CSC

EGU 2013 – Elmer/Ice Splinter Meeting

New Developments for Elmer/Ice

Thomas Zwinger, Mika Malinen, Juha Ruokolainen and **Peter Råback**

CSC – IT Center for Science Ltd.

Espoo,

Finland

Contributions by

Gaël Durand, Fabien Gillet-Chaulet and **Lionel Favier**

LGGE, UJF-Grenoble, France

Jonas Thies

Univ. Uppsala, Sweden

Contents

- New `elmerice` installation package
- New features for glaciological simulations:
  - New implementation of Glen's flow law
  - Internal extrusion
  - Block pre-conditioner
  - Mass-conserving normals
- Preliminary results AAIS

Elmer and Elmer/Ice

- Elmer ~300 000 lines of mixed F90, C and C^{++} code
- Elmer/Ice ~20 000 lines add-on to Elmer
- Main developments in algorithms, parallel performance enhancement driven by Elmer
  - Most work is done within CSC
  - Current developments:
    - OpenMP multi-threading, hybrid MPI-OpenMP
    - Intel PSI porting (many-core systems)
    - Sliding mesh boundaries

Elmer/Ice installation package

- SourceForge (SF): http://sourceforge.net/projects/elmerfem/
- New SVN address:
  - Checkout without SF-ID:

```
svn co svn://svn.code.sf.net/p/elmerfem/code/trunk/
```

  - Checkout with SF-ID (needs password):

```
svn checkout --username=sflogin
  svn+ssh://sflogin@svn.code.sf.net/p/elmerfem/code
  /trunk
```

  - Elmer/Ice is residing in a sub-directory: `trunk/elmerice`

Elmer/Ice installation package

- Prerequisites:
  - existing Elmer installation
  - UNIX/Linux system
  - (GNU)-make
- Either define `ELMERICE_HOME` as the installation path
- Preferably: have `ELMER_HOME` defined and Elmer/Ice then is installed in `$(ELMER_HOME)/share/elmersolver`
  - Mind that you have to have rights to write the `$ELMER_HOME`-tree

Elmer/Ice installation package

- Remove leftovers from previous builds:

  `make purge`

- Compile: `make compile`

- Install: `make install`
  - If you need to use `sudo` option, use `-E` to copy the environment.

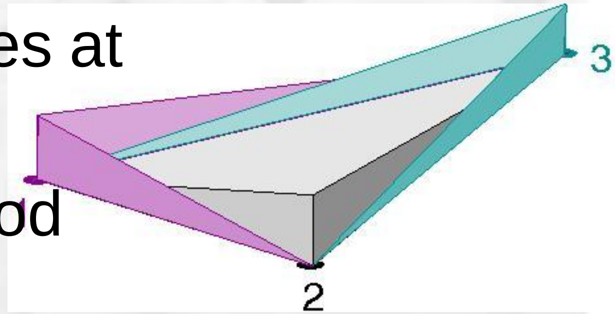
Elmer/Ice installation package

- Installation of two additional shared objects:
  - `ElmerIceSolvers.so` : contains all solver subroutines (physical models)
  - `ElmerIceUSF.so`: contains all user functions (boundary conditions, etc.)
- Call syntax:
  - `Procedure = File "ElmerIceSolvers" "`*`NameSolver`*`"`
  - Description of all Solvers on Wiki page http://elmerice.elmerfem.org/wiki/doku.php?id=solvers

Glen's flow law

$$\eta = \frac{1}{2}(EA)^{-1/n}\dot{\varepsilon}_e^{(1-n)/n}.$$

- Until recently:
  - used the Elmer built-in **`power law`** and provided the temperature-dependent part at the nodes only (MATC function)
- New Viscosity law in Elmer:
  - Viscosity model **`Glen`** in Material section
  - Evaluates all variable dependencies at integration points
  - Increased stability – Newton method works
  - Documentation in Elmer/Ice Wiki



