Recent Elmer developments with potential for Elmer/Ice community

Peter Råback
ElmerTeam
CSC – IT Center for Science

Elmer/Ice Advanced Workshop
CSC, 29-13.10.2018

Recent developments

- Last official release ”8.3” released 25 April 2018
    - New official relase hopefully still this year
- 660 commits since last release
- This presentation represents some sherry picking of these features
    - Many are still not fully documented
    - Not typically made for Elmer/Ice in mind but could be of use

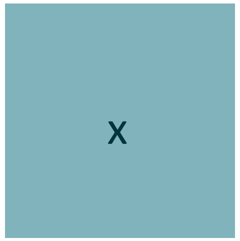
New types of fields

- Initial motivation came from code coupling between Elmer and OpenFOAM
  - The nodal data is not always the best choice
- There has been some limited support of such features before but now the treatment is more systematic, Var % Type
  - variable_on_nodes (-nodal)
  - variable_on_elements (-elem)
  - variable_on_nodes_on_elements (-dg)
  - variable_on_gauss_points (-ip)
- Since the initial use we have extended the use of these types of fields to other uses as well

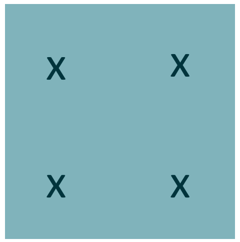
Interpolation from OpenFOAM into Elmer variables

- Variable on nodes
  - Pros: Creates a natural field that may be interpolated using FE basis, few points
  - Cons: Requires continuous fields, problems at boundaries

- Variable on elements
  - Pros: Maintains discontinuities, few points
  - Cons: Zeroth order

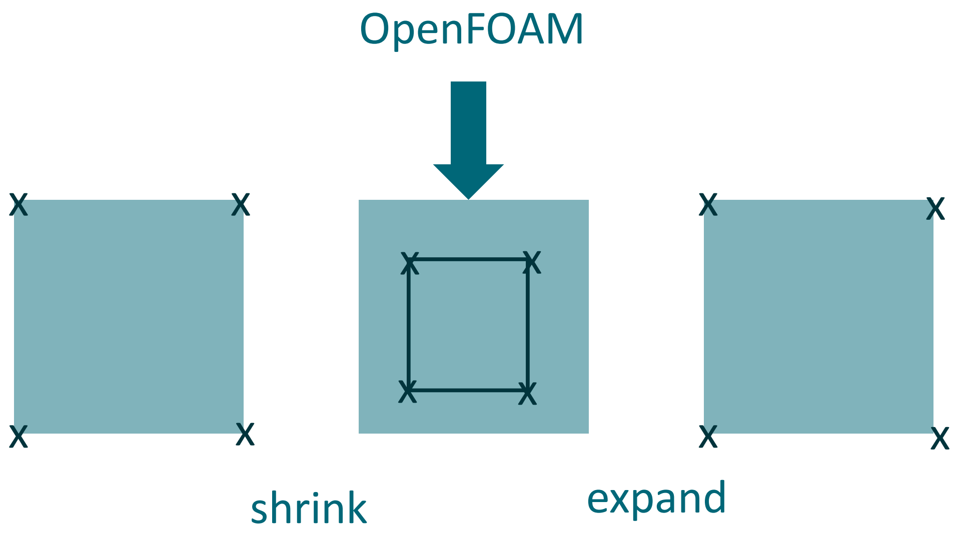


- Variable on Gauss points
  - Pros: Optimal for FE assembly, maintains discontinuities
  - Cons: Integration rules may be heavy -> many points



Interpolation from OpenFOAM into Elmer variables

- Different interplation have different merits
  - Interpolation to Gauss points is optimal but it lacks the ability to use interpolate values beyond the integration points
  - Discontinuous Galerkin type of basis would allow discontinuities and interpolation
- DG interpolation to Gauss points
  - Shrink the element such that the nodes coinside with the 2nd order Gaussian integration points (factor $1/\sqrt{3}$)
  - Interpolate fields from OpenFOAM to Elmer elements
  - Fit the nodal values so that there is agreement at the corners of the shrinked element



Testing for interpolation

Testing for

- Interpolation error convergence with mesh size
- Interpolation error with different intepolation techniques
- Parallel scalability of the interpolation

