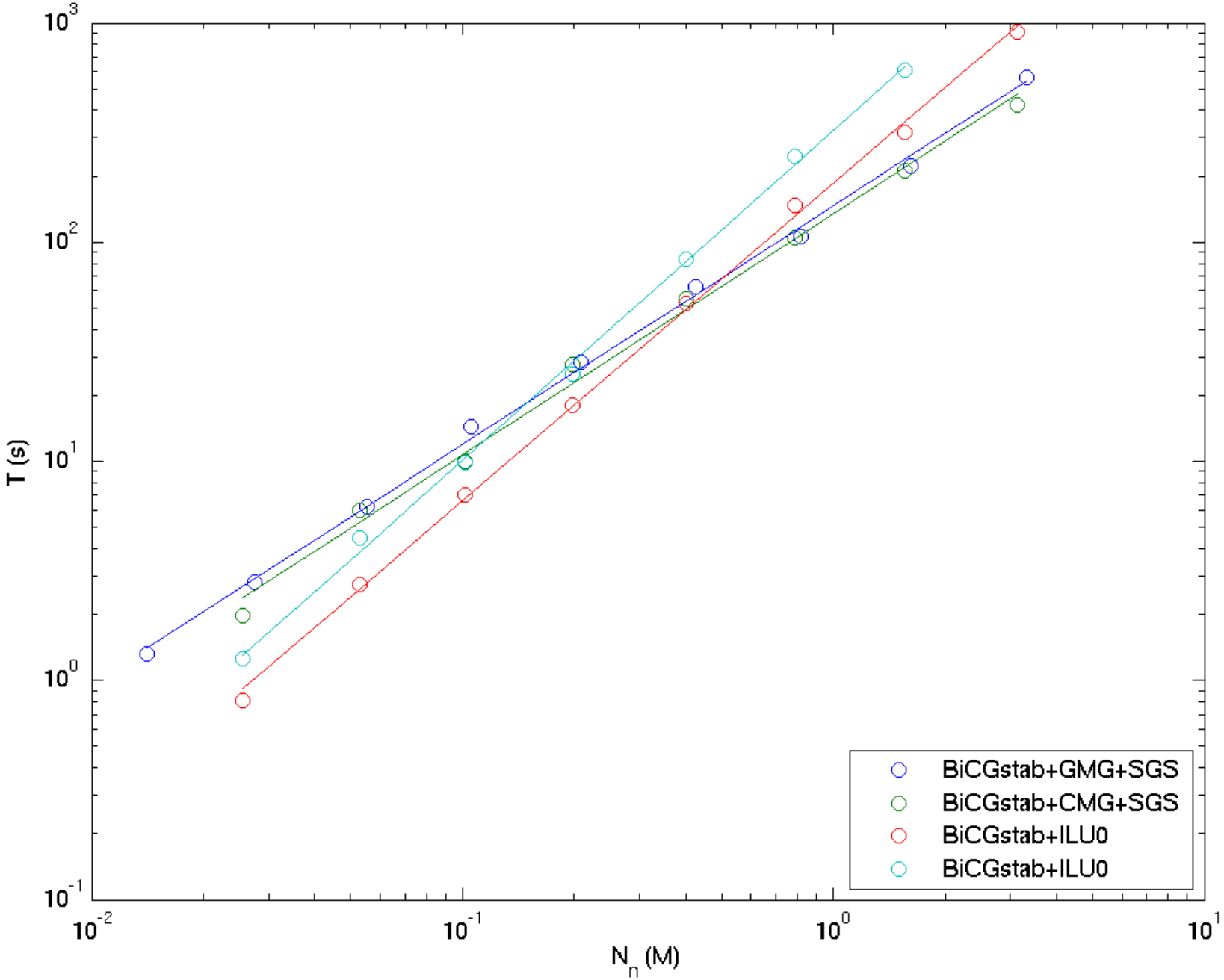
Benchmark of Linear Solvers in Elmer

Platform: Lenovo T60 running Elmer under mingw
Equation: Steady state heat equation **TempDist.sif** **cp=1** **k=1** **rho=1**
Geometry: 3d angle with even spacing **angle3d.grd**
Elementtype: 808

