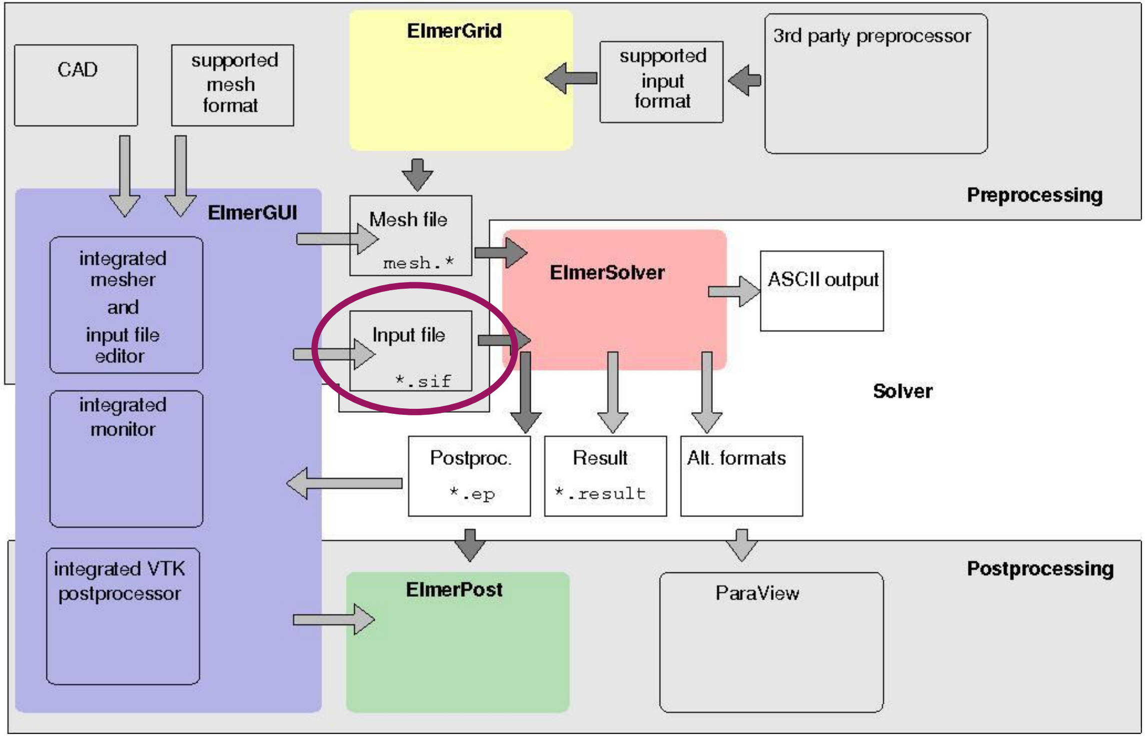
ElmerSolver Input File (SIF) Explained

Elmer Team
CSC – IT Center for Science Ltd.

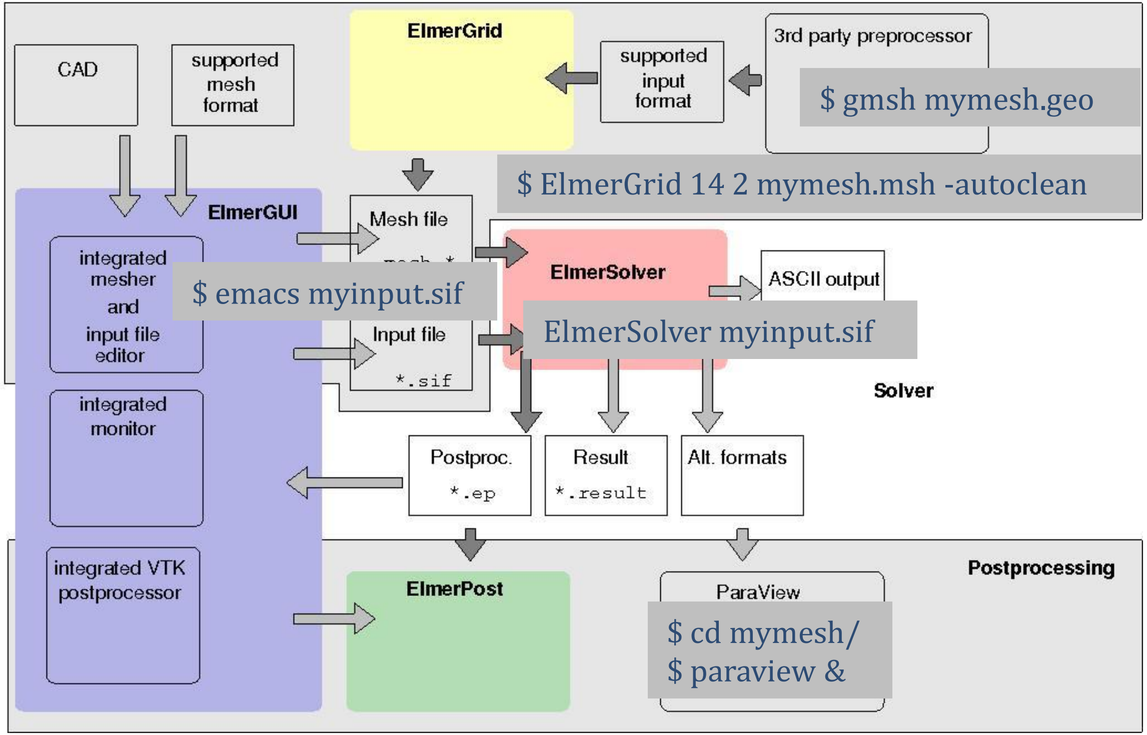
Contents

- Elmer Modules
- Syntax of SIF
  - Parameters, etc.
- Sections of SIF:
  - Header
  - Constants
  - Simulation
  - Solver
  - Body
  - Equation
  - Body Force
  - Material
  - Initial Condition
  - Boundary Condition
- Tables and Arrays
- MATC
- User Defined Functions

Elmer - Modules



Elmer - Modules



```
$ gmsh mymesh.geo
```

```
$ ElmerGrid 14 2 mymesh.msh -autoclean
```

```
$ emacs myinput.sif
```

```
ElmerSolver myinput.sif
```

```
$ cd mymesh/
$ paraview &
```

Sections of SIF

- The SIF is structured into sections
  - `Header`
  - `Constants`
  - `Simulation`
  - `Solver`
  - `Body`
  - `Equation`
  - `Body Force`
  - `Material`
  - `Initial Condition`
  - `Boundary Condition`

The contents of each section is between the keyword above and an **`End`**-statement

Sections of SIF: Header

- Declares search paths for main directories

```
Header
 Mesh DB "." "dirname"



 Include Path "includename"


 Results Directory "resultdir"


End
```

- preceding path + directory name of mesh database

  Mesh directory under ***dirname***

- Shared objects, etc. under ***includename***

- different output directory ***resultdir***

  `By default under mesh-directory`

Sections of SIF: Constants

- Declares simulation-wide constants

```
Constants
  Gas Constant = Real 8.314E00
  Gravity(4) = 0 -1 0 9.81

End
```

- a casted scalar constant
- Gravity vector, an array with a registered name (special setup for certain solvers)

Sections of SIF: Simulation

- Declares details of the simulation:

```
Simulation
  Coordinate System = "Cartesian 2D"



  Coordinate Mapping(3) = Integer 1 2 3


  Coordinate Scaling(3) = Real 1.0 1.0 0.001
  Simulation Type = "Transient"


  Output Intervals(2) = 10 1
```

- choices: **`Cartesian{1D,2D,3D}, Polar{2D,3D}, Cylindric, Cylindric Symmetric, Axi Symmetric`**
- Permute, if you want to interchange directions in mesh
- That would scale the 3rd direction by 1/1000
- **`Steady State, Transient or Scanning`**
- Interval of results being written to disk

Sections of SIF: Simulation

- Declares details of the simulation:

```
Steady State Max Iterations = 10

Steady State Min Iterations = 2

Timestepping Method = "BDF"

BDF Order = 1

Timestep Intervals(2) = 10 100

Timestep Sizes(2) =  0.1 1.0

Output File = "name.result"

Post File = "name.vtu"
```

- How many min/max rounds on one timelevel/in a steady state simulation (see later)
- Choices: **`BDF`**, **`Newmark`** or **`Crank-Nicholson`**
- This would be implicit Euler
- Has to match array dimension of **`Timestep Sizes`**
- The length of one time step
- Contains data for restarting
- Contains output data for ParaView (**`vtu`**)
  - alternatively, suffix **`.ep`** would produce ElmerPost legacy output

Sections of SIF: Simulation

- Declares details of the simulation:

```
  Restart File = “previous.result”

  Restart Position = 10

  Restart Time = 100

  Initialize Dirichlet Condition = False

  Restart Before Initial Conditions = True


  Max Output Level = 5


End
```

- Restart from this file at file-entry (not necessarilly timestep!) no. 10 and set time to 100 time-units
- Default is True. If false, Dirichlet conditions are called at Solver execution and not at beginning
- Default is False. If True, then Initial Condition can overwrite previous results
- Level of verbosity:

  1 = errors
  3 = warnings
  4 = default
  10 = all (sometimes too much) information

Sections of SIF: Solver

- Declares a physical model to be solved

```
Solver 3

 Equation = "Navier-Stokes"

 Exec Solver = "Always"


 Linear System Solver = "Iterative"


 Linear System Iterative Method = "BiCGStab"

 Linear System Convergence Tolerance = 1.0e-6

 Linear System Abort Not Converged = True


 Linear System Preconditioning = "ILU2"
```

- Numbering from 1 (priority)
- The name of the equation
- **`Always`** (default), **`Before/After Simulation/Timestep/Saving`**
- Choices: **`Iterative, Direct, MultiGrid`**
- Lots of choices here, if
- Convergence criterion
- If not True (default) continues simulation in any case
- Pre-conditioning method

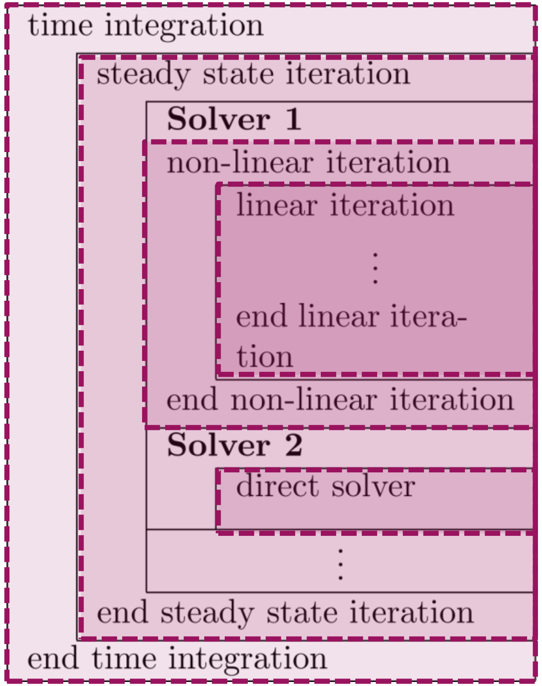
Sections of SIF: Solver

- Declares a physical model to be solved

```
  Nonlinear System Convergence Tolerance= 1.0e-5

  Nonlinear System Max Iterations = 20

  Nonlinear System Min Iterations = 1

  Nonlinear System Newton After Iterations=10


  Nonlinear System Newton AfterTolerance=1.0e-3


  Steady State Convergence Tolerance = 1.0e-3

  Stabilization Method = Stabilized

End
```

- Convergence criterion for non-linear problem
- The maximum rounds
- The minimum rounds
- Switch from fixed-point to Newton scheme after 10 iterations ...
- ... or after this criterion (NV.: has to be smaller than convergence criterion ot hit)
- The convergence on the time-level
- advection needs stabilization. Alternatives: **`Bubbles`**, **`VMS`**, **`P2/P1`**

Sections of SIF: Solver



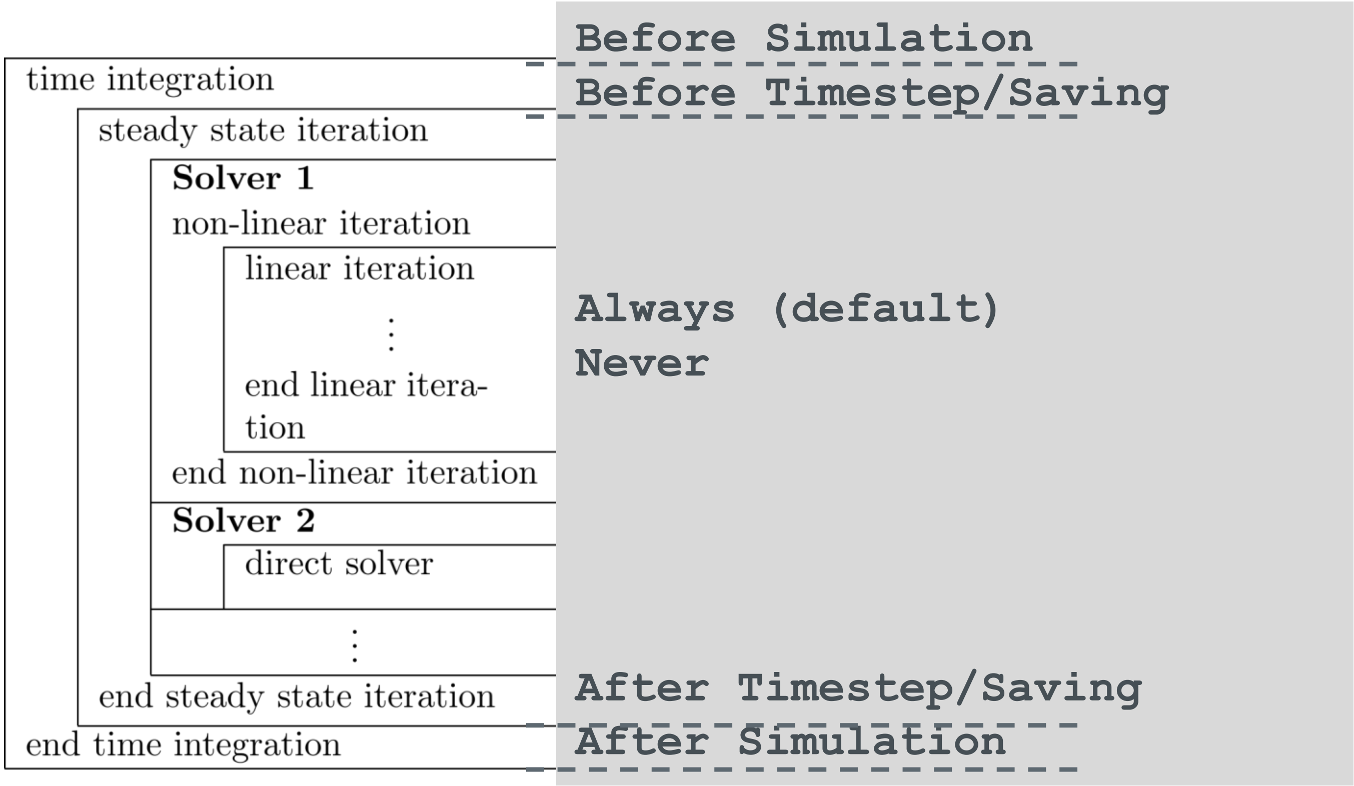
1. `Timestep Intervals`
2. `Steady State Max Iterations`
3. `Nonlinear Max Iterations`
4. `Linear System Max Iterations`
4. `Linear System Convergence Tolerance` ϵ_l
3. `Nonlinear System Convergence Tolerance` ϵ_n

`Nonlinear System Convergence Tolerance`

2. `Steady State Convergence Tolerance` ϵ_s
1.

$\epsilon_l < \epsilon_n < \epsilon_s$

Sections of SIF: Solver



time integration

steady state iteration

Solver 1

non-linear iteration

linear iteration

⋮

end linear iteration

end non-linear iteration

Solver 2

direct solver

⋮

end steady state iteration

end time integration

```
Before Simulation
Before Timestep/Saving
Always (default)
Never
After Timestep/Saving
After Simulation
```

Sections of SIF: Body

- Declares a physical model to be solved

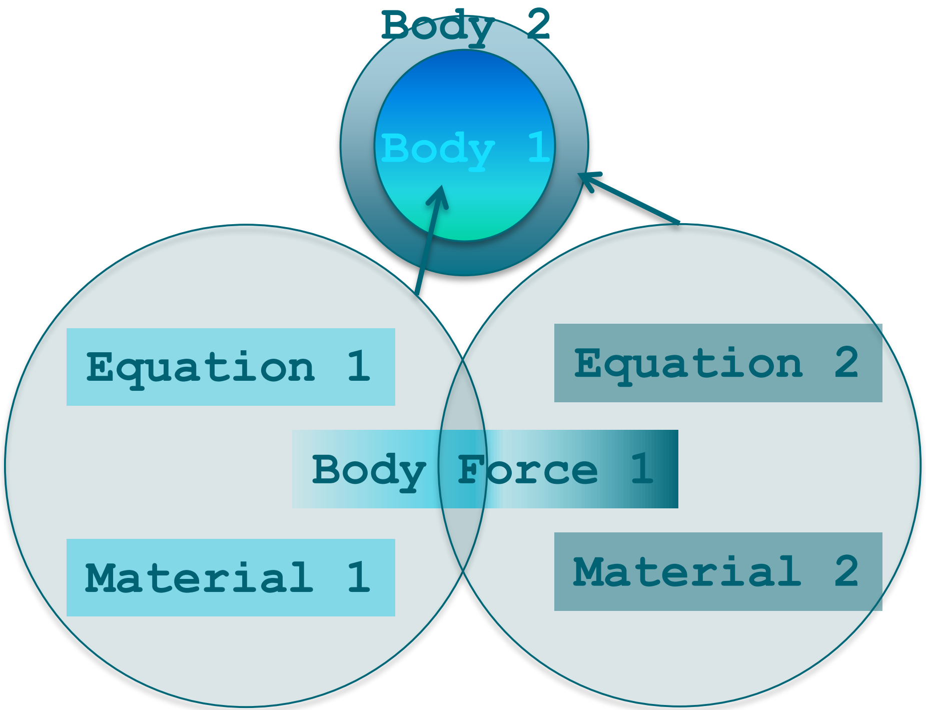
```
Body 2


 Name = "pipe"
 Equation = 2
 Material = 2
 Body Force = 1
 Initial Condition = 2


End
```

- Numbering from 1 to number of bodies

- Identifier of the body
- The assigned set of equations
- The assigned material section
- The assigned body force
- The assigned initial condition

Sections of SIF: Body

- Each **`Body`** has to have an **`Equation`** and **`Material`** assigned
  - **`Body Force`**, **`Initial Condition`** optional
- Two bodies can have the same **`Material/Equation/ Body Force/Initial Condition`** section assigned



Sections of SIF: Equation

- Declares set of solvers for a body

```
Equation 2

 Active Solvers(2) = 1 3

 Convection = Computed

End
```

- Numbering from 1 to number of equation sets
- Declares the solvers (according to their numbers) to be solved within this set
- Important switch to account for convection term. Alternatives: None and Constant (needs Convection Velocity to be declared in the Material section)