Not testing for

- Correctness of Elmer or OpenFOAM solution – these should be verified elsewhere
- Discretization error
- Interpolation error in OpenFOAM perspective

Unit cube (1x1x1) test - interpolation accuracy

OpenFOAM interpolators

- cell
- cellPoint

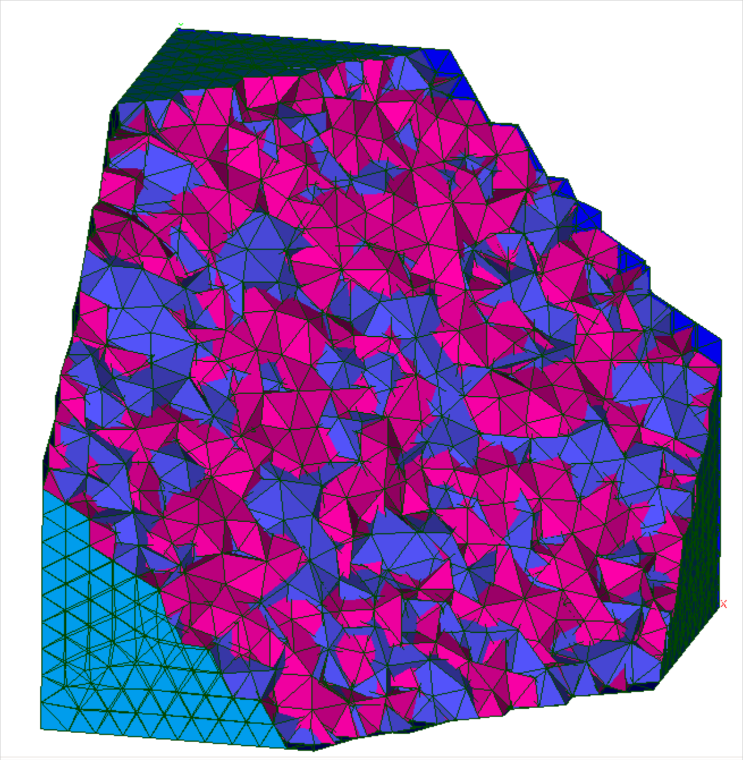
Elmer field types

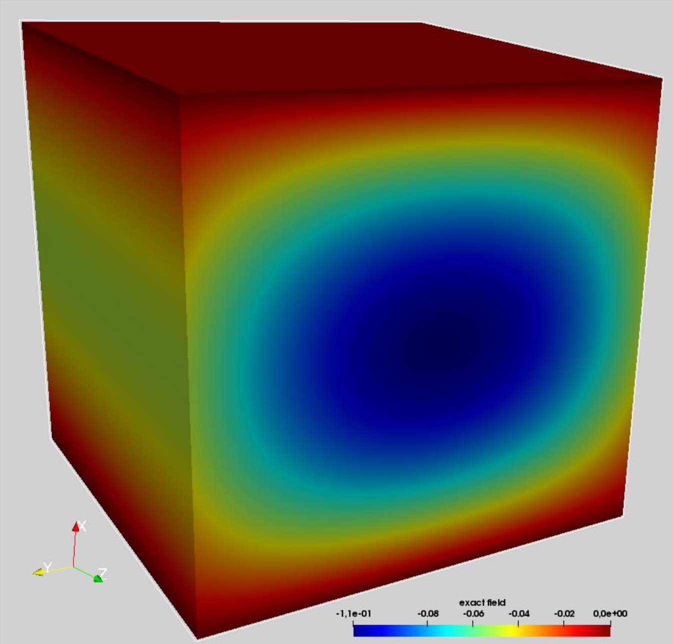
- ip (integration points)
- elem (elemental)
- dg (discontinuous Galerkin)

Mesh tet max size

- 0.1 (**4.7**k tets)
- 0.05 (**32**k tets)
- 0.025 (**245**k tets)

Mesh is rotated around x-axis to obtain “different” mesh





Initial field distribution

$$(x^2 - x) \cdot (y^2 - y - 0.2)$$

Such distribution allows testing all kinds of boundary conditions

Unit cube (1x1x1) test - accuracy test

- Data direction: **Elmer ⇒ OpenFOAM ⇒ Elmer**
- Elmer knows initial distribution & computes error
- These are **combined errors from Elmer's perspective**