Glen's flow law

$$\eta = \frac{1}{2}(EA)^{-1/n}\dot{\varepsilon}_e^{(1-n)/n}.$$

```
Viscosity Model = String "Glen"
!Viscosity has to be set to a dummy value
! Use "sane" value for ParStokes
Viscosity = Real
  $1.0E13*365.25*24*3600*1.0E-06
Glen Exponent = Real 3.0
Critical Shear Rate = Real 1.0e-10
 ! Rate factors
Rate Factor 1 = Real 1.258e13
Rate Factor 2 = Real 6.046e28
Activation Energy 1 = Real 60e3
Activation Energy 2 = Real 139e3
Glen Enhancement Factor = Real 1.0
```

$$A(T') = A_0 \exp\left(-Q/R(T_0 - T')\right)$$

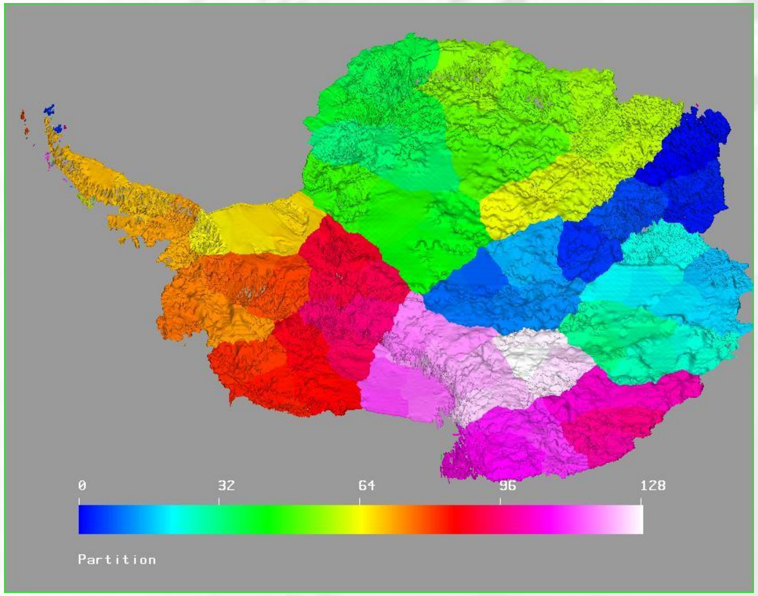
Glen's flow law

$$A(T) = A_0 \exp\left(-Q/R(T_(0) - T')\right)$$

```
! the temperature to switch between the
! two regimes in the flow law
Limit Temperature = Real -10.0

Temperature Field Variable = String "Temp
  Homologous"

! In case there is no temperature variable
  !Constant Temperature = Real -10.0
```

Internal mesh extrusion

- Until recently:
  - Build 2D footprint (optimize footprint)
  - Extrude externally (e.g. ExtrudeMesh)
  - Split resulting 3D mesh into partitions
  - Disadvantages:
    - 3D bottleneck
    - limited in size
    - Not able to utilize vertical columns

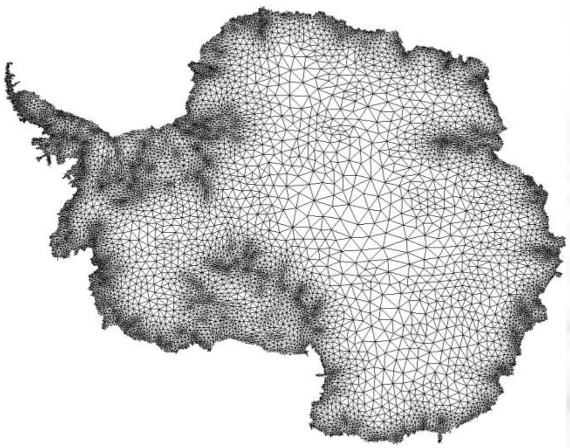


Internal mesh extrusion

- New approach:
  – Create footprint (like earlier)
  – Partition footprint
  – The rest is done inside Elmer
- Internal extrusion:
  – Keyword in **`Simulation`**:

    `Extruded Mesh Levels=10`

  – This extrudes the footprint (here in 10 levels) to unit-height
  – Still need to prescribe the bedrock and surface topography



Internal mesh extrusion

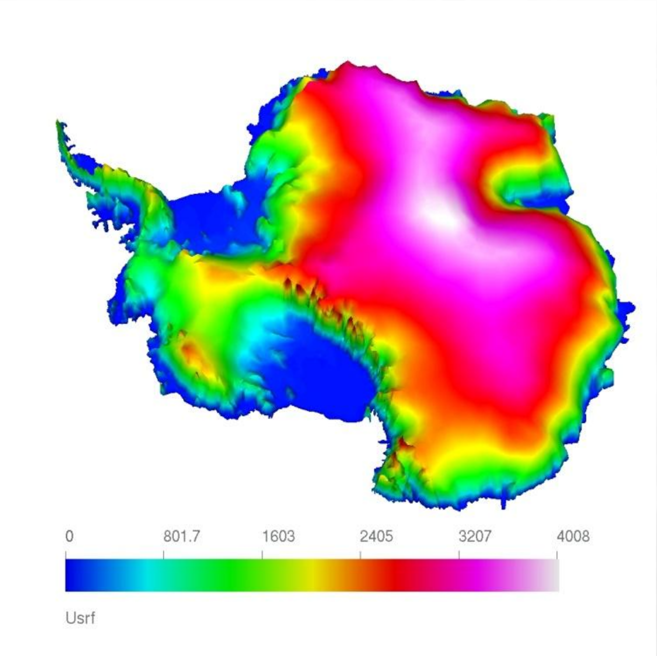
- Reading NetCDF information:
  - **GridDataReader** Under elmerice/netcdf2 (earlier under misc-tree)
  - Naturally, needs working NetCDF installation

```
Solver 1
  Equation = “DataReader"
  Exec Solver = "Before All"
  Procedure = "GridDataReader" "GridDataReader"
  Filename = File "netcdf/ALBMAPv1.nc"
  X Name = String "x1"
  Y Name = String "y1"
  !--- Interpolation variable tolerances
  X Epsilon = Real 1.0e-2
  Y Epsilon = Real 1.0e-2
  Epsilon Time = Real 0.01
  !---- offsets and stretching
  Interpolation Bias = Real 0.0
  Interpolation Multiplier = Real 1.0
```

Internal mesh extrusion

- Reading NetCDF information:

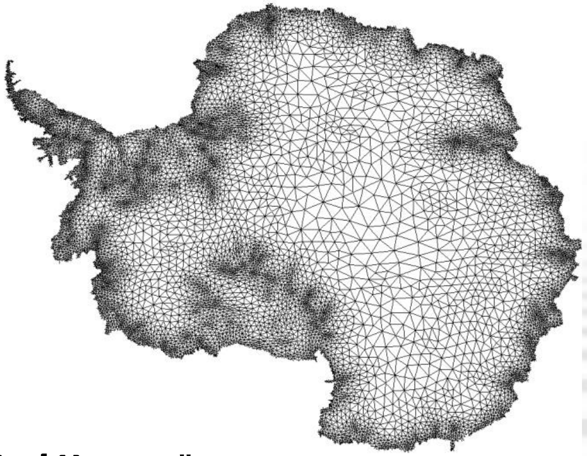
```
  Is Time Counter = Logical True

  Variable 1 = usrf  ! upper surface
  Variable 2 = lsrf2 ! lower surface
  Valid Min Value 1 = Real  0.0
  Valid Min Value 2 = Real -3000.0

  ! Scales the Elmer grid to match the
  ! NetCDF grid – usually not a good idea
  Enable Scaling = Logical False
End
```