Sections of SIF: Body Force

- Declares body forces and bulk and execution conditions for a body

```
Body Force 3
 Flow Body Force 1 = 0.0
 Flow Body Force 2 = -9.81


 MyVariable = Real 0.0


 Heat Source = 1.0
End
```

- Numbering from 1 to number of body forces
- Gravity pointing in negative x-direction applied to Navier-Stokes solver
- A Dirichlet condition for a variable set within the body
- Heat source for the heat equation

Sections of SIF: Material

- Declares set of material parameters for body

```
Material 1
 Density = 1000.0

 Heat Conductivity(3,3) = 1 0 0\
                          0 1 0\
                          0 0 2
 Viscosity = Variable Temperature
    Real MATC "viscosity(tx)"

 Heat Capacity = Variable Temperature
  Procedure "filename" "functionname"


 MyMaterialParameter = Real 0.0

End
```

- Numbering from 1 to number of material
- Always declare a density
- Parameters can be arrays
- Or MATC functions of other variables
- Or Fortran functions with/without dependency on input variables
- Non-keyword DB parameters have to be casted

Sections of SIF: Initial Condition

- Declares initial conditions for a body (by default restart values are used)

```
Initial Condition 2
 Velocity 1 = Variable Coordinate 2
    Real MATC "42.0*(1.0 – tx/100.0)"

 Velocity 2 = 0.0

 Velocity 3 = Variable Coordinate 3
   Procedure "filename" "functionname"


  MyVariable = Real 20.0

End
```

- Numbering from 1 to number of IC's
- Initial condition as a MATC function of a variable ...
- ... and as a constant value
- ... and as a user function
- Non-keyword DB parameters have to be casted

Sections of SIF: Boundary Condition

- Declares conditions at certain boundaries

```
Boundary Condition 3

 Target Boundaries(2) = 1 4

 Velocity 1 = Variable Coordinate 2
    Real MATC "42.0*(1.0 – tx/100.0)"

 Velocity 2 = 0.0

 Velocity 3 = Variable Coordinate 3
   Procedure "filename" "functionname"

 Normal-Tangential Velocity = Logical True

End
```

- Numbering from 1 to number of BC's
- The assigned mesh boundaries
- Variable as a MATC function and …

  … as a constant

  … as a user function
- Set velocities in normal-tangential system

Tables and Arrays

- Tables (piecewise linear or cubic):

```
Density = Variable Temperature
Real cubic
    0  900
  273 1000
  300 1020
  400 1000
End
```

- Arrays:

```
Target Boundaries(3) = 5 7 10
MyParamterArray(3,2) = Real 1 2\
3 4\
5 6
```

- Expresions:

```
OneThird = Real $1.0/3.0
```

MATC

- Syntax close to C
- Even if-conditions and loops
- Can be use for on-the-fly functions inside the SIF
- Documentation on web-pages
- Do not use with simple numeric expressions:

```
OneThird = Real $1.0/3.0
```

is much faster than

```
OneThird = Real MATC "1.0/3.0"
```

MATC

- Use directly in section:

```
Heat Capacity = Variable Temperature
 Real MATC "2.1275E3 + 7.253E0*(tx - 273.16)"
```

- Even with more than one dependency:

```
Temp = Variable Latitude, Coordinate 3
 Real MATC "49.13 + 273.16 - 0.7576*tx(0)- 7.992E-03*tx(1)"
```

- Or declare functions (somewhere in SIF, outside a section)

```
$ function stemp(X) {\
  _stemp = 49.13 + 273.16 - 0.7576*X(0) - 7.992E-03*X(1)\
}
```

being called by:

```
Temp = Variable Latitude, Coordinate 3
    Real MATC "stemp(tx)"
```

User Defined Functions (UDF)

- Written in Fortran 90
- Dynamically linked to Elmer
- Faster, if more complicated computations involved
- Compilation command **`elmerf90`**

```
$ elmerf90 myUDF.f90 –o myUDF.so
```

- Call from within section:

```
MyVariable = Variable Temperature
  Real Procedure "myUDF.so" "myRoutine"
```

User Defined Functions (UDF)

- Example: $\rho(T[K]) = 1000.0 \cdot \left[1 - 1 \times 10^{-4} \cdot (T - 273.15)\right]$

```
FUNCTION getdensity( Model, N, T ) RESULT(dens)
 USE DefUtils !important definitions
 IMPLICIT None
 TYPE(Model_t) :: Model
 INTEGER :: N
 REAL(KIND=dp) :: T, dens
 dens = 1000.0_dp*(1.0_dp - 1.0d-04*(T - 273.0_dp))
END FUNCTION getdensity
```

  - Definitions loaded from **`DefUtils`**
  - Header: **`Model`** access-point to all ElmerSolver inside data; Node number **`N`**; input value **`T`**

Elmer

Software Development Practices
APIs for Solver and UDF

ElmerTeam

CSC – IT Center for Science, Finland

CSC, 2018

Elmer programming languages

- Fortran90 (and newer)
  - ElmerSolver (~¨300,000 lines of which ~50% in DLLs)
- C++
  - ElmerGUI (~18,000 lines)
  - ElmerSolver (~15,000 lines)
- C
  - ElmerGrid (~30,000 lines)
  - MATC (~11,000 lines)
  - ElmerPost (~45,000 lines)

Tools for Elmer development

- Programming languages
  - Fortran90 (and newer), C, C++
- Compilation
  - Compiler (e.g. gnu), configure, automake, make, (cmake)
- Editing
  - emacs, vi, notepad++,…
- Code hosting (git)
  - https://github.com/ElmerCSC
- Consistency tests
  - Currently around 450
- Code documentation
  - Doxygen
- Theory documentation

Elmer libraries

- ElmerSolver
  - Required: Matc, HutIter, Lapack, Blas, Umfpack (GPL)
  - Optional: Arpack, Mumps, Hypre, Pardiso, Trilinos, SuperLU, Cholmod, NetCDF, HDF5, ...
- ElmerGUI
  - Required: Qt, ElmerGrid, Netgen
  - Optional: Tetgen, OpenCASCADE, VTK, QVT

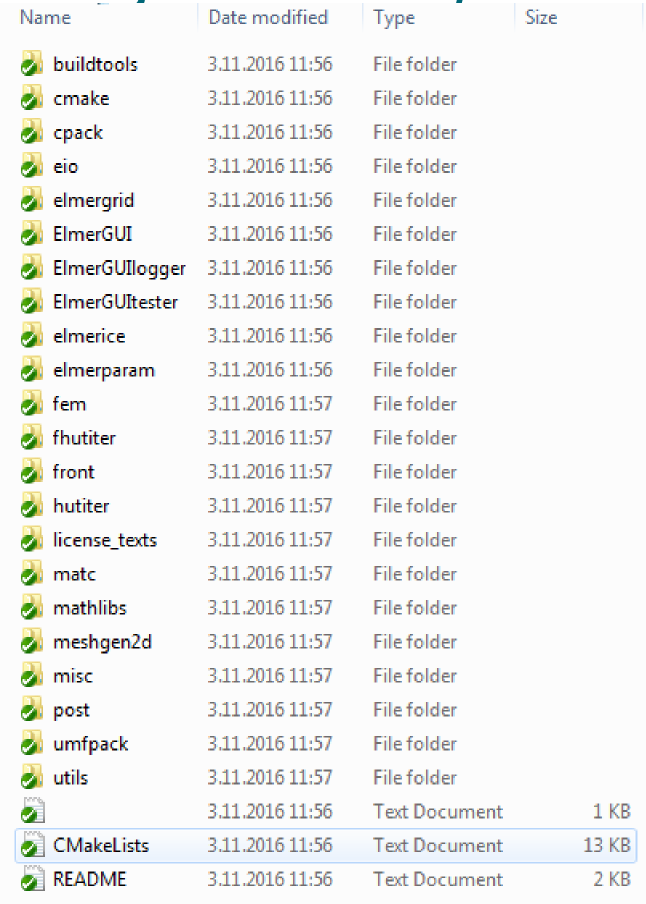
Elmer licenses

- ElmerSolver library is published under LGPL
  - Enables linking with all license types
  - It is possible to make a new solver even under proprierity license
  - Note: some optional libraries may constrain this freedom due to use of GPL licences
- Most other parts of Elmer published under GPL
  - Derived work must also be under same license (“copyleft”)
- Proprierity modules linked with ElmerSolver may be freely licensed if they are not derived work
  - Note that you must not violete licences of other libraries

Elmer version control at GitHub

- In 2015 the official version control of Elmer was transferred from svn at sf.net to git hosted at GitHub
- Git offers more flexibility over svn
  - Distributed version control system
  - Easier to maintain several development branches
  - More options and hence also steeper learning curve
  - Developed by Linus Torvalds to host Linux kernel development
- GitHub is a portal providing Git and some additional servives
  - Management of user rights
  - Controlling pull requests

Directory listing of elmerfem/trunk with TortoiseGIT:



Name	Date modified	Type	Size
buildtools	3.11.2016 11:56	File folder	
cmake	3.11.2016 11:56	File folder	
cpack	3.11.2016 11:56	File folder	
eio	3.11.2016 11:56	File folder	
elmergrid	3.11.2016 11:56	File folder	
ElmerGUI	3.11.2016 11:56	File folder	
ElmerGUIlogger	3.11.2016 11:56	File folder	
ElmerGUItester	3.11.2016 11:56	File folder	
elmerice	3.11.2016 11:56	File folder	
elmerparam	3.11.2016 11:56	File folder	
fem	3.11.2016 11:57	File folder	
fhutiter	3.11.2016 11:57	File folder	
front	3.11.2016 11:57	File folder	
hutiter	3.11.2016 11:57	File folder	
license_texts	3.11.2016 11:57	File folder	
matc	3.11.2016 11:57	File folder	
mathlibs	3.11.2016 11:57	File folder	
meshgen2d	3.11.2016 11:57	File folder	
misc	3.11.2016 11:57	File folder	
post	3.11.2016 11:57	File folder	
umfpack	3.11.2016 11:57	File folder	
utils	3.11.2016 11:57	File folder	
	3.11.2016 11:56	Text Document	1 KB
CMakeLists	3.11.2016 11:56	Text Document	13 KB
README	3.11.2016 11:56	Text Document	2 KB

ElmerGrid mesh manipulation
ElmerGUI graphical user interface

Elmer/ICE community developments
ElmerParam optimization module
ElmerSolver library and modules

HUTiter Krylov methods library
ElmerFront: Initial user interface (obsolite)

MATC library
Basic math libraries
Mesh2D (Delaunay triangularization,obsolite)

ElmerPost: Initial postprocessor (obsolite)
Umfpack sparse direct solver undel GPL

Cmake build system

- During 2014-2015 Elmer was migrated from gnu autotools into cmake
- Cmake offers several advantages
  - Enables cross compilation for diffirent platforms (e.g. Intel MICs)
  - More standardizes installation scripts
  - Straight-forward package creation for many systems (using cpack)
  - Great testing utility with ctest – now also in parallel
- Transition to cmake required significant code changes
  - ISO C-bindings & many changes in APIs
  - Backward compatibility in compilation lost

Compiling fresh Elmer source from GitHub

```
# clone the git repository.
$ git clone https://www.github.com/ElmerCSC/elmerfem

# Switch to devel branch (currently the default branch)
$ cd elmerfem
$ git checkout devel
$ cd ..

# create build directory
$ mkdir build
$ cd build
```

```
$ cmake -DWITH_ELMERGUI:BOOL=FALSE -
DWITH_MPI:BOOL=FALSE -
DCMAKE_INSTALL_PREFIX=../install ../elmerfem
```

```
$ cmake <flags>
# You can tune the compilation parameters graphically with $ ccmake or $cmake-gui .

$ make install
# or alternatively compile in parallel (4 procs) $ make -j4 install
```

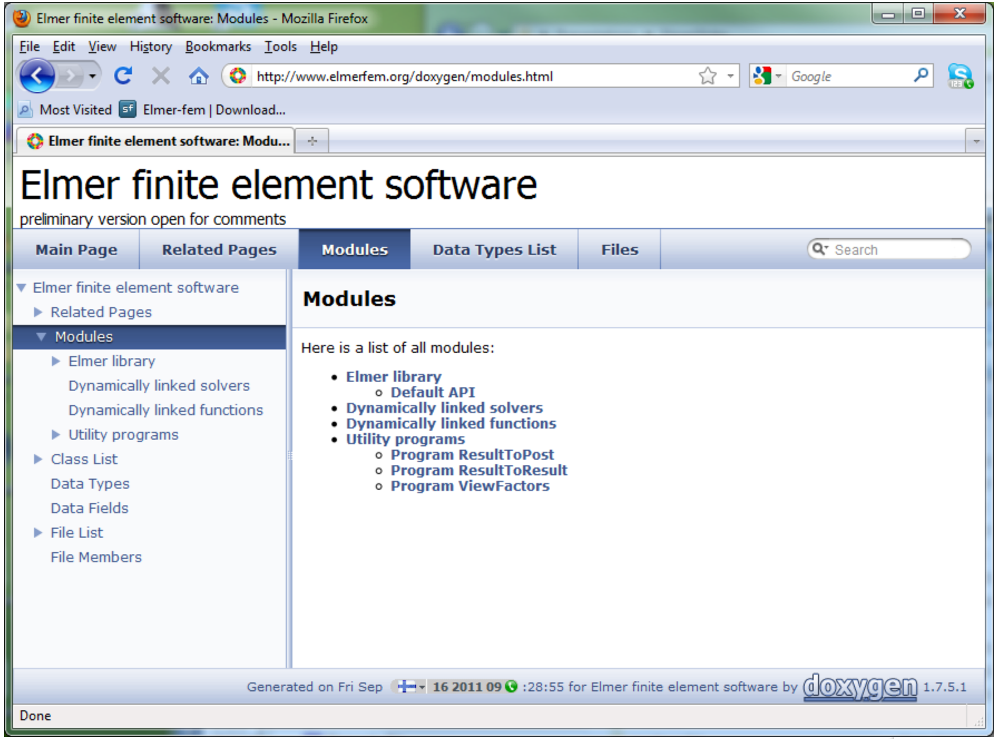
Consistency tests

- Utilize ctest system to run a set of Elmer cases
  - Upon success each case writes 1 to file TEST.PASSED, and on failure 0, respectively
- There are more than 580 consistency tests (May 2018)
  - Located under fem/tests
- Each time a significant commit is made the tests are run with the fresh version
  - Aim: even devel version is a stable
  - New tests for each major new feature
- The consistency tests provide a good starting point for taking some Solver into use
  - cut-paste from sif file

Executing the consistency tests of Elmer

```
>ctest -j4 -LE elmerice
      Start 143: mgdyn_torus_harmonic
        Start 304: ThermalActuator
        Start 344: RotatingBCMagnetoDynamicsGeneric
  1/310 Test #344: RotatingBCMagnetoDynamicsGeneric ...   Passed   43.18 sec
        Start 293: mgdyn_lamstack_lowfreq_harmonic
  2/310 Test #304: ThermalActuator ....................   Passed   59.78 sec
        Start 222: mgdyn_transient_loss
  3/310 Test #293: mgdyn_lamstack_lowfreq_harmonic ....   Passed   21.80 sec
        Start 322: mgdyn_bh

…

308/310 Test  #46: CoupledPoisson7 ....................   Passed    0.38 sec
309/310 Test #212: CoordinateScaling ..................   Passed    0.38 sec
        Start  54: RotatingBCPoisson3DSymmSkev
310/310 Test  #54: RotatingBCPoisson3DSymmSkev ........   Passed    6.34 sec

100% tests passed, 0 tests failed out of 310

Total Test time (real) = 365.62 sec
```

Doxygen – WWW documentation



Elmer finite element software: Modules - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.elmerfem.org/doxygen/modules.html
Google
Most Visited
Elmer-fem | Download...
Elmer finite element software: Modu...

Elmer finite element software

preliminary version open for comments

Main Page
Related Pages
Modules
Data Types List
Files
Search
Elmer finite element software
Related Pages
Modules
Elmer library
Dynamically linked solvers
Dynamically linked functions
Utility programs
Class List
Data Types
Data Fields
File List
File Members

Modules

Here is a list of all modules:

- Elmer library
  - Default API
- Dynamically linked solvers
- Dynamically linked functions
- Utility programs
  - Program ResultToPost
  - Program ResultToResult
  - Program ViewFactors

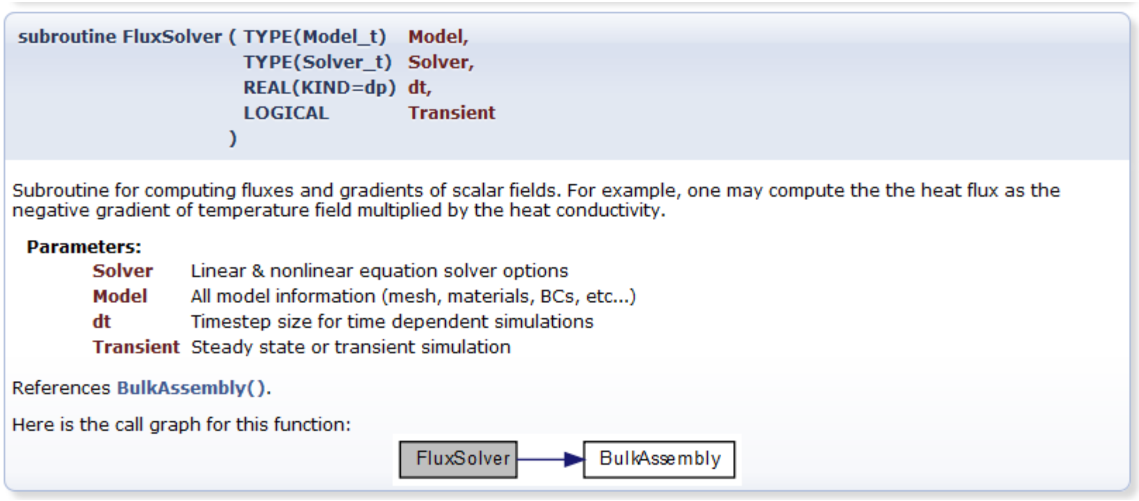
Generated on Fri Sep 16 2011 09 :28:55 for Elmer finite element software by doxygen 1.7.5.1

Doxygen – Example in code

- Special comment indicators: !> and <!

```
!------------------------------------------------------------------------------
!> Subroutine for computing fluxes and gradients of scalar fields.
!> For example, one may compute the the heat flux as the negative gradient of temperature
!> field multiplied by the heat conductivity.
!> \ingroup Solvers
!------------------------------------------------------------------------------
SUBROUTINE FluxSolver( Model,Solver,dt,Transient )
!------------------------------------------------------------------------------
USE CoordinateSystems
USE DefUtils
IMPLICIT NONE
!------------------------------------------------------------------------------
TYPE(Solver_t) :: Solver  !< Linear & nonlinear equation solver options
TYPE(Model_t) :: Model    !< All model information (mesh, materials, BCs, etc...)
REAL(KIND=dp) :: dt       !< Timestep size for time dependent simulations
LOGICAL :: Transient      !< Steady state or transient simulation
!------------------------------------------------------------------------------
!    Local variables
!------------------------------------------------------------------------------
TYPE(ValueList_t),POINTER :: SolverParams
```

Doxygen – Example in WWW



```
subroutine FluxSolver ( TYPE(Model_t)   Model,
                        TYPE(Solver_t)  Solver,
                        REAL(KIND=dp)   dt,
                        LOGICAL         Transient
                      )
```

Subroutine for computing fluxes and gradients of scalar fields. For example, one may compute the the heat flux as the negative gradient of temperature field multiplied by the heat conductivity.