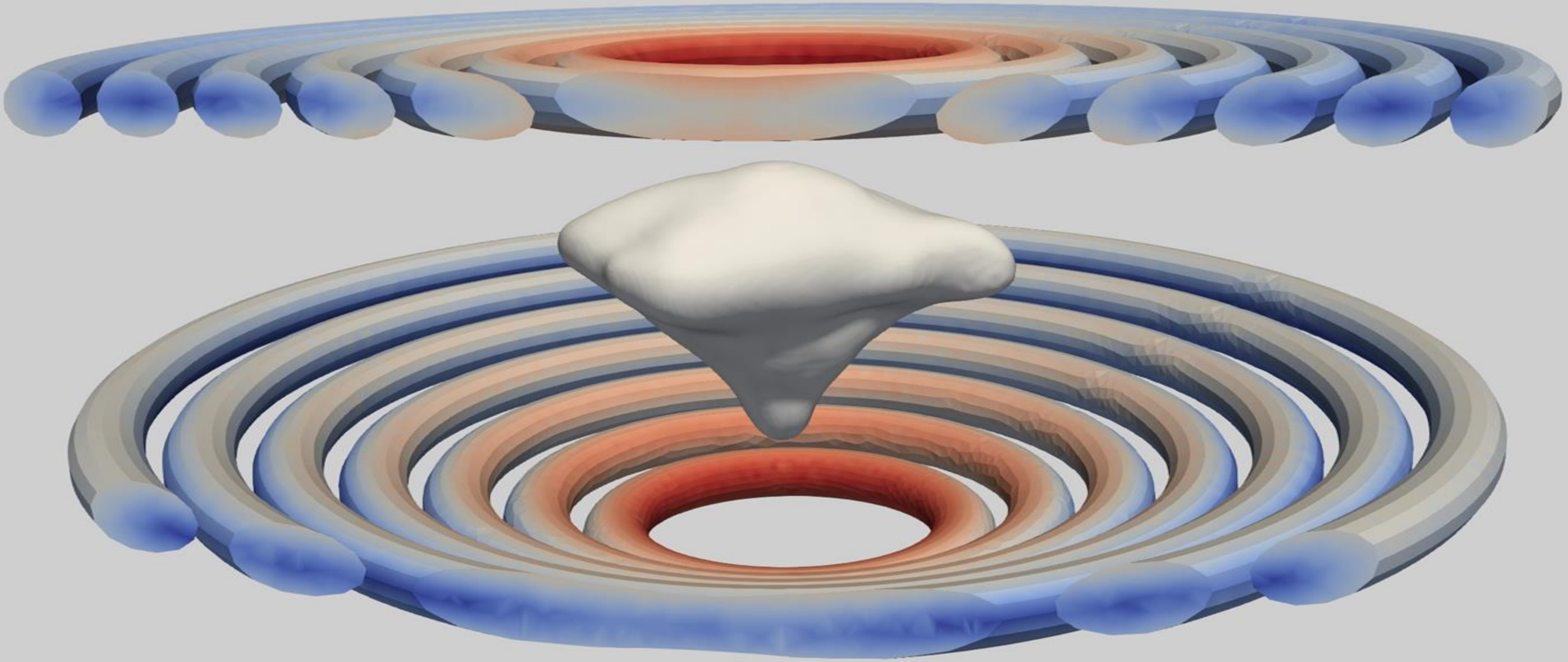
cell *(0th order)*			
tet size	**elem**	**dg**	**ip**
0.1	5.7	5.1	4.5
0.05	2.5	2.4	2.1
0.025	1	0.9	0.8

cellPoint *(1st order)*			
tet size	**elem**	**dg**	**ip**
0.1	6.4	5.4	5.3
0.05	2.2	1.8	1.8
0.025	0.7	0.5	0.5

Normalized **L1-norm** in **%**

- Error b/w OpenFOAM schemes: **cellPoint** < **cell**
- Error b/w Elmer schemes: **ip** < **dg** < **elem**

EOF-Library - Elmer & OpenFOAM coupler



Juris Vencels (University of Latvia)