relh		**1.00**	**0.79**	**0.63**	**0.52**	**0.46**	**0.36**	**0.29**	**0.22**	**min(cumt)**	
No nodes		**3410**	**6929**	**13328**	**24380**	**33902**	**69020**	**121380**	**294978**	**14.95**	
File		**v1**	**v2**	**v3**	**v4**	**v5**	**v7**	**v8**	**v9**		
Method	**Prec/Sm**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**t(s)**	**cumt (s)**	**O(n)**
CG	CMG(0)+SGS	0.03	0.11	0.25	0.50	0.73	1.77	1.77	9.80	14.95	1.20
BiCGstab	CMG(0)+SGS	0.05	0.13	0.31	0.56	0.81	1.92	1.92	12.08	17.78	1.16
QMR	CMG(0)+SGS	0.05	0.11	0.33	0.56	0.80	1.94	1.94	12.09	17.81	1.17
CG	CMG(4)+SGS	0.05	0.13	0.27	0.56	0.88	2.13	2.13	12.30	18.42	1.18
BiCGStabL	diag	0.03	0.08	0.19	0.45	0.72	1.97	1.97	13.66	19.06	1.31
CG	diag	0.02	0.08	0.19	0.52	0.72	1.89	1.89	13.97	19.27	1.40
QMR	CMG(4)+SGS	0.05	0.13	0.30	0.64	0.91	2.24	2.24	13.19	19.67	1.19
BiCGstab	CMG(4)+SGS	0.05	0.14	0.30	0.64	1.08	2.24	2.24	13.88	20.55	1.19
BiCGStab2	diag	0.03	0.08	0.19	0.47	0.77	1.92	1.92	15.28	20.66	1.32
BiCGStab	diag	0.02	0.11	0.25	0.61	0.91	2.42	2.42	16.17	22.91	1.41
QMR	diag	0.02	0.11	0.27	0.64	0.91	2.41	2.41	16.30	23.04	1.40
CG	ILU0	0.05	0.09	0.24	0.53	0.88	2.41	2.41	20.22	26.81	1.31
QMR	ILU0	0.03	0.09	0.24	0.56	0.98	2.69	2.69	21.66	28.94	1.39
BiCGStab	ILU0	0.03	0.09	0.24	0.56	0.98	2.70	2.70	21.69	29.00	1.39
BiCGStabL	ILU0	0.03	0.09	0.23	0.55	0.92	2.58	2.58	22.03	29.02	1.39
BiCGStab2	ILU0	0.03	0.09	0.25	0.56	0.95	2.80	2.80	22.39	29.88	1.40
CMG(0)+SGS		0.05	0.16	0.47	0.80	1.31	3.48	3.48	21.67	31.42	1.29
CMG(4)+SGS		0.05	0.13	0.34	0.75	1.25	3.17	3.17	24.08	32.94	1.33
AMG+SGS		0.09	0.30	0.69	1.38	2.00	4.42	4.42	24.64	37.94	1.16
CG	AMG+SGS	0.11	0.30	0.70	1.39	2.00	4.45	4.45	24.64	38.05	1.14
QMR	AMG+SGS	0.13	0.38	0.81	1.75	2.42	5.47	5.47	32.25	48.67	1.16
BiCGstab	AMG+SGS	0.13	0.38	0.81	1.75	2.42	5.49	5.49	32.34	48.80	1.16
CG	ILU1	0.08	0.19	0.48	1.16	1.86	5.34	5.34	49.09	63.55	1.38
QMR	ILU1	0.08	0.20	0.52	1.17	1.94	5.55	5.55	51.36	66.36	1.38
BiCGStabL	ILU1	0.08	0.22	0.50	1.22	1.92	5.47	5.47	51.53	66.41	1.37
BiCGStab	ILU1	0.08	0.20	0.50	1.17	1.92	5.55	5.55	51.80	66.76	1.38
BiCGStab2	ILU1	0.09	0.22	0.56	1.23	2.05	5.78	5.78	53.36	69.08	1.35
QMR	ILU2	0.22	0.52	1.20	2.66	4.22	11.11	11.11	93.56	124.59	1.29
BiCGStab	ILU2	0.20	0.53	1.19	2.64	4.20	11.14	11.14	93.78	124.83	1.29
BiCGStabL	ILU2	0.23	0.53	1.23	2.81	4.34	11.64	11.64	93.74	126.17	1.28
CG	ILU2	0.20	0.52	1.20	2.64	4.14	10.99	10.99	97.31	127.98	1.30
BiCGStab2	ILU2	0.22	0.55	1.31	2.81	4.34	11.64	11.64	97.56	130.08	1.29
Umfpack		0.25	1.27	5.73	14.25	40.60	**NA**	**NA**	**NA**	1000.00	2.15