Parameters:

Solver	Linear & nonlinear equation solver options
Model	All model information (mesh, materials, BCs, etc...)
dt	Timestep size for time dependent simulations
Transient	Steady state or transient simulation

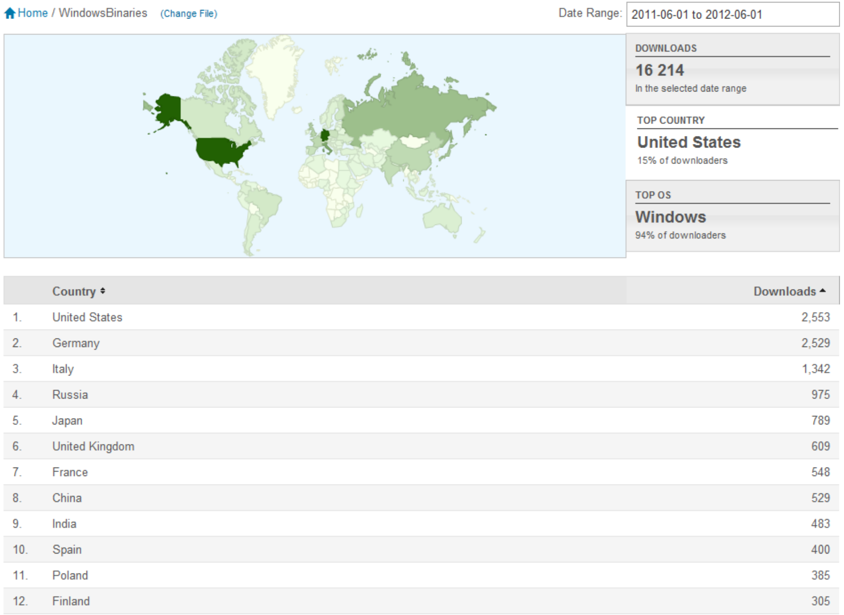
References **BulkAssembly()**.

Here is the call graph for this function:

Installers

- Fresh Windows installers
  - Currently only 64 bit version
  - Also a parallel version with msmpi
  - http://www.nic.funet.fi/pub/sci/physics/elmer/bin/windows/
  - Some version available also at sf.net
- Elmer for Debian & Ubuntu etc. at launchpad
  - Nightly builds from Git repository
  - To install

```
$ sudo apt-add-repository ppa:elmer-csc-ubuntu/elmer-csc-ppa
$ sudo apt-get update
$ sudo apt-get install elmerfem-csc
```



Home / WindowsBinaries (Change File)

Date Range: 2011-06-01 to 2012-06-01

DOWNLOADS
16 214
In the selected date range

TOP COUNTRY
United States
15% of downloaders

TOP OS
Windows
94% of downloaders

	Country	Downloads
1.	United States	2,553
2.	Germany	2,529
3.	Italy	1,342
4.	Russia	975
5.	Japan	789
6.	United Kingdom	609
7.	France	548
8.	China	529
9.	India	483
10.	Spain	400
11.	Poland	385
12.	Finland	305

Compilation of a DLL module

- Applies both to Solvers and User Defined Functions (UDF)
- Assumes that there is a working compile environment that provides ”**`elmerf90`**” script
    - Comes with the Windows installer, and Linux packages
    - Generated automatically when ElmerSolver is compiled

```
elmerf90 MySolver.F90 –o MySolver.so
```

User defined function API

```
!------------------------------------------------------
!> Standard API for UDF
!------------------------------------------------------
FUNCTION MyProperty( Model, n, t ) RESULT(f)
!------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------
  TYPE(Model_t) :: Model    !< Handle to all data
  INTEGER :: n              !< Current node
  REAL(KIND=dp) :: t        !< Parameter(s)
  REAL(KIND=dp) :: f        !< Parameter value at node
!------------------------------------------------------
  Actual code …
```

Function API

```
MyProperty = Variable time
   ”MyModule" ”MyProperty”
```

- User defined function (UDF) typically returns a real valued property at a given point
- It can be located in any section that is used to fetch these values from a list
    - Boundary Condition, Initial Condition, Material,…

Solver API

```
!------------------------------------------------------
!> Standard API for Solver
!------------------------------------------------------
SUBROUTINE MySolver( Model,Solver,dt,Transient )
!------------------------------------------------------
  USE DefUtils
  IMPLICIT NONE
!------------------------------------------------------
  TYPE(Solver_t) :: Solver  !< Current solver
  TYPE(Model_t) :: Model    !< Handle to all data
  REAL(KIND=dp) :: dt       !< Timestep size
  LOGICAL :: Transient      !< Time-dependent or not
!------------------------------------------------------
  Actual code …
```

Solver API

```
Solver 1
   Equation = ″MySolver"
   Procedure = ″MyModule" ″MySolver″
   …
End
```

- Solver is typically a FEM implementation of a physical equation
- But it could also be an auxiliary solver that does something completely different
- Solver is usually called once for each coupled system iteration

Elmer – High level abstractions

- The quite good success of Elmer as a multiphysics code may be addressed to certain design choices
  - Solver is an asbtract dynamically loaded object
  - Parameter value is an abstract property fecthed from a list
- The abstractions mean that new solvers may be implemented without much need to touch the main library
  - Minimizes need of central planning
  - Several applications fields may live their life quite independently (electromagnetics vs. glaceology)
- MATC – a poor man’s Matlab adds to flexibility as algebraic expressions may be evalueted on-the-fly

Solver as an abstract object

- Solver is an dynamically loaded object (.dll or .so)
  - May be developed and compiled seperately
- Solver utilizes heavily common library utilities
  - Most common ones have interfaces in DefUtils
- Any solver has a handle to all of the data
- Typically a solver solves a weak form of a differential equation
- Currently ~60 different Solvers,
  roughly half presenting physical phenomena
  - No upper limit to the number of Solvers
  - Often cases include ~10 solvers
- Solvers may be active in different domains,
  and even meshes
- The menu structure of each solver in ElmerGUI may be defined by an `.xml` file

Property as an abstract object

- Properties are saved in a list structure by their name
- Namespace of properties is not fixed, they may be introduced in the command file
  - E.g. "`MyProperty = Real 1.23`" adds a property "MyProperty" to a list structure related to the solver block
- In code parameters are fetched from the list
  - E.g. "`val = GetReal( Material,'MyProperty',Found)`" retrieves the above value 1.23 from the list
- A "`Real`" property may be any of the following
  - Constant value
  - Linear or cubic dependence via table of values
  - Expression given by MATC (MatLab-type command language)
  - User defined functions with arbitrary dependencies
  - Real vector or tensor
- As a result solvers may be weakly coupled without any *a priori* defined manner
- There is a price to pay for the generic approach but usually it is less than 10%
- `SOLVER.KEYWORDS` file may be used to give the types for the keywords in the command file

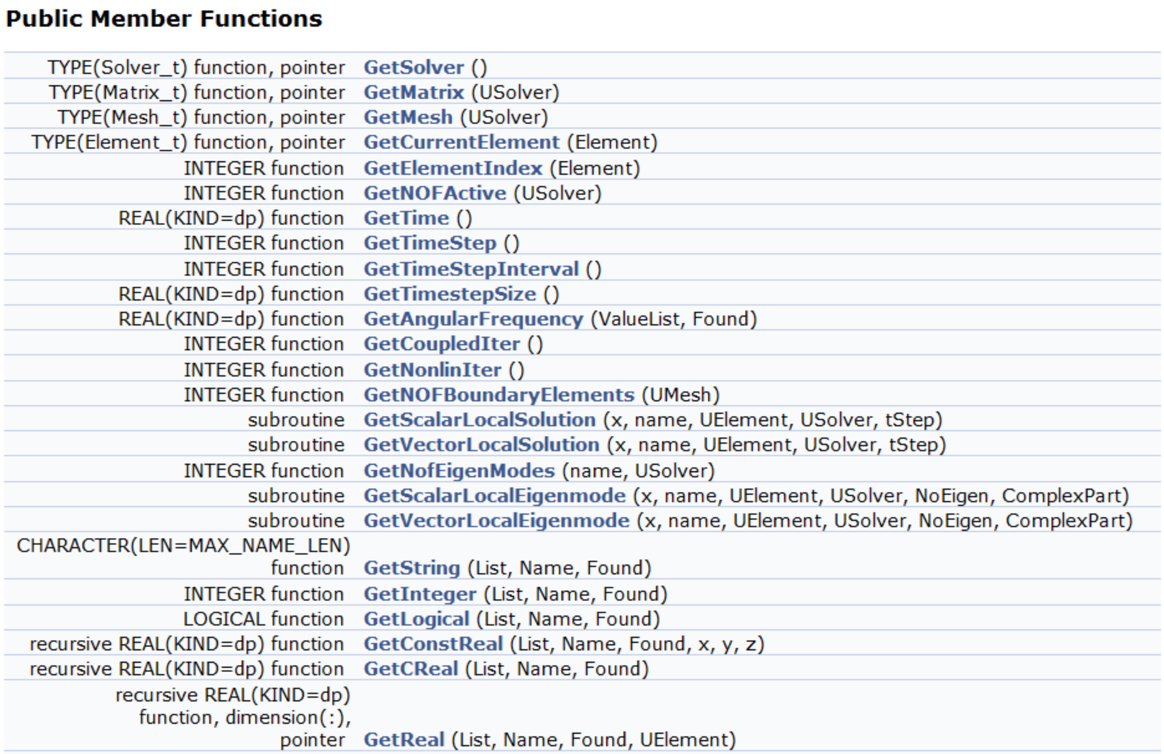
Code structure

- Elmer code structure has evolved over the years
  - There has been no major restructuring operations
- Ufortunately there is no optimal hierarchy and the number of subroutines is rather large
  - ElmerSolver library consists of more than ~40 modules
  - There are all-in-all around 1050 SUBROUTINES and 650 FUNCTIONS (both internal and external)
- To ease the learning curve the most important routines for basic use have been collected into module DefUtils.F90

DefUtils

- DefUtils module includes wrappers to the basic tasks common to standard solvers
    - E.g. ”**`DefaultDirichlet()`**” sets Dirichlet boundary conditions to the given variable of the Solver
    - E.g. ”**`DefaultSolve()`** ” solves linear systems with all available direct, iterative and multilevel solvers, both in serial and parallel
- Programming new Solvers and UDFs may usually be done without knowledge of other modules

DefUtils – some functions



Public Member Functions

TYPE(Solver_t) function, pointer	**GetSolver** ()
TYPE(Matrix_t) function, pointer	**GetMatrix** (USolver)
TYPE(Mesh_t) function, pointer	**GetMesh** (USolver)
TYPE(Element_t) function, pointer	**GetCurrentElement** (Element)
INTEGER function	**GetElementIndex** (Element)
INTEGER function	**GetNOFActive** (USolver)
REAL(KIND=dp) function	**GetTime** ()
INTEGER function	**GetTimeStep** ()
INTEGER function	**GetTimeStepInterval** ()
REAL(KIND=dp) function	**GetTimestepSize** ()
REAL(KIND=dp) function	**GetAngularFrequency** (ValueList, Found)
INTEGER function	**GetCoupledIter** ()
INTEGER function	**GetNonlinIter** ()
INTEGER function	**GetNOFBoundaryElements** (UMesh)
subroutine	**GetScalarLocalSolution** (x, name, UElement, USolver, tStep)
subroutine	**GetVectorLocalSolution** (x, name, UElement, USolver, tStep)
INTEGER function	**GetNofEigenModes** (name, USolver)
subroutine	**GetScalarLocalEigenmode** (x, name, UElement, USolver, NoEigen, ComplexPart)
subroutine	**GetVectorLocalEigenmode** (x, name, UElement, USolver, NoEigen, ComplexPart)
CHARACTER(LEN=MAX_NAME_LEN) function	**GetString** (List, Name, Found)
INTEGER function	**GetInteger** (List, Name, Found)
LOGICAL function	**GetLogical** (List, Name, Found)
recursive REAL(KIND=dp) function	**GetConstReal** (List, Name, Found, x, y, z)
recursive REAL(KIND=dp) function	**GetCReal** (List, Name, Found)
recursive REAL(KIND=dp) function, dimension(:), pointer	**GetReal** (List, Name, Found, UElement)

Modules related to linear algebra

BandMatrix.F90
BandwidthOptimize.F90
BlockSolve.F90
cholmod.c
CircuitUtils.F90
ClusteringMethods.F90
CRSMatrix.F90
DirectSolve.F90
EigenSolve.F90
IterativeMethods.F90
IterSolve.F90
LinearAlgebra.F90
LUDecomposition.F90
MGPrec.F90
Multigrid.F90
Smoothers.F90
SolveBand.F90
SolveHypre.c
SolverUtils.F90
SolveSBand.F90
SolveSuperLU.c
SolveTrilinos.cxx

Modules related to space and time discretization

ElementDescription.F90
ElementUtils.F90
H1ElementBasisFunctions.F90
PElementBase.F90
PElementMaps.F90
TimeIntegrate.F90

Historical modules including physics

Differentials.F90

DiffuseConvectiveAnisotropic.F90

DiffuseConvectiveGeneralAnisotropic.F90

ExchangeCorrelations.F90

MaxwellAxiS.F90

Maxwell.F90

MaxwellGeneral.F90

NavierStokesCylindrical.F90

NavierStokes.F90

NavierStokesGeneral.F90

Stress.F90

StressGeneral.F90

VelocityUpdate.F90

Walls.F90

Example: Poisson equation

$$-\nabla^2\phi = \rho$$

- Implemented as an dynamically linked solver
  - Available under tests/1dtests
- Compilation by:

```
Elmerf90 Poisson.F90 –o Poisson.so
```

- Execution by:

```
ElmerSolver case.sif
```

- The example is ready to go massively parallel and with all a plethora of elementtypes in 1D, 2D and 3D

Poisson equation: code Poisson.F90

```
!------------------------------------------------------------------------------
!> Solve the Poisson equation -\nabla\cdot\nabla \phi = \rho
!------------------------------------------------------------------------------
SUBROUTINE PoissonSolver( Model,Solver,dt,TransientSimulation )
!------------------------------------------------------------------------------
 USE DefUtils
 IMPLICIT NONE
 …

  !Initialize the system and do the assembly:
  !------------------------------------------
  CALL DefaultInitialize()

  active = GetNOFActive()
  DO t=1,active
    Element => GetActiveElement(t)
    n = GetElementNOFNodes()

    LOAD = 0.0d0
    BodyForce => GetBodyForce()
    IF ( ASSOCIATED(BodyForce) ) &
      Load(1:n) = GetReal( BodyForce, 'Source', Found )

    ! Get element local matrix and rhs vector:
    !----------------------------------------
    CALL LocalMatrix(  STIFF, FORCE, LOAD, Element, n )

    ! Update global matrix and rhs vector from local contribs
    !-------------------------------------------------------------
    CALL DefaultUpdateEquations( STIFF, FORCE )
  END DO

  CALL DefaultFinishAssembly()
  CALL DefaultDirichletBCs()
  Norm = DefaultSolve()
```

```
CONTAINS

!------------------------------------------------------------------------------
 SUBROUTINE LocalMatrix(  STIFF, FORCE, LOAD, Element, n )
!------------------------------------------------------------------------------

…

CALL GetElementNodes( Nodes )
  STIFF = 0.0d0
  FORCE = 0.0d0

  ! Numerical integration:
  !----------------------
  IP = GaussPoints( Element )
  DO t=1,IP % n
    ! Basis function values & derivatives at the integration point:
    !--------------------------------------------------------------
    stat = ElementInfo( Element, Nodes, IP % U(t), IP % V(t), &
           IP % W(t),  detJ, Basis, dBasisdx )

    ! The source term at the integration point:
    !------------------------------------------
    LoadAtIP = SUM( Basis(1:n) * LOAD(1:n) )

    ! Finally, the elemental matrix & vector:
    !----------------------------------------
    STIFF(1:n,1:n) = STIFF(1:n,1:n) + IP % s(t) * DetJ * &
       MATMUL( dBasisdx, TRANSPOSE( dBasisdx ) )
    FORCE(1:n) = FORCE(1:n) + IP % s(t) * DetJ * LoadAtIP * Basis(1:n)
  END DO
!------------------------------------------------------------------------------
 END SUBROUTINE LocalMatrix
!------------------------------------------------------------------------------
END SUBROUTINE PoissonSolver
!------------------------------------------------------------------------------
```

Poisson equation: command file case.sif

```
Check Keywords "Warn"

Header
  Mesh DB "." ”mesh"
End

Simulation
  Coordinate System = "Cartesian"
  Simulation Type = Steady State
  Steady State Max Iterations = 50
End

Body 1
  Equation = 1
  Body Force = 1
End

Equation 1
  Active Solvers(1) = 1
End

Solver 1
  Equation = "Poisson"
  Variable = "Potential"
  Variable DOFs = 1
  Procedure = "Poisson" "PoissonSolver"
  Linear System Solver = "Direct”
  Linear System Direct Method = umfpack
  Steady State Convergence Tolerance = 1e-09
End
```

```
Body Force 1
 Source = Variable Potential
  Real Procedure "Source" "Source"
End

Boundary Condition 1
  Target Boundaries(2) = 1 2
  Potential = Real 0
End
```

Poisson equation: source term, examples

Constant source:

```
Source = 1.0
```

Source dependeing piecewise linear on x:

```
Source = Variable Coordinate 1
    Real
      0.0 0.0
      1.0 3.0
      2.0 4.0
    End
```

Source depending on x and y:

```
Source = Variable Coordinate
  Real MATC ”sin(2*pi*tx(0))*cos(2*pi(tx(1))”
```

Source depending on anything

```
Source = Variable Coordinate 1
  Procedure ”Source” ”MySource”
```

Poisson equation: ElmerGUI menus

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE edf>
<edf version="1.0" >
  <PDE Name="Poisson" >
    <Name>Poisson</Name>

    <BodyForce>
    <Parameter Widget="Label" > <Name> Properties </Name> </Parameter>
       <Parameter Widget="Edit" >
        <Name> Source </Name>
        <Type> String </Type>
        <Whatis> Give the source term. </Whatis>
      </Parameter>
    </BodyForce>

    <Solver>
      <Parameter Widget="Edit" >
       <Name> Procedure </Name>
       <DefaultValue> "Poisosn" "PoissonSolver" </DefaultValue>
      </Parameter>
     <Parameter Widget="Edit">
       <Name> Variable </Name>
       <DefaultValue> Potential</DefaultValue>
      </Parameter>
     </Solver>