EOF-Library.com

Creating different variable types in Elmer

- Any solver can allocate additional fields of different types

```
-nodal    ! variable on nodes (the default)
-elem     ! variable on elements
-ip       ! variable on integration points
-dg       ! variable on nodes on elements (Discontinuous Galerkin)
```

- For example, to allocate OpenFOAM temperature at integration points

```
Exported Variable 1 = -dg "of temperature"
```

- Dependencies of interpolated variable works in standard Elmer manner except for ip-variable which must use ListGetElementReal, e.g.

```
Electric Conductivity = Variable "of temperature"
   Real MATC "1.23/(1+3.45*tx)"
```

- Any Elmer parameter may depend on the interpolated OpenFOAM variable!

Permutation in different field types

Type	Index for variable
-nodal	j = Var % Perm(Element % NodeIndexes(i))
-dg	j = Var % Perm(Element % DgIndexes(i))
-elem	j = Var % Perm(Element % ElementIndex)
-ip	j = Var % Perm(Element % ElementIndex)+k

Where **i** is local node index and **k** is local gaussian quadrature index,

Possible uses of non-nodal fields

- You're coupling with another solver and have issues with discontinuities
- You want to save information at the IP-point level

ListGetElement –operations / motivation

ListGet –operations have some limitations

- They assume that parameters are evalued first at nodes and are then interpolated to integration points
  - `Val_atIp = SUM( Basis(1:n) * Val(1:n) )`
  - This fails when the dependence is not linear , for this reason viscosity models etc. are historically hard coded to evaluate directly at integration points
  - Changing the operation from the standard one is laborious
- Lot of redundant work
  - Assume dependences on global variables such as time. There is no built-in intelligence to take this into but same aveluation is done for each node separately.
  - Things become exceedingly costly when using MATC expressions.
  - Even fecthing constants takes time as we need to search them in the list

ListGetElement –operations / improvements

- The main shortcomings are addressed
- We may choose in which order to do interpolation/evaluation
  - Varname At Ip = Logical True ! To enforce evaluation at IP
  - If there is a dependency on IP type of variable the evaluation takes directly use of that
- There is a handle that keeps a ”cheet list” to save time
  - Is the value constant, does it depend only on global variables, where we here last time,…
  - ListFind –operations are minimized
- Savings on time depend on type of variable
  - Most savings are for constant expressions and global functional dependence
  - Dependencies on field types have similar speed as ListGetReal operations

List of function calls

SUBROUTINE ListInitElementKeyword(Handle,Section,Name,minv,maxv)

FUNCTION ListGetElementReal(Handle, Basis, Element, Found) RESULT(Rvalue)
FUNCTION ListGetElementRealVec(Handle, ngp, BasisVec, Element,Found) RESULT(Rvalues)
FUNCTION ListGetElementLogical(Handle, Element, Found) RESULT(Lvalue)
FUNCTION ListGetElementInteger(Handle, Element, Found) RESULT(Ivalue)
FUNCTION ListGetElementString(Handle, Element, Found) RESULT(CValue)

FUNCTION ListCompareElementString(Handle, CValue2, Element, Found) RESULT(Same)

+ some optional keywords when needed, eg. Gauss point index
+ LUA as replacement of MATC (ask Juhani Kataja)