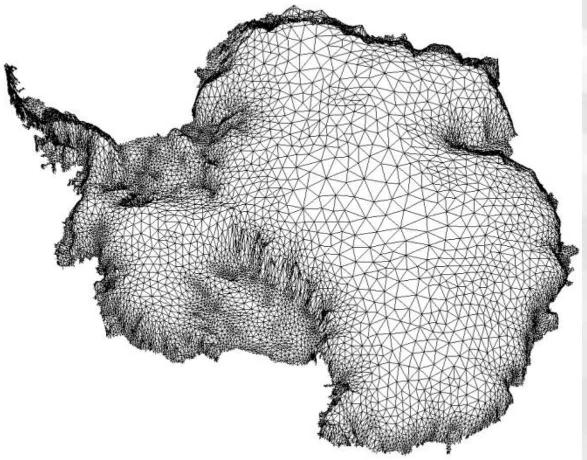
Internal mesh extrusion

- Mapping the surfaces:

  StructuredMeshMapper

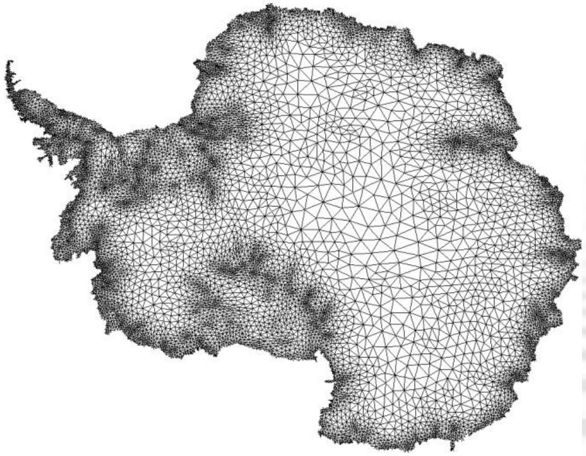


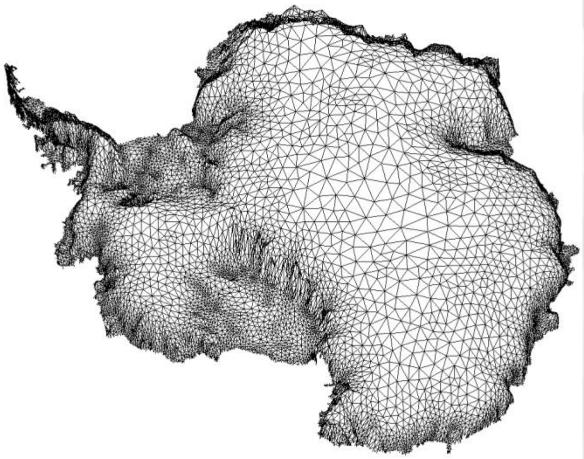
```
Solver 5
  Exec Solver = "Before Simulation"
  Equation = "MapCoordinate"
  Procedure = "StructuredMeshMapper" "StructuredMeshMapper"
  Active Coordinate = Integer 3
  Dot Product Tolerance = Real 0.0001
  Minimum Mesh Height = Real 100.0
End

Boundary Condition 1
  Name = "Bottom"
  Bottom Surface = Equals lsrf2
End

Boundary Condition 3
  Name = "Surface"
  Top Surface =  Equals usrf
End
```



Internal mesh extrusion

- Mapping the surfaces:
  StructuredMeshMapper
- Can be used also in prognostic runs:
  – Uses free surface variable for mapping the upper surface
  – Also possible to be used for isostacy at bedrock
  – Only vertical shifting of mesh, no solution of pseudo-elastic problem; **stability!**





Block pre-conditioner

- Stokes equation:
  - Saddle-point problem: needs stabilization
  - Strong spatial variation/low-shear rate singularity of viscosity: bad condition number
- Until recently: direct solution
  - MUMPS
  - Strong limits due to memory
  - Not good scalability above ~100 processes
  - Need Krylov subspace solver to work

Block pre-conditioner

- Stokes equation:

$$\boldsymbol{K} \cdot \boldsymbol{x} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^{(\mathrm{T})} \\ \boldsymbol{B} & \boldsymbol{C} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{pmatrix} = \boldsymbol{f}$$