    <BoundaryCondition>
      <Parameter Widget="Label" > <Name> Dirichlet conditions </Name> </Parameter>
      <Parameter  Widget="Edit">
       <Name> Potential </Name>
       <Whatis> Give potential value for this boundary. </Whatis>
      </Parameter>
    </BoundaryCondition>
  </PDE>
</edf>
```

Development tools for ElmerSolver

- Basic use
  - Editor (emacs, vi, notepad++, jEdit,…)
  - elmerf90 script
- Advanced
  - Editor
  - svn client
  - Compiler suite (gfortran, ifort, pathf90, pgf90,…)
  - Documentation tools (Doxygen, LaTeX)
  - Debugger (gdb)
  - Profiling tools
  - …

Elmer – some best practices

- Use version control when possible
  - If the code is left to your own local disk, you might as well not write it at all
  - Do not fork! (userbase of 1000's)
- Always make a consistency test for a new feature
  - Always be backward compatible
  - If not, implement a warning to the code
- Maximize the level of abstraction
  - Essential for multiphysics software
  - E.g. any number of physical equations,
    any number of computational meshes,
    any number of physical or numerical parameters – without the need for recompilation

Mesh related features in Elmer

ElmerTeam
CSC – IT Center for Science, Finland

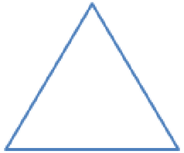
CSC, 2018

Outline

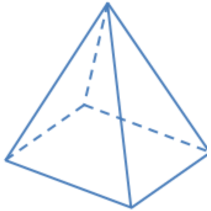
- Supported element types
  - Shapes
  - Basic functions
- Mesh generation within ElmerSolver
  - Mesh multiplication
  - Mesh extrusion
- Adaptivity – very limited
- Mesh deformation & movement
- Mesh projectors
  - Mapping between meshes
  - Mortar finite elements

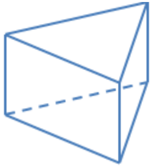
ElmerSolver – Finite element shapes

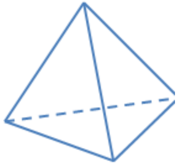
- All standard shaper of Finite Elements are supported
  - 0D: point
  - 1D: segment
  - 2D: triangles, quadrilaterals
  - 3D: tetraherdons, wedges, pyramids, hexahedrons
- Meshes may have mixed element types
- There may be also several meshes in same simulation

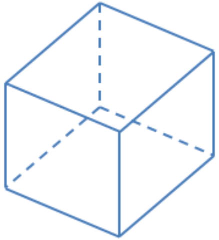

Triangle

Quadrilateral


Pyramid

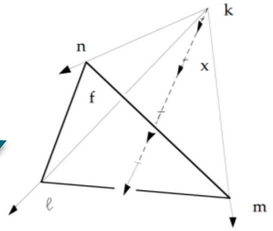

Prism with triangular base

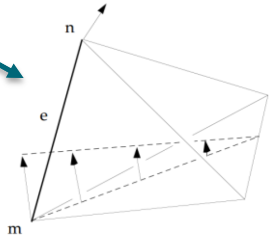

Tetrahedron


Hexahedron

ElmerSolver – basis functions

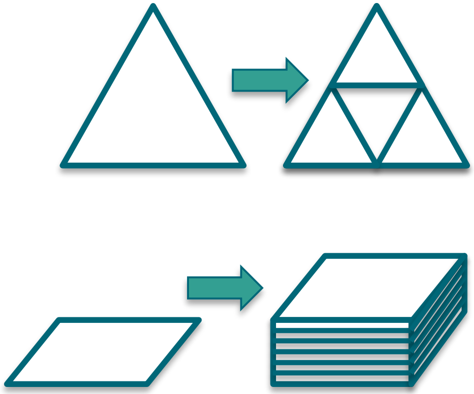
- Element families
  - Nodal (up to 2-4th degree)
  - p-elements (up to 10th degree)
  - Edge & face –elements
    - H(div) - often associated with”face” elements)
    - H(curl) - often associated with ”edge” elements)
- Formulations
  - Galerkin, Discontinuous Galerkin
  - Stabilization
  - Residual free bubbles





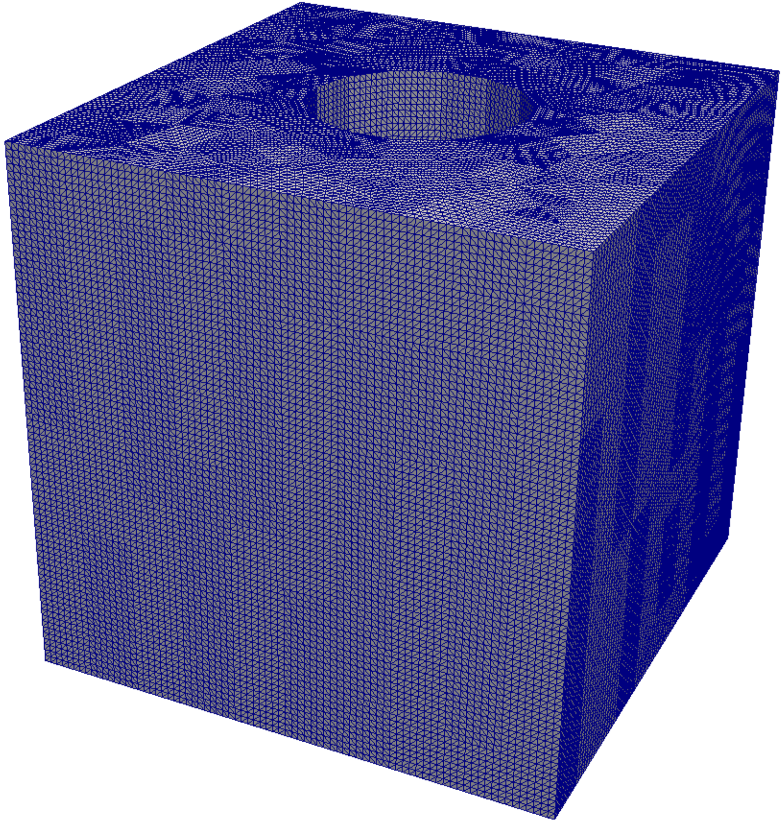
ElmerSolver – internal mesh generation

- Internal mesh division
  - 2^*DIM*^*n* -fold problem-size
  - Known as "**Mesh Multiplication**"
  - Simple inheritance of mesh grading
- Internal mesh extrusion
  - Extruded given number of layers
- Idea is to remove bottle-necks from mesh generation
  - These can also be performed on a parallel level
- Limited by generality since the internal meshing features cannot increase the geometry description



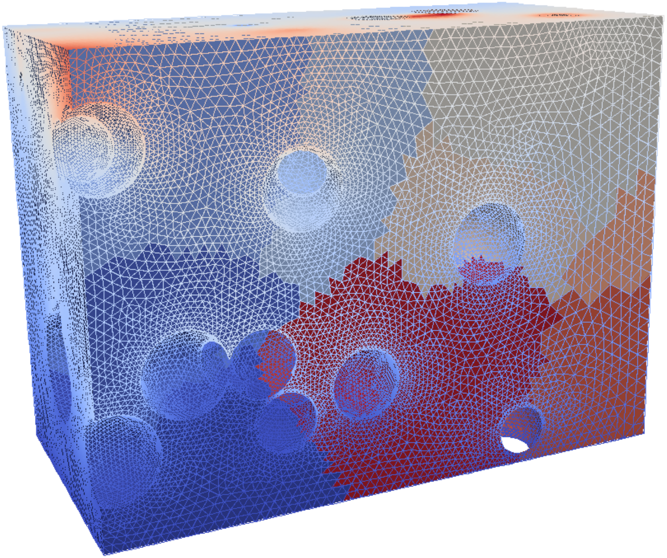
Mesh multiplication example

Mesh Levels	Number of Elements
1	7 920
2	63 360
3	506 880
4	4 055 040



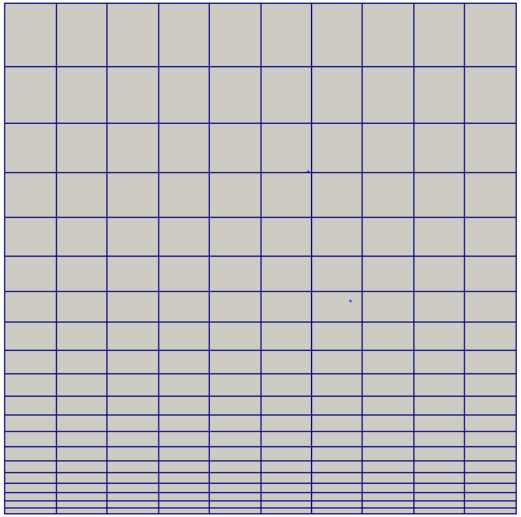
Limitations of mesh multiplication

- Standard mesh multiplication does not increase geometric accuracy
  - Polygons retain their shape
  - Mesh multiplication could be made to honor boundary shapes but this is not currently done
- Optimal mesh grading difficult to achieve
  - The coarsest mesh level does not usually have sufficient information to implement fine level grading



ElmerSolver - Internal mesh extrusion

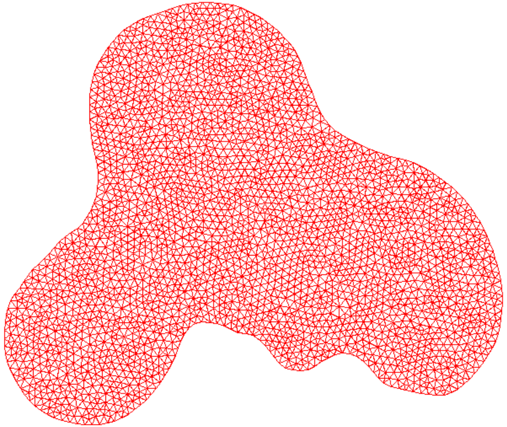
- Start from an initial 2D (1D) mesh and then extrude into 3D (2D)
  - Mesh density may be given by arbitrary function
- Implemented also for partitioned meshes
  - Extruded lines belong to the same partition by construction!
- There are many problems of practical problems where the mesh extrusion of a initial 2D mesh provides a good solution
  - One such field is glasiology where glaciers are thin, yet the 2D approach is not always sufficient in accurary



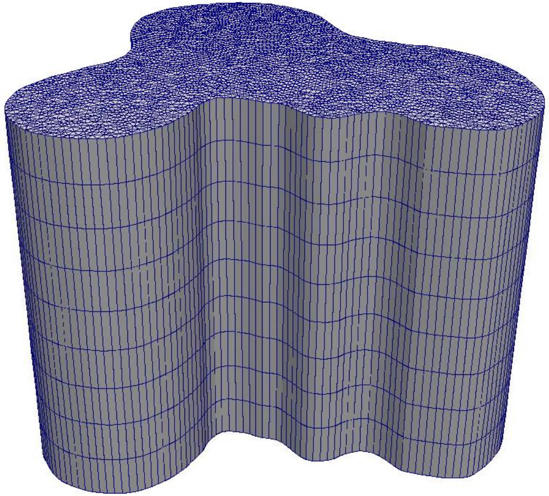
```
Extruded Mesh Levels = 21
Extruded Mesh Density =
  Variable Coordinate 1
  Real MATC "1+10*tx"
```

ElmerSolver - Internal extrusion example

Design Alvar Aalto, 1936



2D mesh by Gmsh



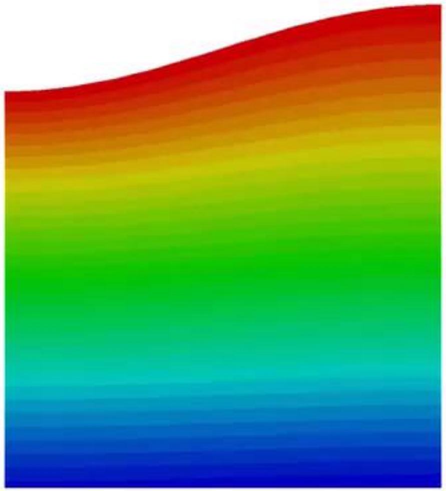
3D internally extruded mesh

Summary: Alternatives for increasing mesh resolution

- Use of higher order nodal elements
  - Elmer supports 2nd to 4th order nodal elements
  - Unfortunately not all preprocessing steps are equally well supported for higher order elements
    - E.g. Netgen output supported only for linear elements
- Use of hierarhical p-element basis functions
  - Support up to 10th degree polynomials
  - In practice **Element = p:2**, or p:3
  - Not supported in all Solvers
- Mesh multiplication
  - Subdivision of elements by splitting

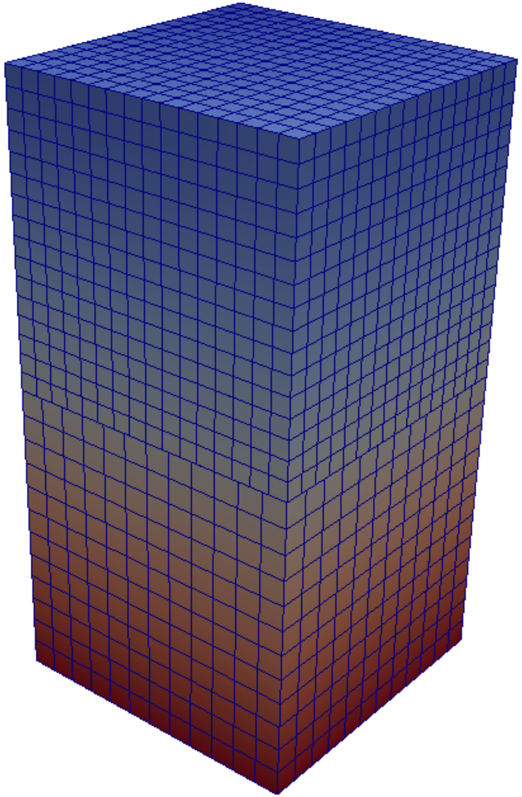
ElmerSolver – Mesh deformation

- Meshes may be internally deformed
- `MeshUpdate` solver uses linear elasticity equation to deform the mesh
- `RigidMeshMapper` uses rigid deformations and their smooth transitions to deform the mesh
- Deforming meshes have number of uses
  - Deforming structures in multiphysics simultion
    - E.g. fluid-structure interaction, ALE
  - Rotating & sliding structures
  - Geometry optimization
    - Mesh topology remains unchanged



Mapping & Projectors

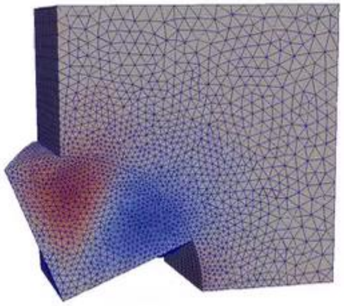
- Ensuring continuity between conforming and nonconforming meshes
  - For boundary and bulk meshes
- On-the-fly interpolation (no matrix created)
  - Mapping of finite element data
    - from mesh to mesh
    - From boundary to boundary
- Creation of interpolation and projection matrices
  - Strong continuity, interpolation: $x_l = Px_r$
  - Weak continuity, Mortar projector: $Qx_l - Px_r = 0$



Tie contact in linear elasticity using mortar finite elements

Example: Mesh utilities applied to rotational problems

- Rigid body movement may be used to implement rotation
- One of several contact pairs are used to define mortar projectors that ensure continuity of soluton
- Most important application area has been the simulation of electrical machines



Concluding remarks on internal meshing features

- Internal meshing features can be used to resolve number of challenges related to meshes
  - Accuracy
  - I/O bottle-necks
  - Continuity requirements
  - Multiphysics coupling
  - Deforming or moving computational domains

Post-processing utilities within ElmerSolver

ElmerTeam
CSC – IT Center for Science, Finland

CSC, 2018

Postprocessing utilities in ElmerSolver

- Saving data
  - FEM data
  - Line data
  - Scalars data
  - Grid data
- Computing data
  - Derived fields (gradient, curl, divecgence,…)
  - Data reduction & filtering
  - Creating fields of material properties
- The functionality is usually achieved by use of atomic auxialiry solvers

Computing derived fields

- Many solvers have internal options or dedicated post-processing solvers for computing derived fields
  - E.g. stress fields by the elasticity solvers
  - E.g. **MagnetoDynamicsCalcFields**
- Elmer offers several auxiliary solvers that may be used in a more generic way
  - `SaveMaterials`: makes a material parameter into field variable
  - `StreamlineSolver`: computes the streamlines of 2D flow
  - `FluxSolver`: given potential, computes the flux $q = - c \nabla \phi$
  - `VorticitySolver`: computes the vorticity of flow, $w = \nabla \times \phi$
  - `PotentialSolver`: given flux, compute the potential $- c \nabla \phi = q$
  - `FilterTimeSeries`: compute filtered data from time series (mean, fourier coefficients,…)
  - …

Derived nodal data

- By default Elmer operates on distributed fields but sometimes nodal values are of interest
  - Multiphysics coupling may also be performed alternatively using nodal values for computing and setting loads
- Elmer computes the nodal loads from *Ax-b* where *A*, and *b* are saved before boundary conditions are applied
  - `Calculate Loads = True`
- This is the most consistant way of obtaining boundary loads
- Note: the nodal data is really pointwise
  - expressed in units N, C, W etc.
    (rather than N/m^2, C/m^2, W/m^2 etc.)
  - For comparison with distributed data divided by the ~size of the surface elements

Derived lower dimensional data

- Derived boundary data
  - SaveLine: Computes fluxes on-the-fly
- Derived lumped (or 0D) data
  - SaveScalars: Computes a large number of different quantities on-the-fly
  - FluidicForce: compute the fluidic force acting on a surface
  - ElectricForce: compute the electrostatic froce using the Maxwell stress tensor
  - Many solvers compute lumped quantities internally for later use (Capacitance, Lumped spring,...)

Exporting FEM data: ResultOutputSolve

- Currently recommened format is **VTU**
  - XML based unstructured VTK
  - Has the most complete set of features
  - Old ElmerPost format (with suffix .ep) is becoming obsolite
  - Simple way to save VTU files: **`Post File = file.vtu`**
- ResultOutputSolve offers additionally several formats
  - vtk: Visualization tookit legacy format
  - vtu: Visualization tookit XML format
  - Gid: GiD software from CIMNE: http://gid.cimne.upc.es
  - Gmsh: Gmsh software: http://www.geuz.org/gmsh
  - Dx: OpenDx software

Exporting 2D/3D data: ResultOutputSolve

An example shows how to save data in unstructured XML VTK (.vtu) files to directory ”results” in single precision binary format.