ModelPDEHandles

```
TYPE(ValueHandle_t) :: Load_h, FieldSource_h, DiffCoeff_h, ReactCoeff_h, ConvCoeff_h, &
     TimeCoeff_h, ConvVelo1_h, ConvVelo2_h, ConvVelo3_h, &
     BCFlux_h, BCCoeff_h, BCExt_h

  CALL ListInitElementKeyword( Load_h,'Body Force','Field Source')
  CALL ListInitElementKeyword( DiffCoeff_h,'Material','Diffusion Coefficient')
  CALL ListInitElementKeyword( ReactCoeff_h,'Material','Reaction Coefficient')
  CALL ListInitElementKeyword( ConvCoeff_h,'Material','Convection Coefficient')
  CALL ListInitElementKeyword( TimeCoeff_h,'Material','Time Derivative Coefficient')
  CALL ListInitElementKeyword( ConvVelo1_h,'Material','Convection Velocity 1')
  CALL ListInitElementKeyword( ConvVelo2_h,'Material','Convection Velocity 2')
  CALL ListInitElementKeyword( ConvVelo3_h,'Material','Convection Velocity 3')

  CALL ListInitElementKeyword( BCFlux_h,'Boundary Condition','Field Flux')
  CALL ListInitElementKeyword( BCCoeff_h,'Boundary Condition','Robin Coefficient')
  CALL ListInitElementKeyword( BCExt_h,'Boundary Condition','External Field')
```

```
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 = ListGetElementReal( Load_h, Basis, Element, Found )
   rho = ListGetElementReal( TimeCoeff_h, Basis, Element, Found )

   a(1) = ListGetElementReal( ConvVelo1_h, Basis, Element, Found )
   a(2) = ListGetElementReal( ConvVelo2_h, Basis, Element, Found )
   IF( dim == 3 ) THEN
     a(3) = ListGetElementReal( ConvVelo3_h, Basis, Element, Found )
   END IF

   D = ListGetElementReal( DiffCoeff_h, Basis, Element, Found )
   C = ListGetElementReal( ConvCoeff_h, Basis, Element, Found )
   R = ListGetElementReal( ReactCoeff_h, Basis, Element, Found )

   Weight = IP % s(t) * DetJ

   ! diffusion term (D*grad(u),grad(v)):
   ! -----------------------------------
   STIFF(1:nd,1:nd) = STIFF(1:nd,1:nd) + Weight * &
          D * MATMUL( dBasisdx, TRANSPOSE( dBasisdx ) )

   DO p=1,nd
     DO q=1,nd
       ! advection term (C*grad(u),v)
       !
```

Speed-up for different ListGetElement -operations

```
KeywordHandleTimer: ListGetLogical/String/Integer
  Logical:                     6.33
  Integer:                     8.20
  String:                      2.93
  String comparison:           5.83

KeywordHandleTimer2: ListGetElementReal vs. ListGetReal
  Real Constant:               1.29
  Real Global MATC:            5.97
  Real Variable MATC:          1.02
  Real Variable MATC at IP:    0.85 (more accurate!)
  Real Global UDF:             1.01
  Real Variable UDF:           1.35

KeywordHandleTimer3: ListGetElementRealVec vs. ListGetReal
  RealVec Constant:            6.48
  RealVec Global MATC:         27.02
  RealVec Variable MATC:       1.02
  RealVec Variable MATC at IP:1.06 (more accurate!)
  RealVec Global UDF:          5.01
  RealVec Variable UDF:        7.45
```

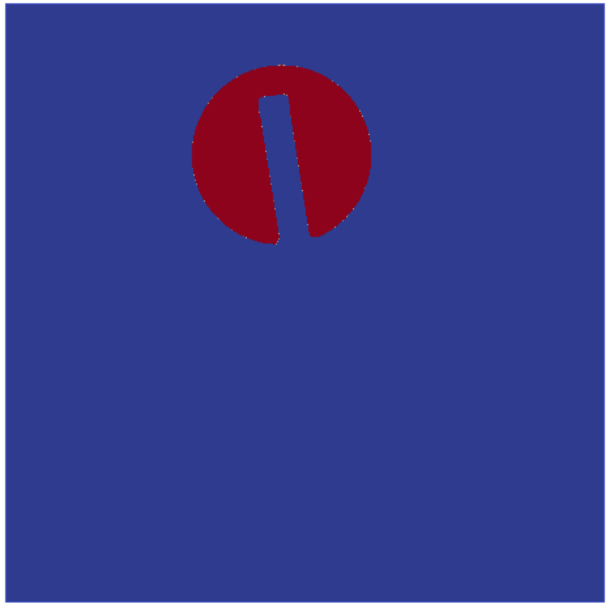
Particle –related features

- ParticleAdvector
  - Ideal method for fully convective problems
  - Follow particles backward in time and register the field value
  - Advected quantities: time & passive scalars