43%

CPU time for solution – one level vs. multilevel



Parallellism in Elmer library

- Parallelization with MPI
  - Some initial work on hybrid methods i.e. **Open MP + MPI**
- Assembly
  - Each partition assemblies it's own part, no communication
- Parallel Linear solvers included in Elmer
  - Iterative Krylov methods
    - CG, BiCGstab, BiCGStabl, QCR, GMRes, TFQMR,…
    - Require only matrix-vector product with parallel communication
  - Geometric Multigrid
    - Utilizes mesh hierarchies created by mesh multiplication
  - Domain Decomposition: **FETI**
  - Preconditioners
    - ILUn performed block-wise
    - Diagonal and Vanka exactly the same in parallel
    - Multigrid methods more robust as preconditioners

Parallel external libraries for Elmer

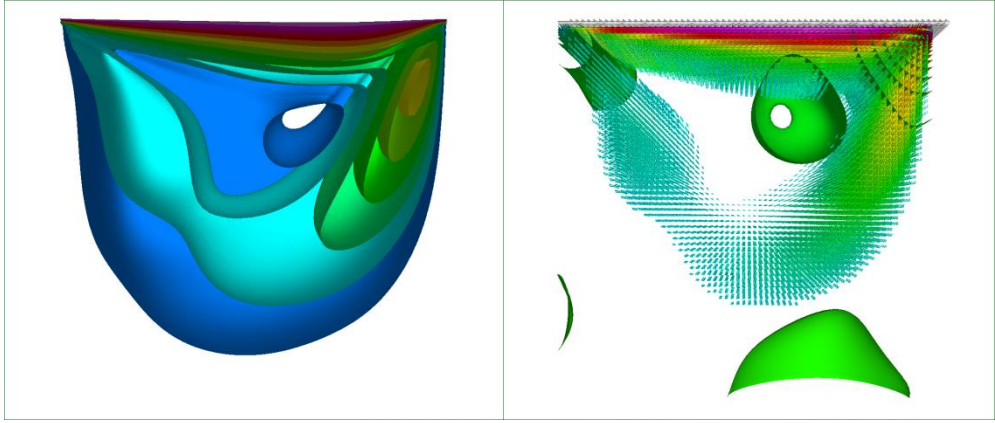
- MUMPS
  - Direct solver that may work when averything else fails
- Hypre
  - Large selection of methods
  - Algebraic multigrid: Boomer MG
  - Parallel ILU preconditioning
  - Approximate inverse preconditioning: Parasails
- Interface to Trilinos
  - Implemented by Jonas Thies, Univ. of Uppsala

Observations in parallel runs

- Typically good scale-up in parallel runs requires around 1e4 dofs in each partition
  - Otherwise communication of shared node data will start to dominate
  - In 3D the number of shared nodes for each partition is ~(N/P)^(2/3) giving rise to a speedup-up of ~1.58 at doubling of partitions even if communication dominates
- To take use of the local memory hierarchies the local problem should not be too big either
  - Sometimes superlinear speed-up is observed when the local linear problem fits to the cache memory
- Good scaling has been shown up to thousands of cores
- Simulation with over one billion unknowns has been performed