```
Solver n
  Exec Solver = after timestep
  Equation = "result output"
  Procedure = "ResultOutputSolve""ResultOutputSolver"
  Output File Name = "case"
  Output Format = String ”vtu”
  Binary Output = True
  Single Precision = True
End
```

Saving 1D data: SaveLine

- Lines of interest may be defined on-the-fly
- Data can either be saved in uniform 1D grid, or where element faces and lines intersect
- Flux computation using integration points on the boundary – not the most accurate
- By default saves all existing field variables

Saving 1D data: SaveLine…

```
Solver n
 Equation = "SaveLine"
  Procedure = File "SaveData" "SaveLine"
  Filename =  "g.dat"
  File Append = Logical True
  Polyline Coordinates(2,2) = Real 0.0 1.0 0.0 2.0
End

Boundary Condition m
   Save Line = Logical True
End
```

Computing and saving 0D data: SaveScalars

Operators on bodies

- Statistical operators
  - Min, max, min abs, max abs, mean, variance, deviation, rms
- Integral operators (quadratures on bodies)
  - volume, int mean, int variance, int rms
  - Diffusive energy, convective energy, potential energy

Operators on boundaries

- Statistical operators
  - Boundary min, boundary max, boundary min abs, max abs, mean, boundary variance, boundary deviation, boundary sum, boundary rms
  - Min, max, minabs, maxabs, mean
- Integral operators (quadratures on boundary)
  - area
  - Diffusive flux, convective flux

Other operators

- nonlinear change, steady state change, time, timestep size,…

Saving 0D data: SaveScalars…

```
Solver n
  Exec Solver = after timestep
  Equation = String SaveScalars
  Procedure = File "SaveData" "SaveScalars"
  Filename = File "f.dat"
  Variable 1 = String Temperature
  Operator 1 = String max
  Variable 2 = String Temperature
  Operator 2 = String min
  Variable 3 = String Temperature
  Operator 3 = String mean
End

Boundary Condition m
  Save Scalars = Logical True
End
```

Slots for executing postprocessing solvers

- Often the postprocessing solver need to computed only at desired slots, not at every time-step or coupled system iteration
- The execution is controlled by the ”Exec Solver” keyword
  - Exec Solver = before simulation
  - Exec Solver = after simulation
  - Exec Solver = before timesteo
  - Exec Solver = after timestep
  - Exec Solver = before saving
  - Exec Solver = after saving
- The before/after saving slot is controlled by the output intervals
  - Derived solvers often use the ”before saving” slot
  - Data is often saved with the ”after saving” slot

Case: TwelveSolvers

Natural convection with ten auxialiary solvers

Case: Motivation

- The purpose of the example is to show the flexibility of the modular structure
- The users should not be afraid to add new atomistic solvers to perform specific tasks
- A case of 12 solvers is rather rare, yet not totally unrealitistic

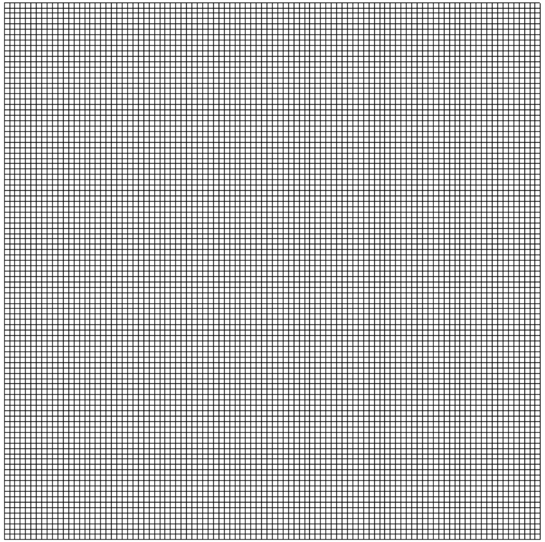
Case: preliminaries

- Square with hot wall on right and cold wall on left
- Filled with viscous fluid
- Bouyancy modeled with Boussinesq approximation
- Temperature difference initiates a convection roll

Cold wall
Hot wall

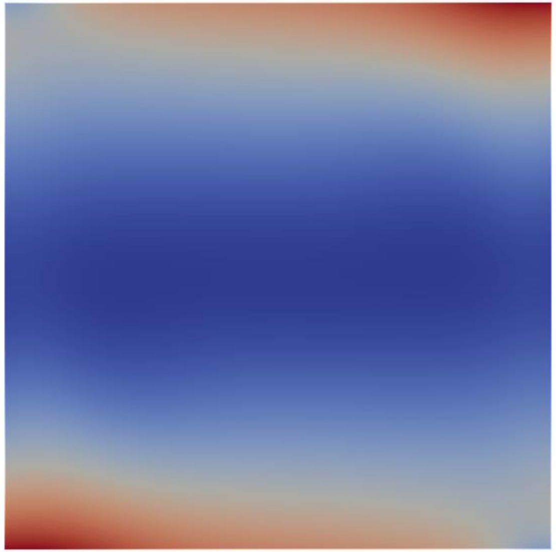
Case: 12 solvers

1. **HeatSolver**
2. **FlowSolver**
3. **FluxSolver**: solve the heat flux
4. **StreamSolver**: solve the stream function
5. **VorticitySolver**: solve the vorticity field (curl of vector field)
6. **DivergenceSolver**: solve the divergence
7. **ShearrateSolver**: calculate the shearrate
8. **IsosurfaceSolver**: generate an isosurface at given value
9. **ResultOutputSolver**: write data
10. **SaveGridData**: save data on uniform grid
11. **SaveLine**: save data on given lines
12. **SaveScalars**: save various reductions

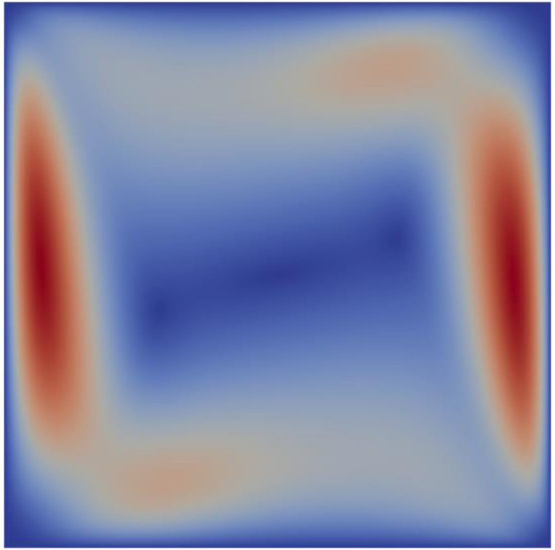


Mesh of 10000 bilinear elements

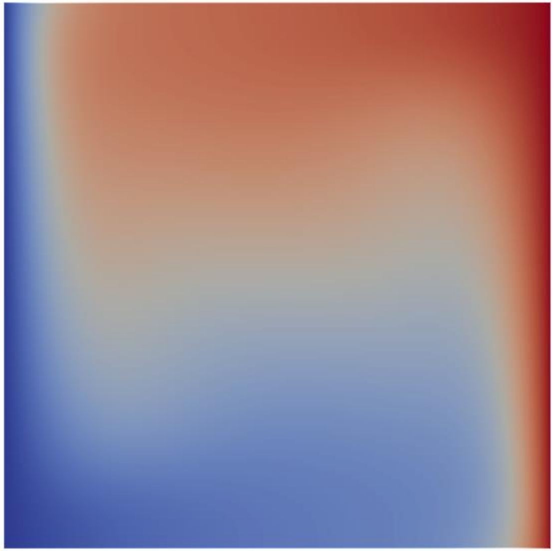
Primary fields for natural convection



Pressure

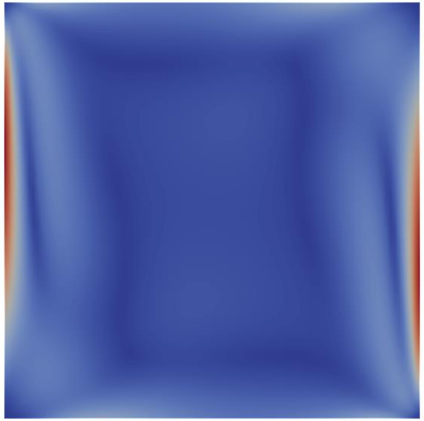


Velocity

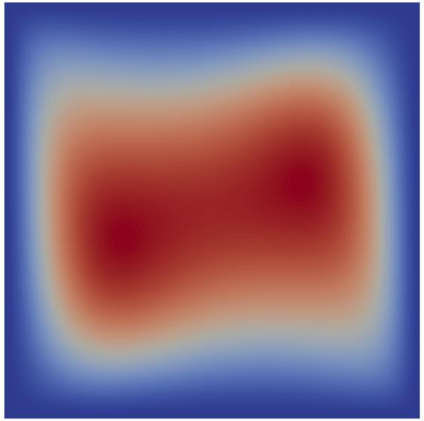


Temperature

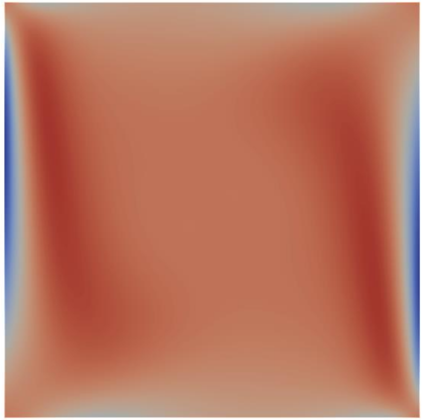
Derived fields for Navier-Stokes solution



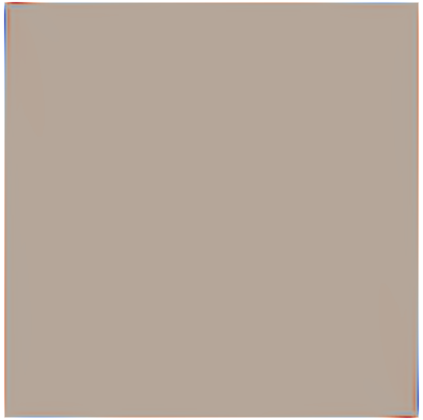
Shearrate field



Stream function

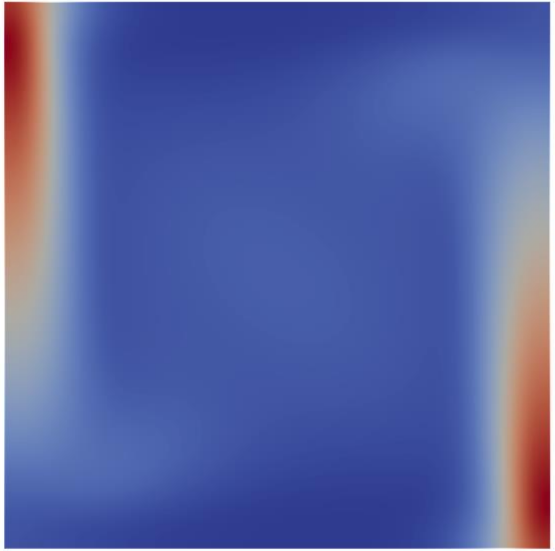


Vorticity field

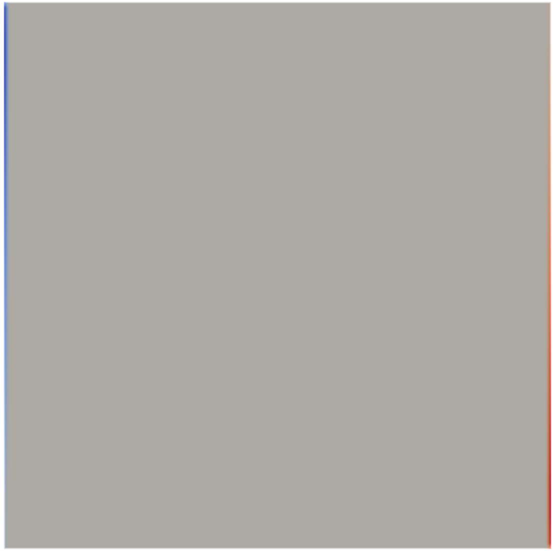


Divergence field

Derived fields for heat equation



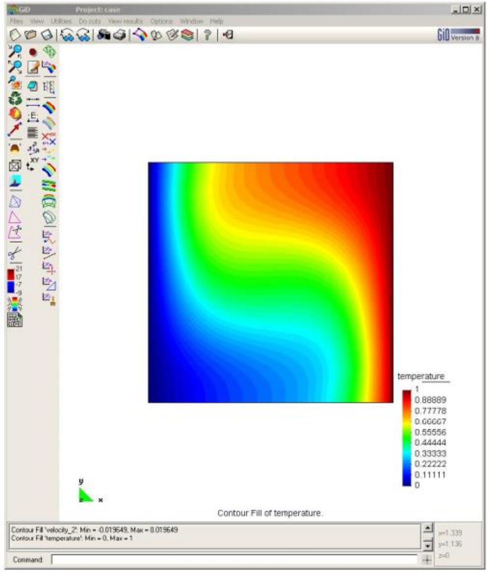
Heat flux



Nodal heat loads

- Nodal loads only occur at boundaries (nonzero heat source)
- Nodal loads are associated to continuous heat flux by element size factor

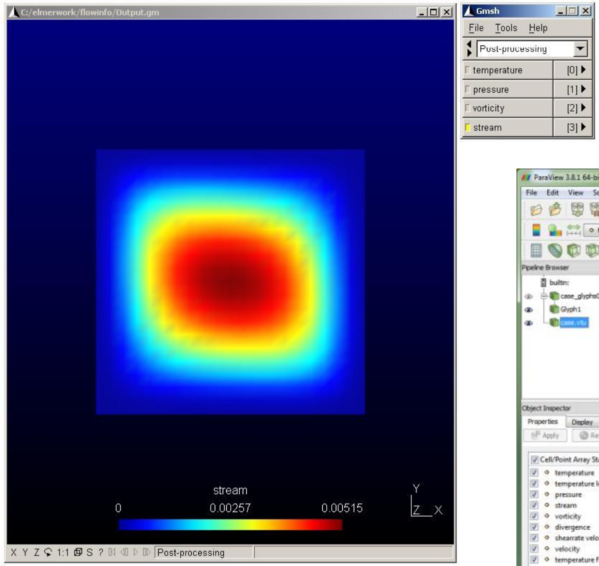
Visualization in differen postprocessors