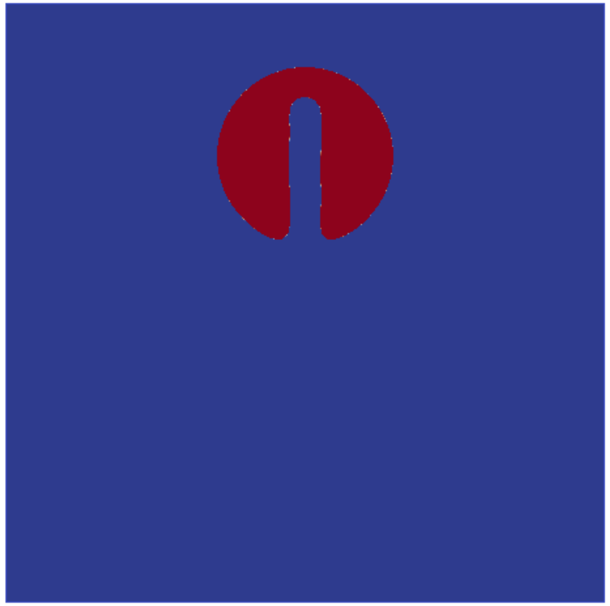
Model 32

Semi-Lagrangian advection using particle tracking

Mdule name: ParticleAdvector
Module subroutines: ParticleAdvector
Module authors: Peter Råback, Juha Ruokolainen
Module status: Alpha
Document authors: Peter Råback
Document created: 16.6.2010
Document edited: 16.6.2010