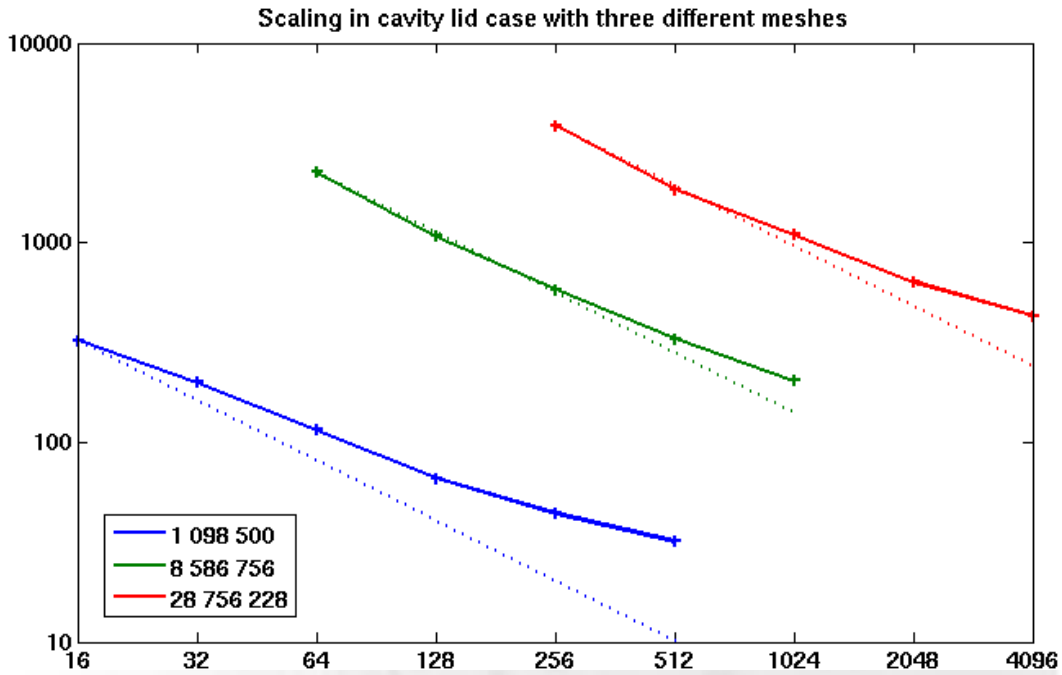
Parallel performance

- Cavity lid case solved with the monolithic N-S solver
- Partitioning with Metis
- Solver Gmres with ILU0 preconditioner



Simulation Juha Ruokolainen
CSC, visualization Matti Gröhn, CSC .

Louhi: Cray XT4/XT5 with 2.3 GHz 4-core AMD Opteron. All-in-all 9424 cores and Peak power of 86.7 Tflops.



Parallel computation, example

- Poisson equation with 100 M dofs
- Mesh created from coarse mesh using mesh multiplication
- Solution with geometric multigrid (GMG) utilizing the different mesh levels
- Very good speedup up to ~1000 partitions

#procs	T(P) / s	T(P)/T(2P)
272	279	-
544	152	1.84
1088	76	2.00
2176	50	1.52

Louhi: Cray XT4/XT5 with 2.3 GHz 4-core AMD Opteron. All-in-all 9424 cores and Peak power of 86.7 Tflops.

Failure of standard methods, example

- Already linear elasticity equation may pose problems for parallel solution
- Case of linear elasticity with 500,000 unknowns
- Many standard methods fail to converge, or scale sub-optimally
  - Need for methods with better robustness & scalability

Method \ T (s)	1	2	4	16	32
Umfpack	53533				
Pardiso	290	175	105	80	65
Mumps		285	190	135	86
BiCG+diag	1270	750	450	225	180
BiCG+ILU(1)	1690	1450	X	580	X
Hypre-BiCG+Parasails		505	295	145	110
Hypre-BiCG+ILU(0)		X	X	506	X

Note: Calculations performed on vuori.csc.fi cluster in 2010.
The solvers may have improved in performance since

FETI Implementation in Elmer

- Finite Element Tearing and Interconnect
- Domain decomposition method for solving linear equations resulting from FEM
- Robust parallel method for elliptic PDEs such as Poisson's or Navier's (linear elasticity) equation
- In each iteration requires solution of a primal problem in each partition, and a dual problem on the global level
- Number of iterations may be shown to be bounded

- Related PARA2012 presentation:
  V. Hapla et al.:
  Massively Parallel Implementation of TFETI DDM on PETSc and Trilinos

FETI Parallel performance

#procs	Time / s	#iters
27	10.52	26
64	12.30	29
125	9.27	31
216	9.96	31
343	10.26	32
512	11.18	32
729	12.13	33
1000	19.88	33
3375	31.52	35

Linear elasticity equation in a unit cube with constant loading from the side.

The size of the case was kept fixed at 8000 elements for each partition.

The largest case includes thus around 80 Mdofs.

Excellent weak scaling up to 729 cores

Computations carried out on Curie (CAE) in 2011

Example, Swiss cheese case with FETI

- Total FETI
- Umfpack as factorizer + ARPACK / Eigenvectors nullspace detection
  - Partititions not continuous
- Lagrange multipliers used as fixing local system

#procs	T_total/s	T(N)/T(2N)	T_factor/s	#CPG_iters
2	222.51		84.70	55
4	84.62	2.63	15.27	60
8	63.54	1.33	4.89	80
16	22.76	2.79	1.30	100
32	26.05(?)	0.87	1.19	100
64	11.03	2.36	0.17	115
128	4.03	2.73	0.09	100

Example, Swiss cheese case with Hypre

- Solution with Hypre
- BiCGStab + Parallel ILU0
- For this small case outperforms FETI
- Strange variations; computing nodes weren't reserved