```
Contour Fill 'velocity_2': Min = -0.019649, Max = 0.019649
Contour Fill 'temperature': Min = 0, Max = 1
```

Command:

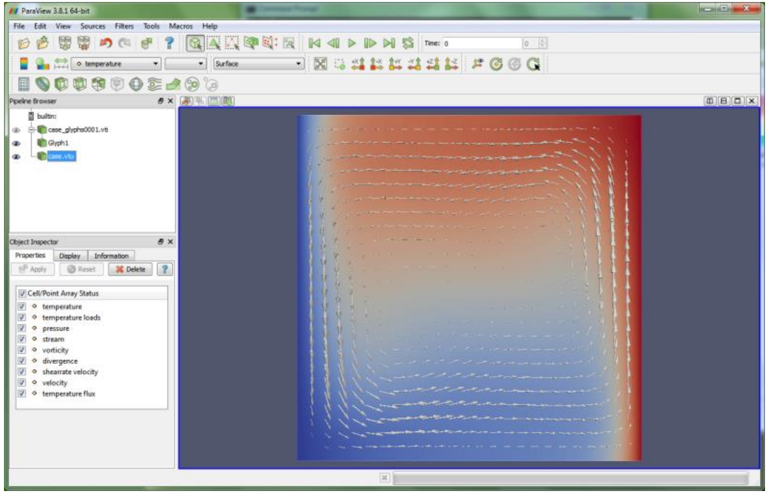
GiD



Post-processing

temperature	[0]
pressure	[1]
vorticity	[2]
stream	[3]

gmsh



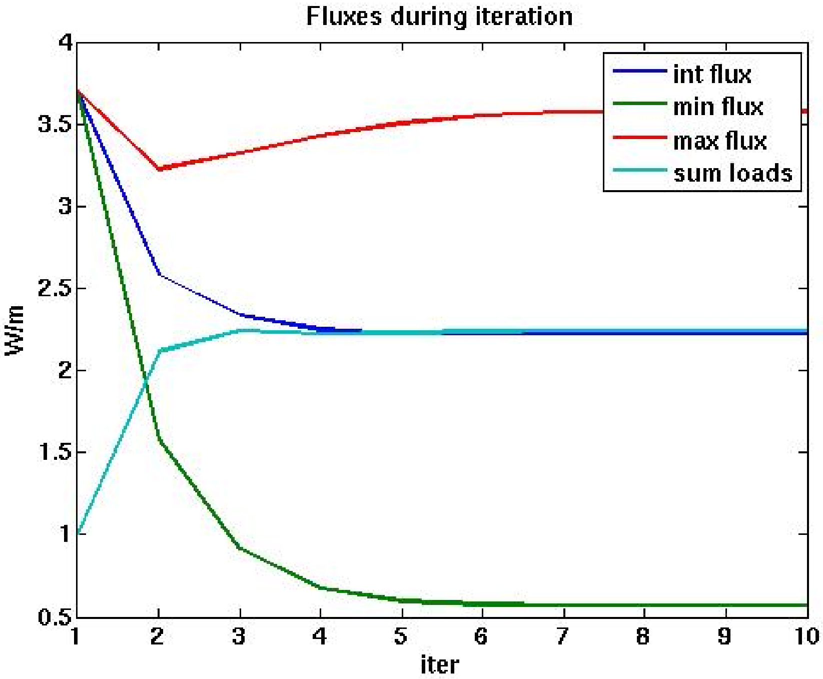
☑ Cell/Point Array Status

- ☑ temperature
- ☑ temperature loads
- ☑ pressure
- ☑ stream
- ☑ vorticity
- ☑ divergence
- ☑ shearrate velocity
- ☑ velocity
- ☑ temperature flux

Paraview

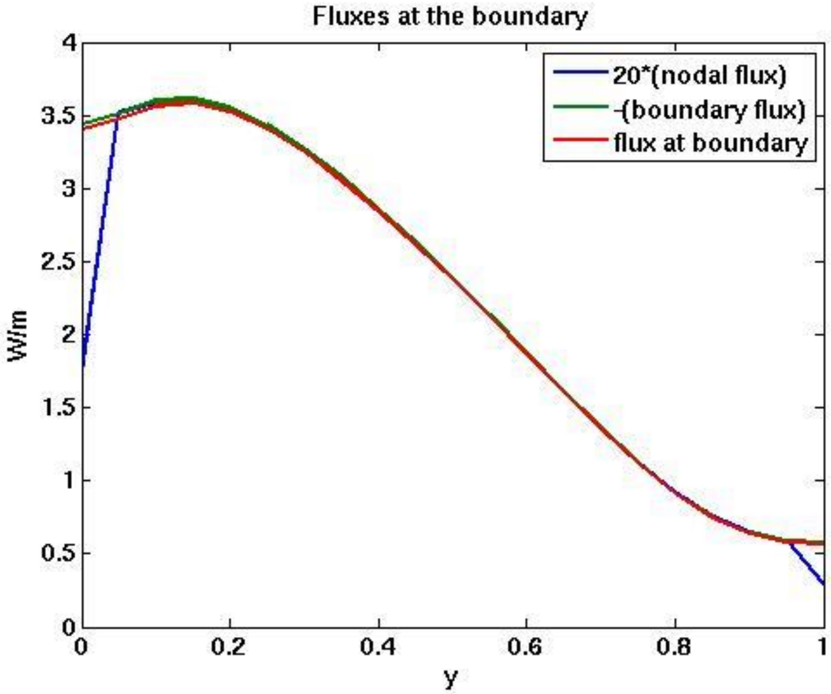
Example: total flux

- Saved by SaveScalars
- Two ways of computing the total flux give different approximations
- When convergence is reached the agreement is good



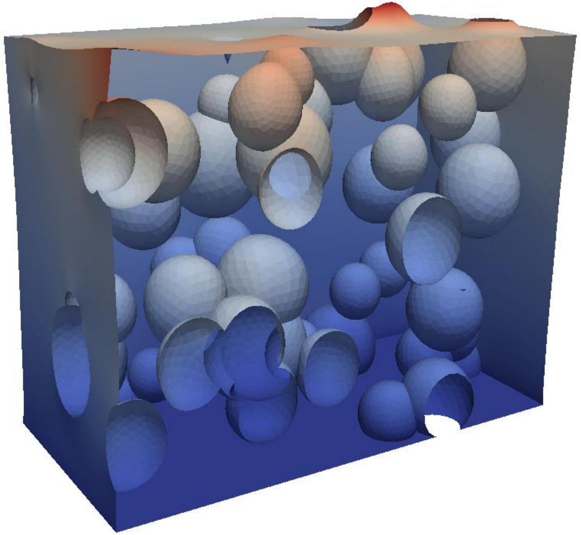
Example: boundary flux

- Saved by SaveLine
- Three ways of computing the boundary flux give different approximations
- At the corner the nodal flux should be normalized using only *h/2*



Example, saving boundaries in .sif file

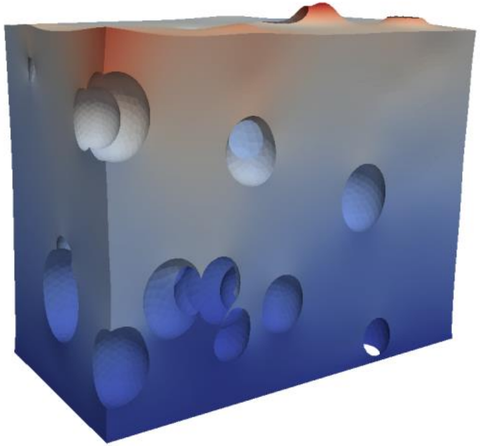
```
Solver 2
  Exec Solver = Always
  Equation = "result output"
  Procedure = "ResultOutputSolve”
"ResultOutputSolver"
  Output File Name = case
  Vtu Format = Logical True
  Save Boundaries Only = Logical True
End
```



Example, File size in Swiss Cheese

- 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



Manually editing the command files

- Only the most important solvers and features are supported by the GUI
- Minor modifications are most easily done by manual manipulation of the files
- The tutorials, test cases and documentation all include usable sif file pieces
- Use your favorite text editor (emacs, notepad++,...) and copy-paste new definitions to your .sif file
- If your additiones were sensible you can rerun your case
- Note: you cannot read in the changes made in the .sif file

Exercise

- Study the command file with 12 solvers
- Copy-paste an appropriate solver from there to some existing case of your own
  - ResultOutputSolver for VTU output
  - StreamSolver, VorticitySolver, FluxSolver,…
- Note: Make sure that the numbering of Solvers is consistant
  - Solvers that involve finite element solution you need to activate by **`Active Solvers`**
- Run the modified case
- Visualize results in Paraview in different ways

Using tests as a starting point

- There are over 500 consistancy tests that come with the Elmer distribution
  - The hope is to minimize the propability of new bugs
- The tests are small for speedy computation
- Step-by-step instructions
  1. Go to tests at
     $ELMER_HOME/tests
  2. Choose a test case relevant to you (by name, or by grep)
     - Look in Models manual for good search strings
  3. Copy the tests to your working directory
  4. Edit the sif file
     - Activate the output writing: Post File
     - Make the solver more verbose: Max Output Level
  5. Run the case (see runtest.cmake for the meshing procedure)
     - Often just: ElmerSolver
  6. Open the result file to see what you got
  7. Modify the case and rerun etc.

Conclusions

- It is good to think in advance what kind of data you need
  - 3D volume and 2D surface data
  - Derived fields
  - 1D line data
  - 0D lumped data
- Internal strategies may allow better accuracy than doing the analysis with external postprocessing software
  - Consistent use of basis functions to evaluate the data
- Often the same reduction operations may be done also at later stages but with significantly greater effort

Parallel computing with Elmer

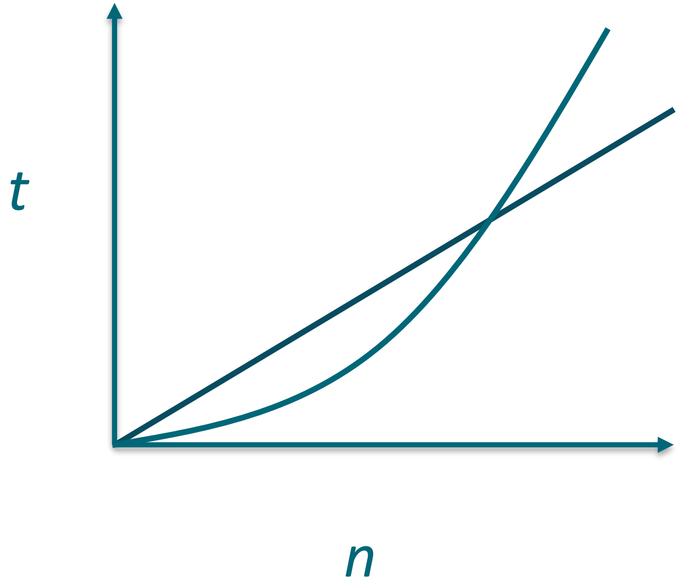
ElmerTeam

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

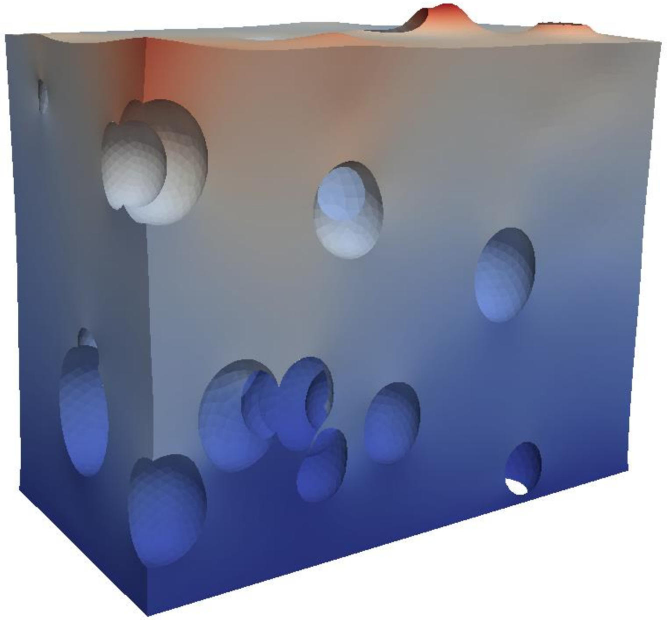
Motivation for using optimal linear solvers

- 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

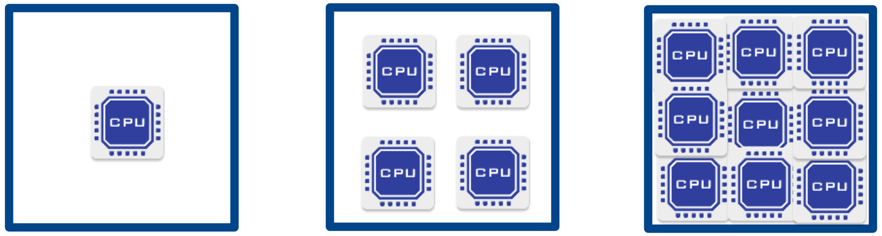
Programming models

- 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

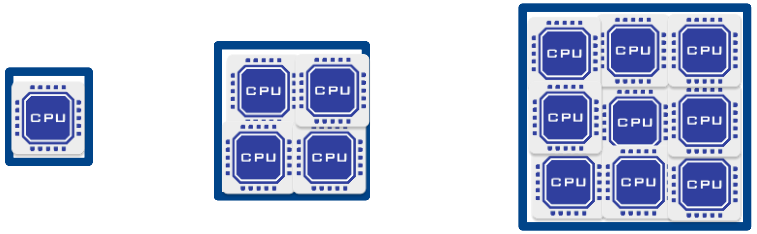
Strong scaling

- 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



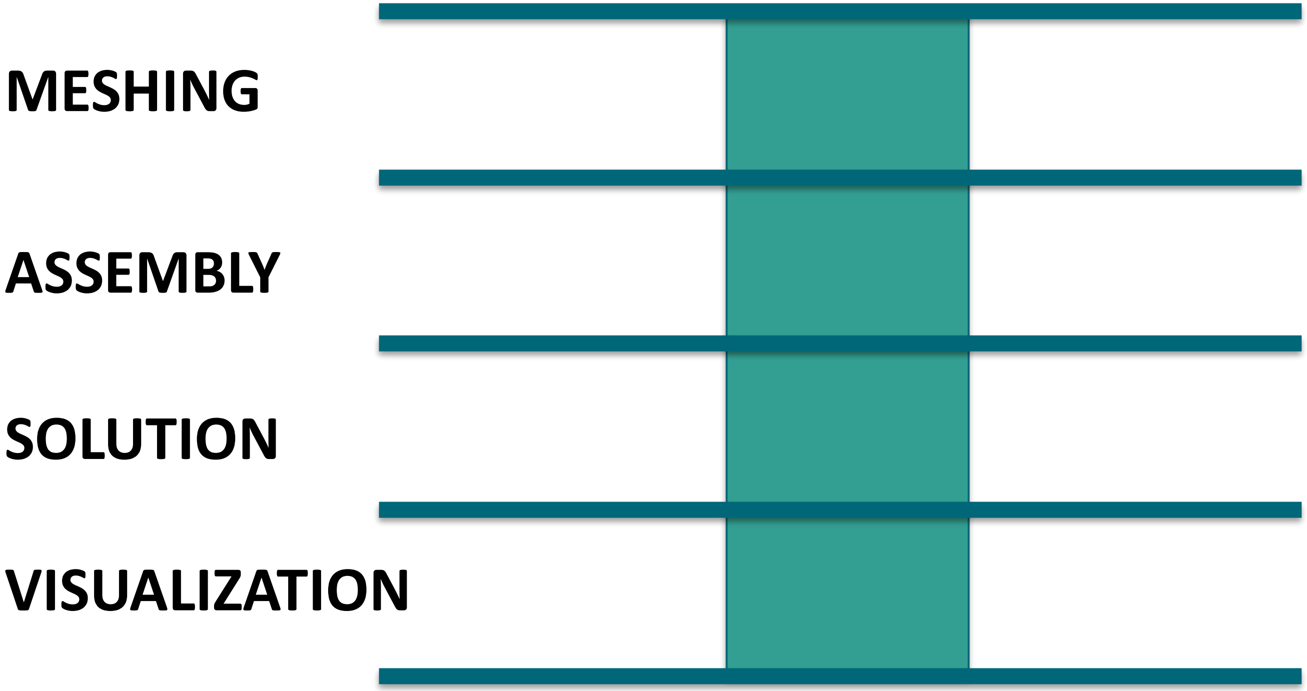
Weak scaling

- 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



MESHING

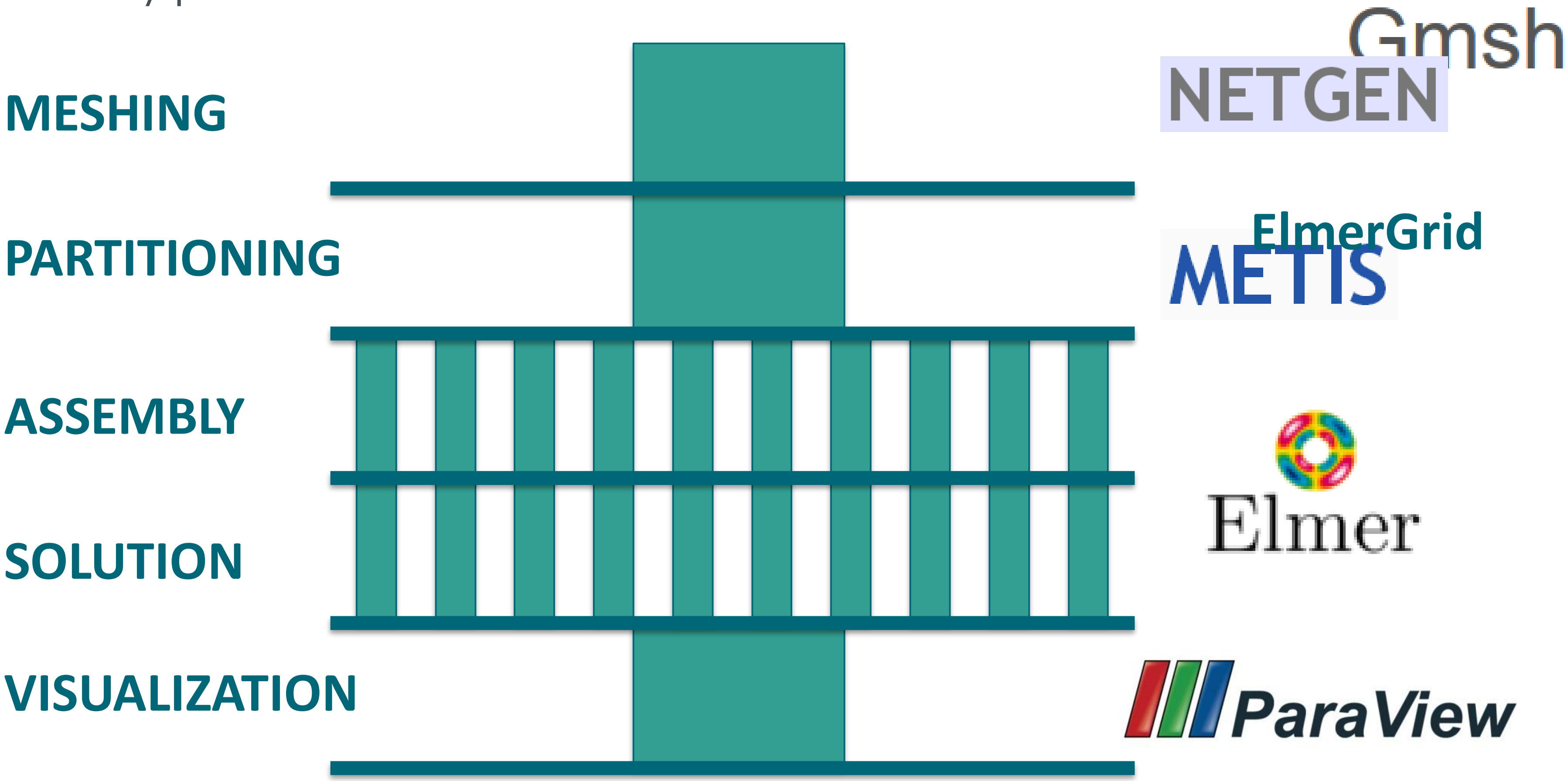
ASSEMBLY

SOLUTION

VISUALIZATION

Basic Parallel workflow of Elmer

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



ElmerGrid partitioning commands

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

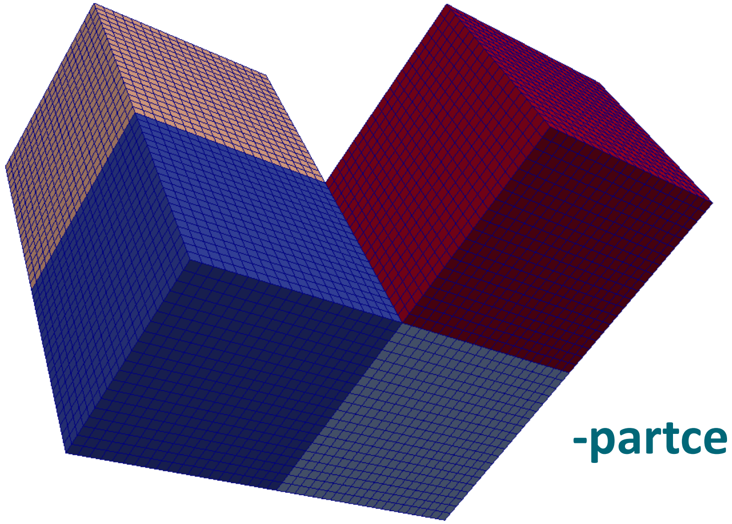
```
-partition int[3]    : the mesh will be partitioned in cartesian main directions
-partorder real[3]   : in the 'partition' method set the direction of the ordering
-partcell int[3]     : the mesh will be partitioned in cells of fixed sizes
-partcyl int[3]      : the mesh will be partitioned in cylindrical 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.

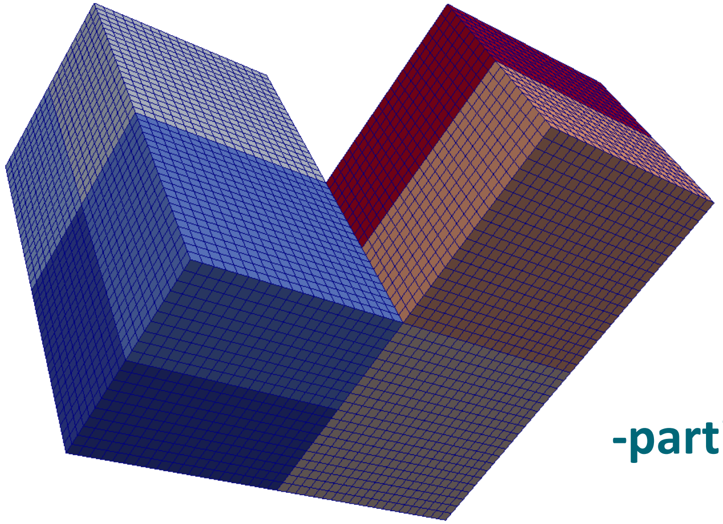
ElmerGrid partitioning examples

- 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

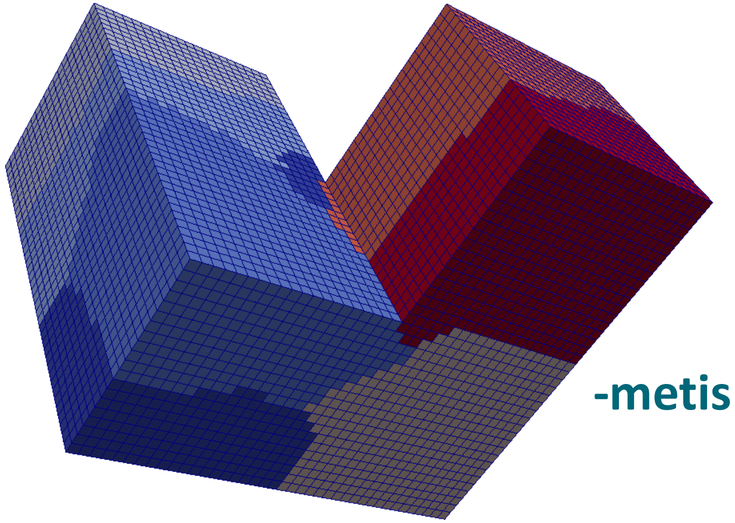
Mesh partitioning with ElmerGrid – structured mesh