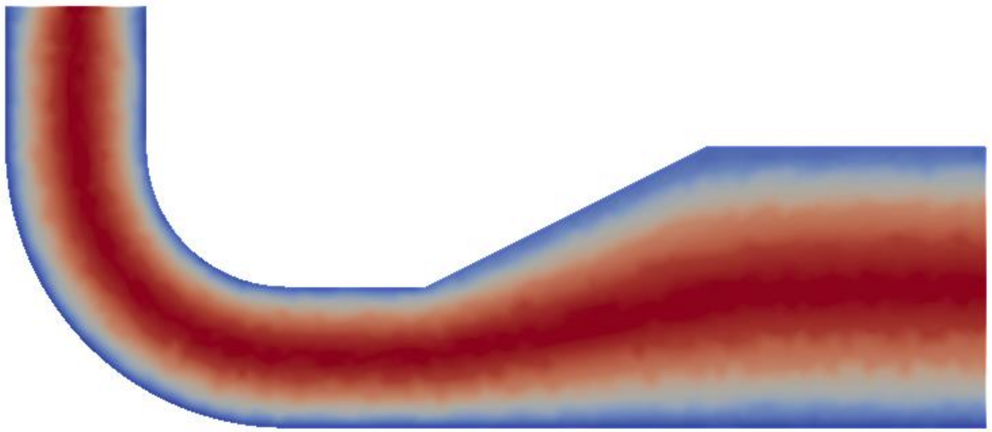


360°



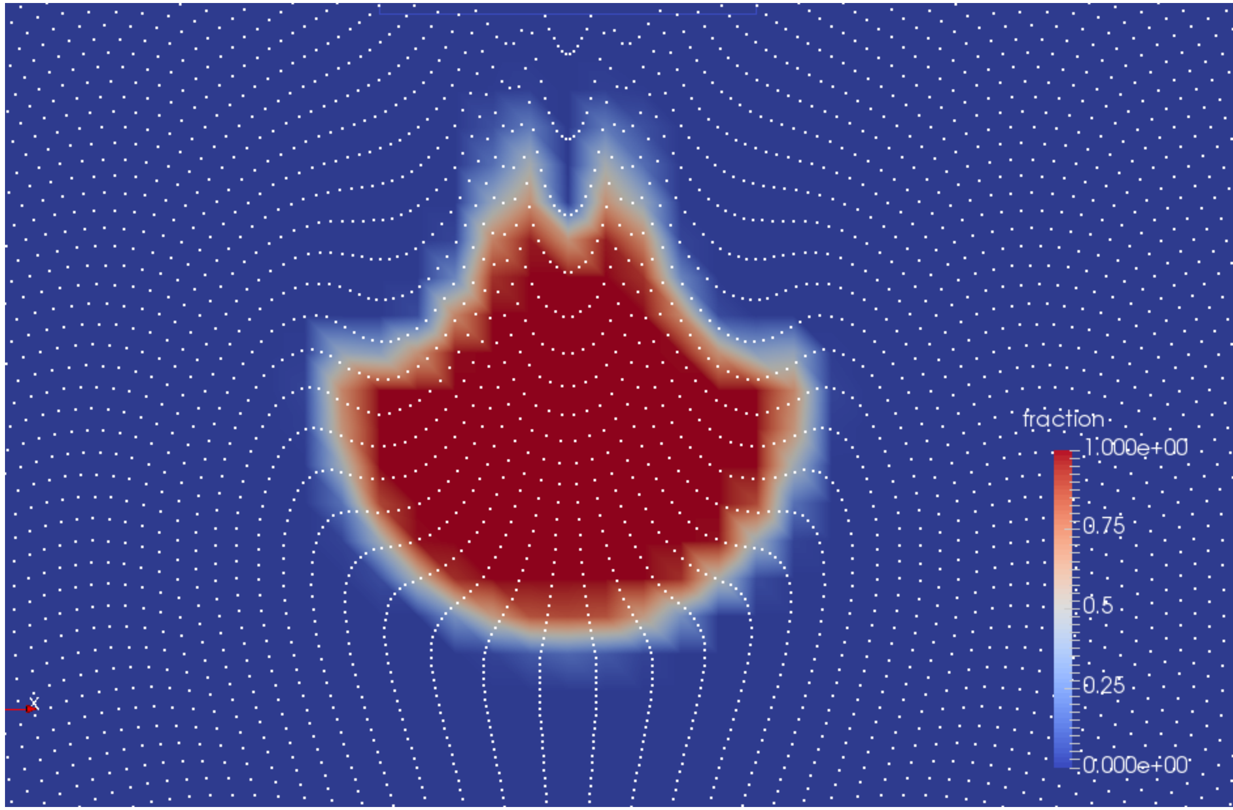
Particle –related features

- Recent development just out of oven
- ParticleAdvector
  - Some fixes for parallel operation
  - We may initialize particles not only at nodes, but also to
    - Gauss points "-ip"
    - Element centers "-elem"
    - Discontinuous Galerkin "-dg" (scaled)
  - The idea is to make the following of particles more robust since they do not immediately shoot out from the external domains at start.



Particle –related features II

- Particle Dynamics
  - Different particle sets may be sent
  - May be used to evaluete material fraction, for example
  - Features akin to ”MaterialPointMethod”
  - See test case ”ParticleFallingBlock”



Steps towards internal partitions

- Synergy with Joe's work related to adaptive meshing
- Currently only geometric division supported in master-slave strategy
  - In the future it should be modest work to add direct support for Zoltan
  - Then oftentimes partitioning with ElmerGrid could be avoided
- Current test cases
  - PartitioningDirectionalQuads
  - PartitioningUniformQuads
- Benefits
  - Elimination of a preprocessing steps
  - It will be easier to make “physics-aware” partitioning that can directly utilize command file
    - For example, minimize communication related to periodic BCs

Restart features

- Restart with different mesh
  - Possibly also with different number of partitions
  - Mesh2MeshSolver - wrapper to GetVariable
  - Test cases: NonconformingRestart*
- Higher order restart
  - Saves also the PrevValues thereby allowing accurate restart also for higher order schemes

Linear solver strategies

- Fallback strategy
  - Use of namespaces
  - Linsys1: , Linsys2: etc.
  - Test cases: LinearSolverNamespaces

Dirichlet BCs

- Way in which Dirichlet conditions are set has changed
- The routines create ConstrainedDOFs and DValues tables
- These are communicated in parallel
  - No need for Orphan nodes in partitioning routines
- More robust operation, simpler scaling etc.

Reduced integration rules for p-bubbles

- Relax p-bubble integration rules (don't overintegrate).
  - F.ex. should speed up assembly of p-bubble stabilized elements, especially with bricks.
- Changes the desired value of "Relative Integration Order"

ElmerGrid

- Preliminary version of Gmsh version 4 import
  - Version number automatically detected
- Use of more recent Metis library
  - not the changed calling convention)

Further information

- http://www.csc.fi/elmer
  - Official Homepage of Elmer
- http://www.elmerfem.org
  - Discussion forum, wiki, elmerice community
- https://github.com/elmercsc/elmerfem
  - GIT version control
- http://eof-library.com/
  - Elmer-OpenFOAM libary by Juris Vencels
- Email: peter.raback@csc.fi

Thank you for your attention!