#procs	T_tot	T(N)/T(2N)
2	16.31	
4	11.52	1.41
8	6.86	1.67
16	5.37	1.28
32	5.16	1.04
64	1.03	5.00

Block preconditioning in Elmer

- 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 equation,...
- Idea: Use as preconditioner a procedure where the components are solved one-by-one (like in Gauss-Seidel) 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 (AMG)
- Related PARA2012 Presentation:
  M. Malinen et al.
  ”*Parallel Block Preconditioning by Using the Solver of Elmer*”

Hybridization in Elmer

- Preliminary work on Open MP + MPI hybridization
- Open MP pragmas have been added for
  - Matrix assembly
  - Sparse matrix-vector multiplication
- The current implementation is efficient for
  - iterative Krylov methods with
  - diagonal or Vanka as preconditioner
- Scaling within one CPU is excellent
- Hybridization will become increasingly important when the number of cores increase

#threds	T(s)
1	341
2	130
4	69
8	47
16	38
32	27

Table: Navier-Stokes equation solved with BiCGStabl(4) and Vanka preconditioning on a HP ProLiant DL580 G7 with quad-core Intel Xeon processors.

Sparse Matrix-Vector product

- In some cases up to 80% of time is used in sparse matrix-vector product
- Sparse matrix presentation (CRS) means indirect memory addressing which kills performance
- Could code efficiency be improved with improved matrix presentation?

Related PARA2012 presentation:
V. Karakasis, G. Goumas et al.
"*Using State-Of-The-Art Sparse Matrix Optimizations for Accelerating the Performance of Multiphysics Simulations*"

Matrix-vector product code in Elmer

```
!------------------------------------------------------------------------------
!>    Matrix vector product (v = Au) for a matrix given in CRS format.
!------------------------------------------------------------------------------
  SUBROUTINE CRS_MatrixVectorMultiply( A,u,v )
!------------------------------------------------------------------------------
    REAL(KIND=dp), DIMENSION(*), INTENT(IN) :: u   !< Vector to be multiplied
    REAL(KIND=dp), DIMENSION(*), INTENT(OUT) :: v  !< Result vector
    TYPE(Matrix_t), INTENT(IN) :: A                !< Structure holding matrix
!------------------------------------------------------------------------------
     INTEGER, POINTER  CONTIG :: Cols(:),Rows(:)
     REAL(KIND=dp), POINTER  CONTIG :: Values(:)

     INTEGER :: i,j,n
     REAL(KIND=dp) :: rsum
!------------------------------------------------------------------------------

     n = A % NumberOfRows
     Rows   => A % Rows
     Cols   => A % Cols
     Values => A % Values

!$omp parallel do private(j,rsum)
     DO i=1,n
        rsum = 0.0d0
        DO j=Rows(i),Rows(i+1)-1
           rsum = rsum + u(Cols(j)) * Values(j)
        END DO
        v(i) = rsum
     END DO
!$omp end parallel do
!------------------------------------------------------------------------------
  END SUBROUTINE CRS_MatrixVectorMultiply
!------------------------------------------------------------------------------
```

Overcoming bottle-necks in postprocessing

- Visualization
  - Paraview and Visit excellent tools for parallel visualization
  - Still the sheer amount of data may be overwhelming and 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

Memory consumption of files, example

- Memory consumption of vtu-files (for Paraview) was studied in the ”swiss cheese” case
- The ResultOutputSolver with different flags was used to write output in parallel
- Saving just boundaries in single precision binary format may save over 90% in files size compared to full data in ascii
- With larger problem sizes the benefits are amplified

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

Relative importance of bottle-necks

- Serial runs
  - Solution is typically the bottle-neck
  - Algorithmic scalability not a major issue
- Small parallel runs (P~=10)
  - Balance between pre- prosessing rather good
  - Algorithmic scalabity already a concern
- Large parallel runs (P~=100)
  - Preprocessing often a bottle-neck but just managable
  - Postprocessing may be heavy and requires consideration
- Massively parallel runs (P~>1000)
  - All phases require special attension
  - Preprocessing either cheap and simple, or complex and parallel
  - Postprocessing often requires parallel strategies
  - Extra care must be be put to the finest details
    - Just taking an FE norm may introduce a bottle-neck

Recipes for optimal scalability in Elmer

- Finalize mesh on a parallel level (no I/O)
  - Mesh multiplication or parallel mesh generation
- Use algorithms that scale well
  - I.e. 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