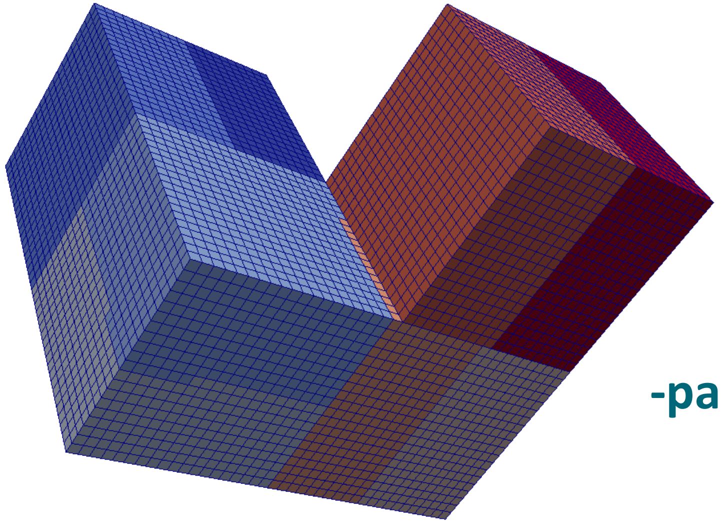
-partcell 2 2 2



-partition 2 2 2

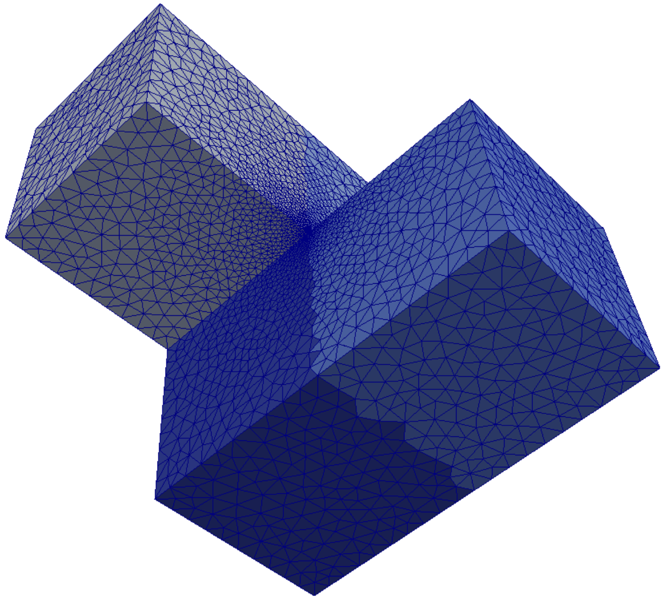


-metis 8

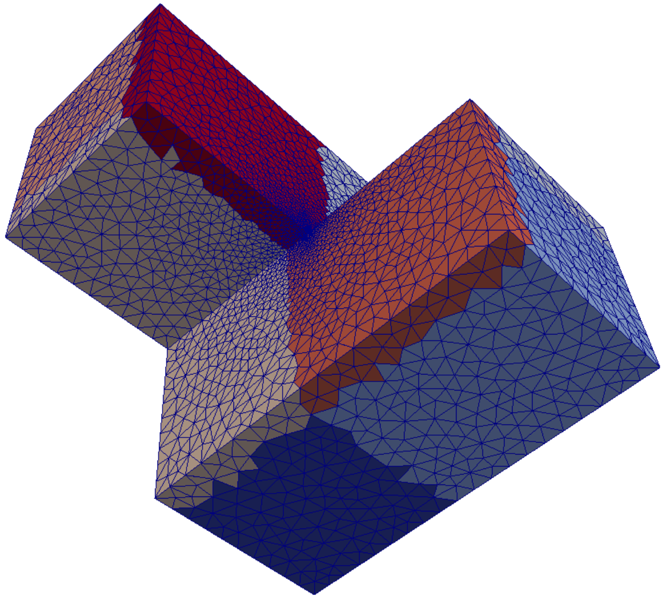


-partdual -metisrec 8

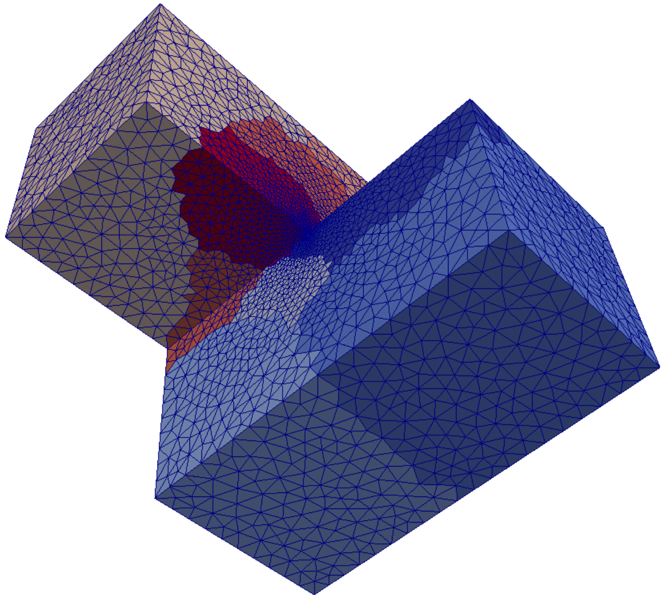
Mesh partitioning with ElmerGrid – unstructured mesh



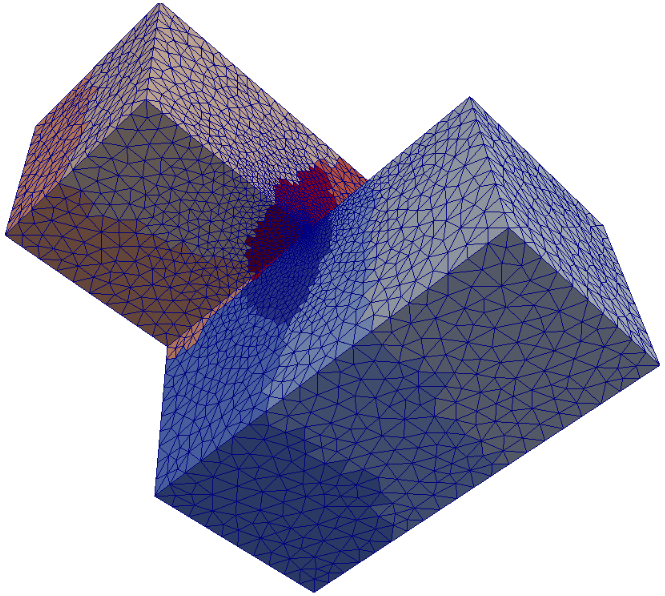
-partcell 2 2 2



-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

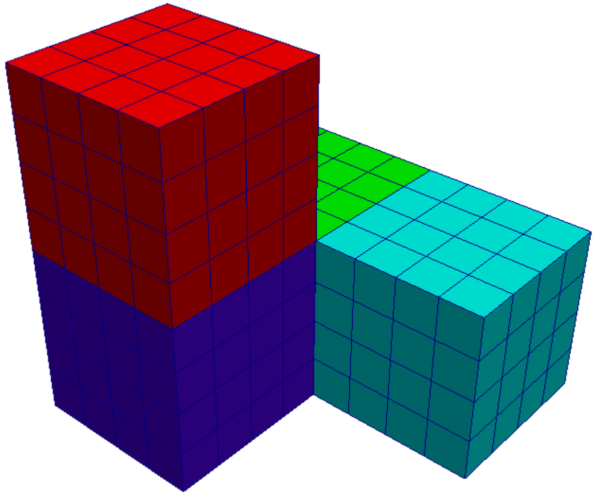
Iterative

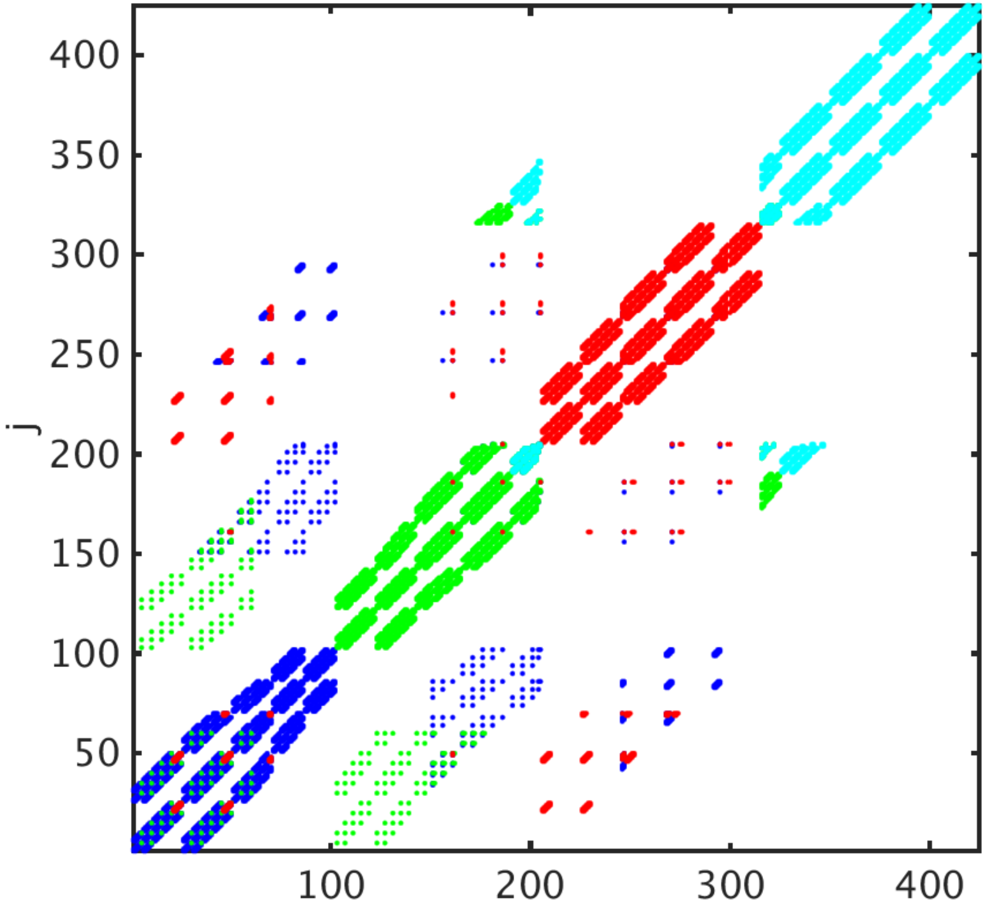
- HUTITER
  - Krylov methods initially coded at HUT
- Hypre
  - Krylov solvers
  - Algebraic multigrid: BoomerAMG
  - Truly parallel ILU and Parasails preconditioning
- Trilinos
  - Krylov solvers
  - Algebraic multigrid: ML
  - ...
- ESPRESO
  - FETI library of IT4I http://espreso.it4i.cz/

Direct

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

Partitioning and matrix structure

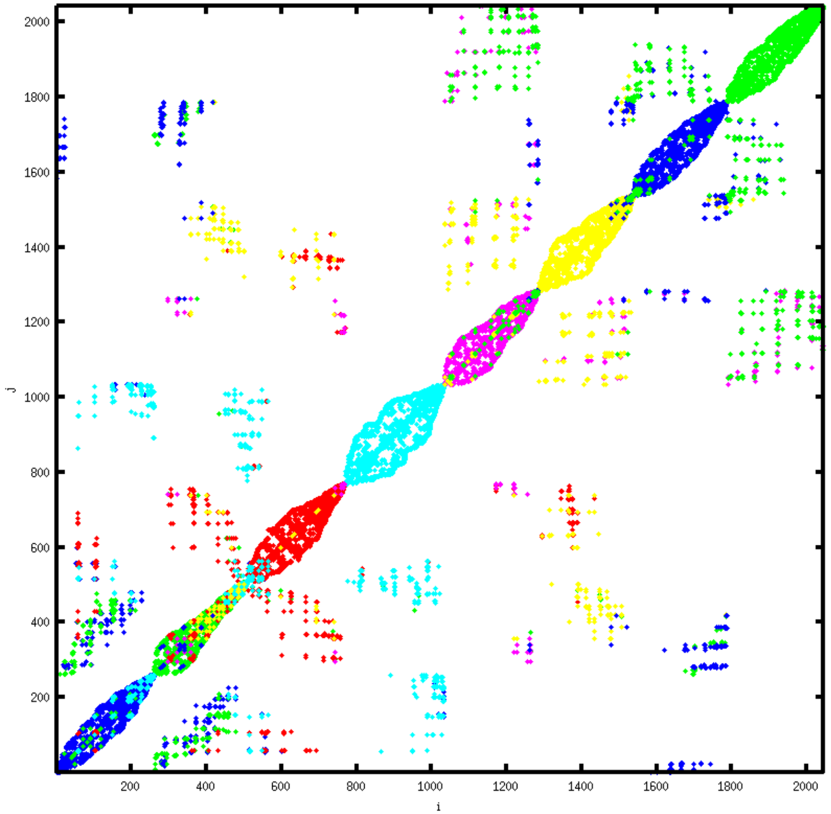


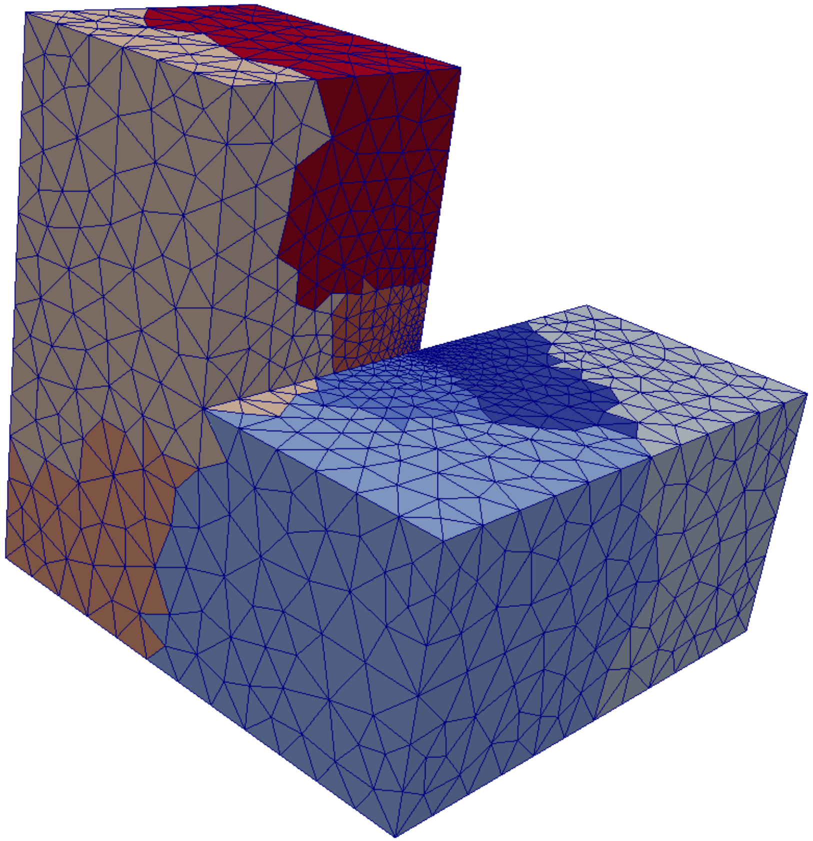


Contiguous parallel numbering used

- 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



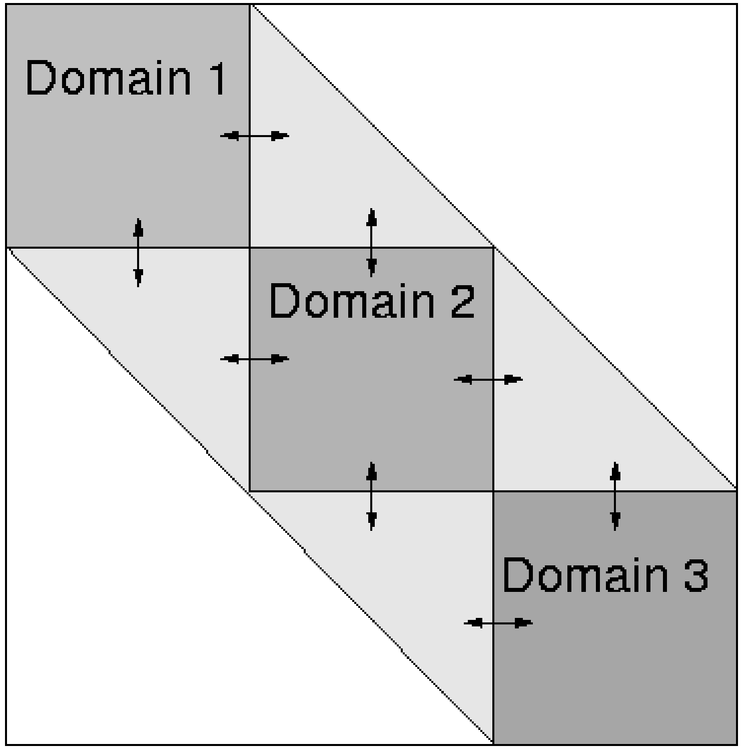


Metis partitioning into 8

- 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

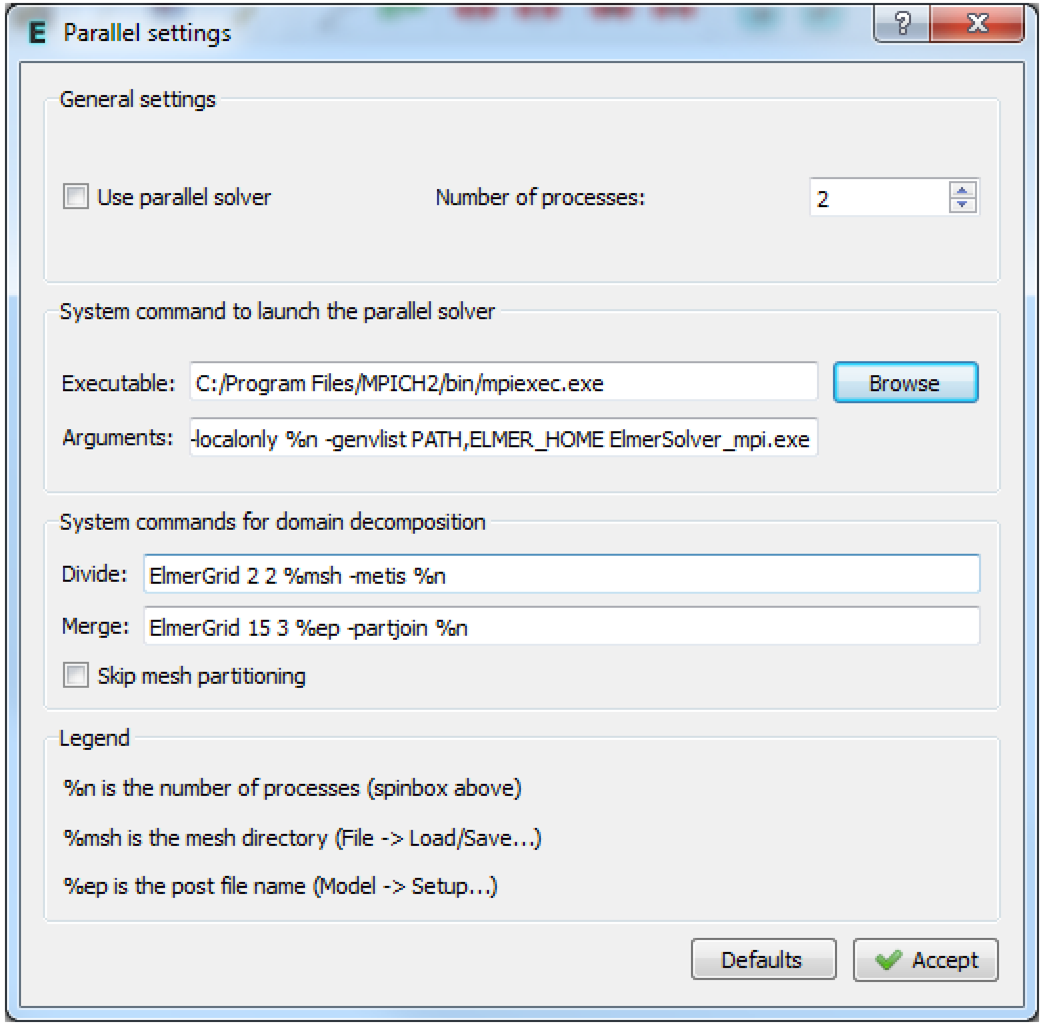
Differences in serial and parallel algorithms

- 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 settings

General settings

□ Use parallel solver

Number of processes: 2

System command to launch the parallel solver

Executable: C:/Program Files/MPICH2/bin/mpiexec.exe

Arguments: -localonly %n -genvlist PATH,ELMER_HOME ElmerSolver_mpi.exe

System commands for domain decomposition

Divide: ElmerGrid 2 2 %msh -metis %n

Merge: ElmerGrid 15 3 %ep -partjoin %n

□ Skip mesh partitioning

Legend

%n is the number of processes (spinbox above)

%msh is the mesh directory (File -> Load/Save...)

%ep is the post file name (Model -> Setup...)

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

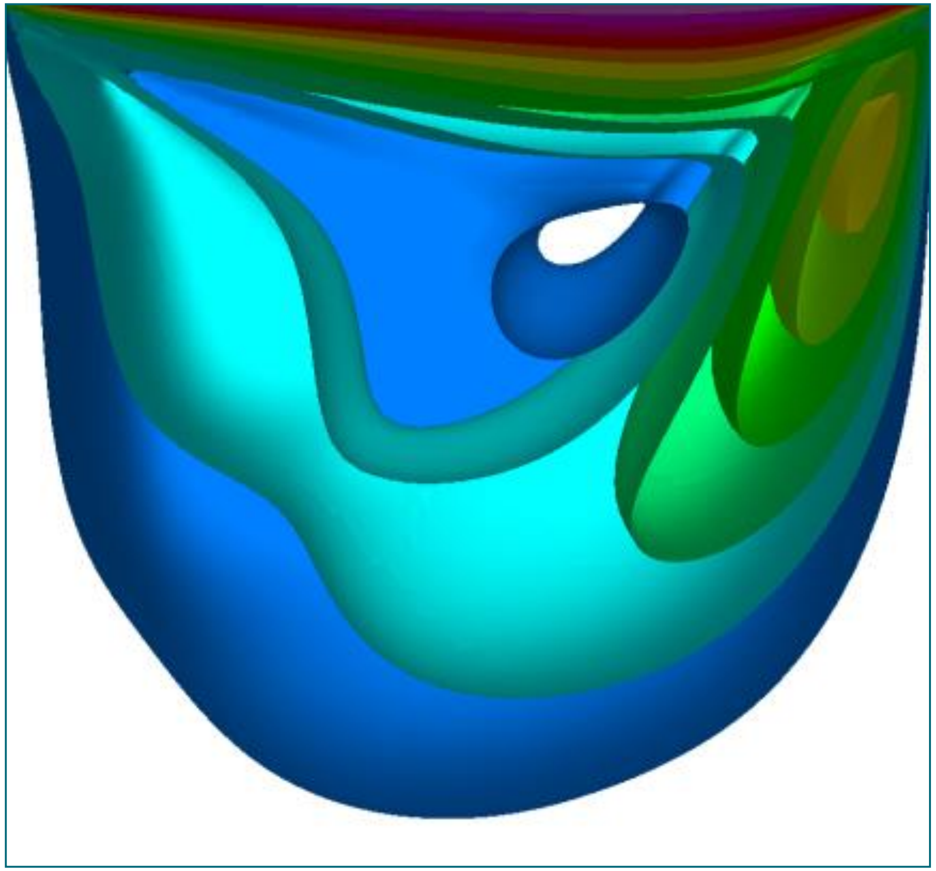
Example: Weak scaling of Elmer (FETI)

#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

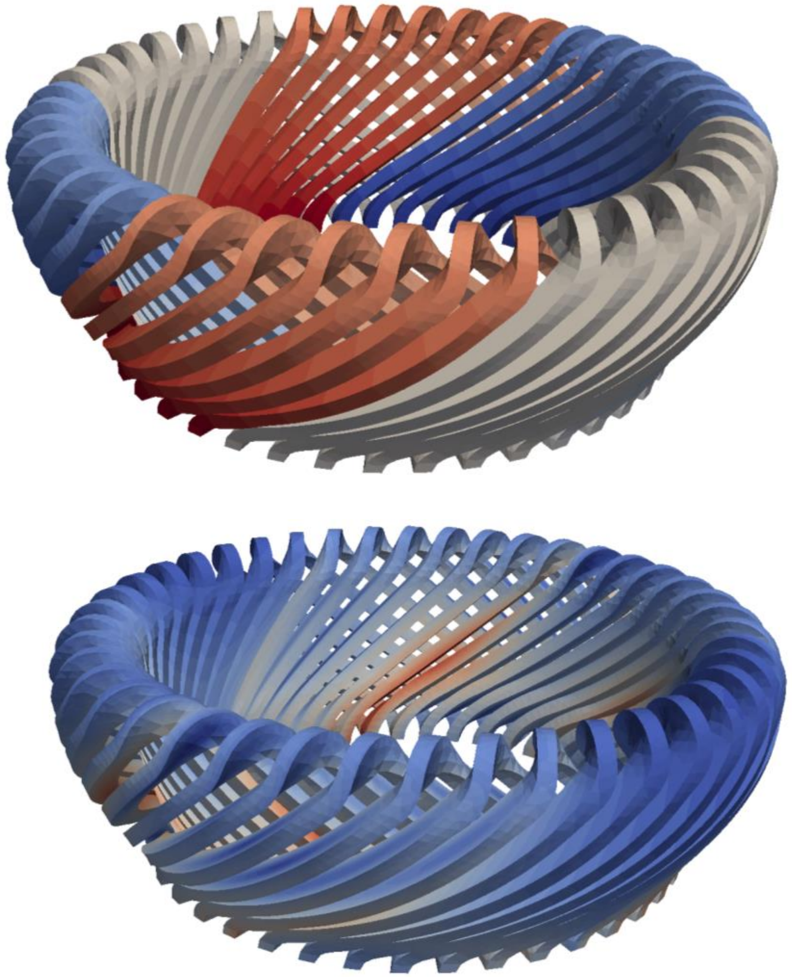
Elems	Dofs	#procs	Time (s)
34^3	171,500	16	44.2
43^3	340,736	32	60.3
54^3	665,500	64	66.7
68^3	1,314,036	128	73.6
86^3	2,634,012	256	83.5
108^3	5,180,116	512	102.0
132^3	9,410,548	1024	106.8



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

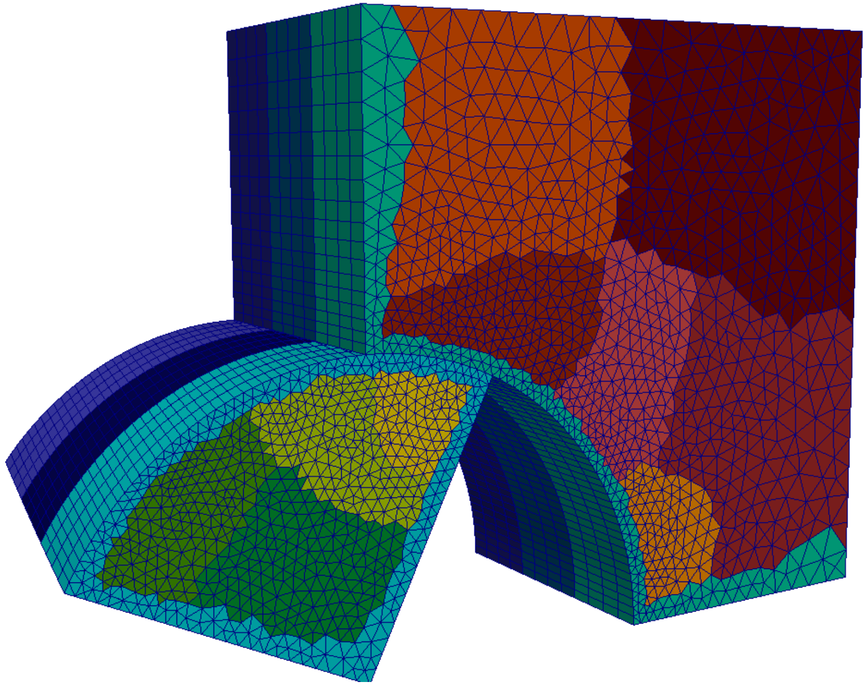
- Monolithic parallel linear system including
  - Electric scalar potential using nodal elements
  - Magnetic vector potential using edge elements (in 3D)
  - Mortar projector for the nodal dofs P_v (for conductors)
  - Mortar projector for the edge dofs P_a (in 3D)
  - Current conditions for case driven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

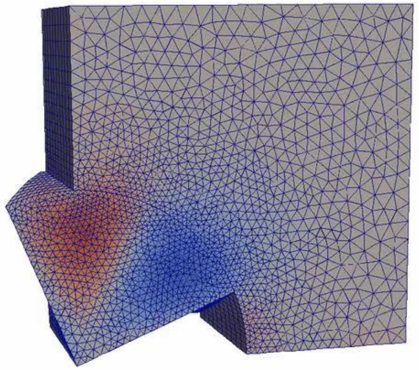
$$\begin{pmatrix} V_v & V_a & P_v^T & 0 & 0 \\ A_v & A_a & 0 & P_a^T & N \\ P_v & 0 & 0 & 0 & 0 \\ 0 & P_a & 0 & 0 & 0 \\ 0 & S & 0 & 0 & R \end{pmatrix} \begin{pmatrix} v \\ a \\ \lambda_v \\ \lambda_a \\ i \end{pmatrix} = \begin{pmatrix} f_a \\ f_v \\ 0 \\ 0 \\ V_{ext} \end{pmatrix}$$

- 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

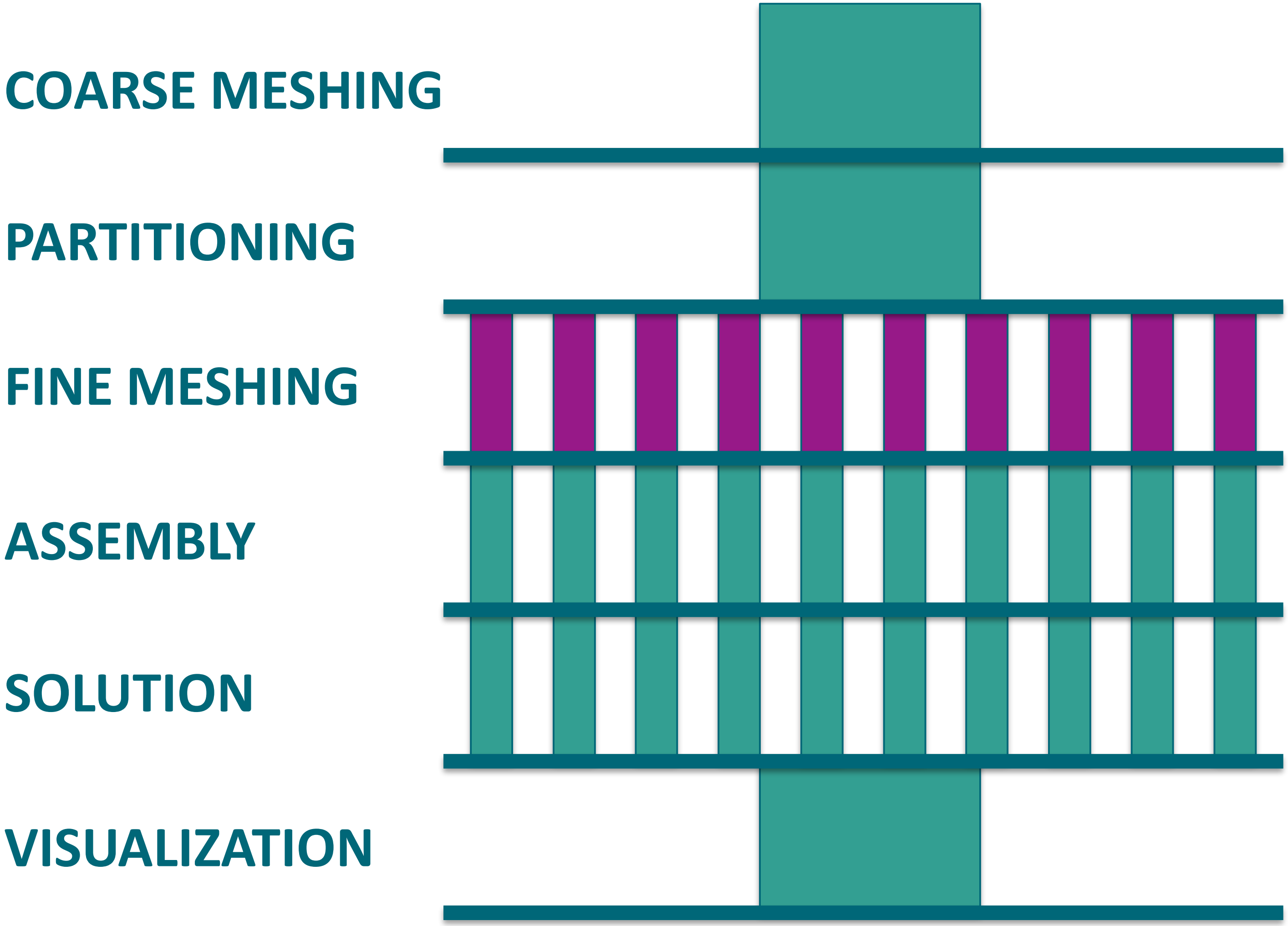
- The linear system arising from the electromagentic problem must be solved together with the continuity constraints
- To minimize communication (and coding) effort we partition the mesh cleverly
- Electrical machines have always rotating interface: Partition the interface elements so that opposing element 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



COARSE MESHING

PARTITIONING

FINE MESHING

ASSEMBLY

SOLUTION

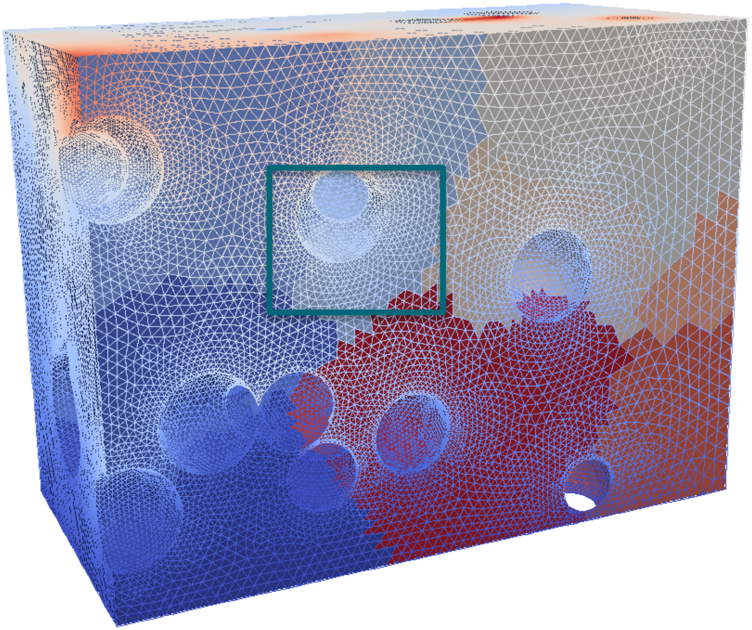
VISUALIZATION

Mesh Multiplication

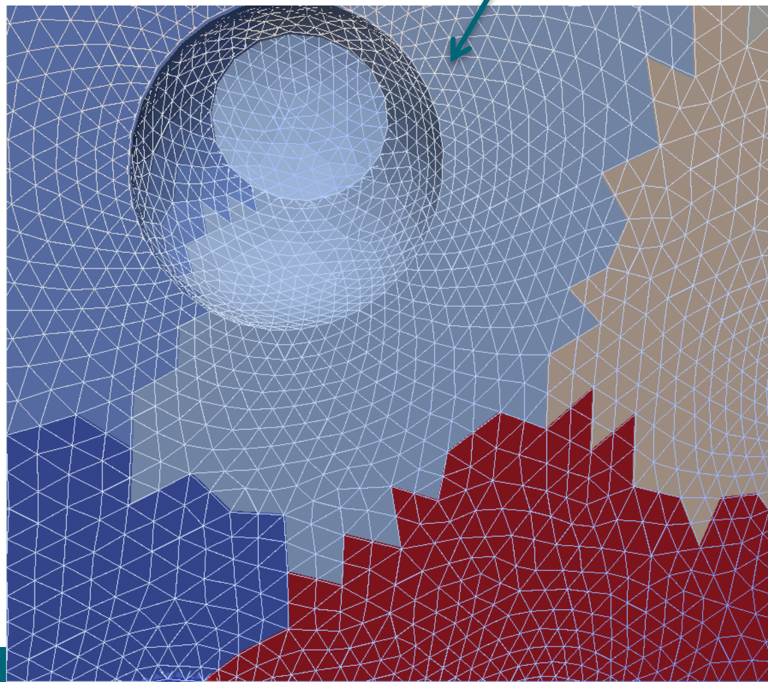
- Split elements edges after partitioning **at parallel level**
    - effectively eliminating memory and I/O bottle-necks
    - Each multiplication creates 2^DIM-fold number of elements
    - Does not increase accuracy of geometry presentation
    - May inherit mesh grading
    - CPU time used in neglible

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

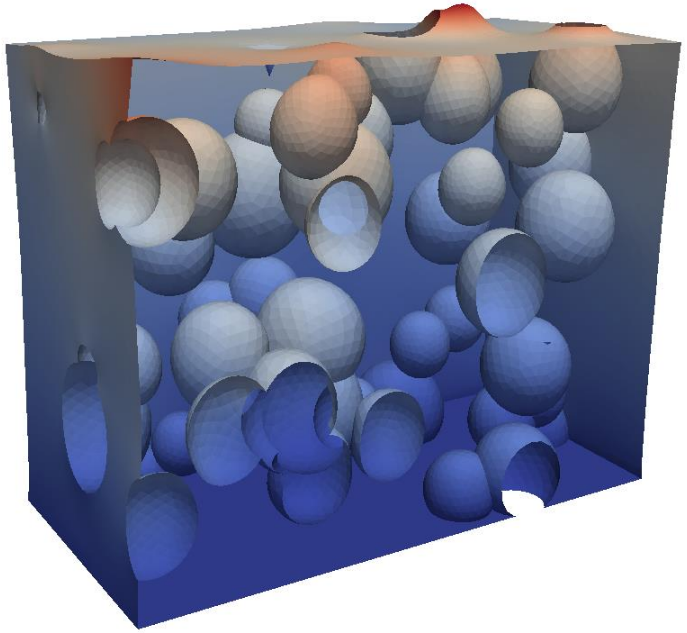


Mesh grading nicely preserved



Overcoming bottle-necks in postprocessing

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