  – $\boldsymbol{A}$ is similar to an elasticity problem (Navier-equation)
  – $\boldsymbol{B}$ is the discretized negative divergence
  – $\boldsymbol{C}$ results from stabilization
- Strategy: use pre-conditioner and solve with Krylov-subspace method (in our case GCR)

Block pre-conditioner

- GCR: $$\boldsymbol{x} = \boldsymbol{x}_0 + \sum_{i=1}^{k} \alpha_i \, \boldsymbol{s}_i$$
  - Builds solution space from initial solution $\boldsymbol{x}_0$ and a series of directional updates $\boldsymbol{s}_i$
  - Minimizes the residual $\|\boldsymbol{r}\| = \|\boldsymbol{K} \cdot \boldsymbol{x} - \boldsymbol{f}\| \rightarrow \min$
- Block pre-conditioner:
  $$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^{(\mathrm{T})} \\ \boldsymbol{0} & \boldsymbol{M} \end{pmatrix}$$
  - We use P instead of K to get directions: $\boldsymbol{P} \cdot \boldsymbol{s}_{i+1} = \boldsymbol{r}_i$
  - $\boldsymbol{M}$ is a mass-matrix scaled with the element-wise fluidity (instead of exact pressure-Schur complement)

Block pre-conditioner

- Block pre-conditioner:
  - Remaining issue: Need approximated inverse of $\boldsymbol{P}$ to get a solution of $\boldsymbol{P} \cdot \boldsymbol{s}_{i+1} = \boldsymbol{r}_i$

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^{(\mathrm{T})} \\ \boldsymbol{0} & \boldsymbol{M} \end{pmatrix}$$

  - Advantage: can get separate solution for $\boldsymbol{A}$-block and $\boldsymbol{M}$
  - Elmer provides interfaces to different libraries (Hypre, Trilinos) to get this solved

Block pre-conditioner

- How to use it?
  - Source code is in `trunk/fem/src/modules/ParStokes.src`
  - Copy the code to your directory (possibly rename the suffix to `.f90`, as some Fortran compilers are picky about this) and then simply compile it:

```
elmerf90 ParStokes.f90 -o
ParStokes.so
```

  - Create a dummy routine (see next slide) for the blocks and compile it:

```
elmerf90 DummySolver.f90 -o
DummySolver.so
```

Block pre-conditioner

```
SUBROUTINE DummyRoutine( Model,Solver,dt,TransientSimulation )
 USE DefUtils
 USE SolverUtils
 USE ElementUtils
 IMPLICIT NONE
 TYPE(Solver_t) :: Solver
 TYPE(Model_t) :: Model
 REAL(KIND=dp) :: dt
 LOGICAL :: TransientSimulation

 PRINT *,”Setting up block matrix”

END SUBROUTINE DummyRoutine
```

Block pre-conditioner

$$P = \begin{pmatrix} A & B^{(T)} \\ 0 & M \end{pmatrix}$$

```
Solver 1
  Equation = "Velocity Preconditioning"
  Procedure = "DummyRoutine" "DummyRoutine"
  Variable = -dofs 3 "V"
  Variable Output = False
  Exec Solver = "before simulation"
  Element = "p:1 b:4"
  Bubbles in Global System = False
  Linear System Symmetric = True
  Linear System Scaling = True
  Linear System Row Equilibration = Logical False
  Linear System Solver = Iterative
  Linear System Iterative Method = BiCGStab
  Linear System Max Iterations = 250
  Linear System Preconditioning = ILU0
  Linear System Convergence Tolerance = 1.0e-6
  Linear System Abort Not Converged = False
  Skip Compute Nonlinear Change = Logical True
  Back Rotate N-T Solution = Logical False
  Linear System Timing = True
 End
```

- Dummy solver, just to allocate the matrix block
- Defines the solution parameters that are taken over by ParStokes

Block pre-conditioner

$$P = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^{(\mathrm{T})} \\ \boldsymbol{0} & \boldsymbol{M} \end{pmatrix}$$

```
Solver 2
  Equation = "Pressure Preconditioning"
  Procedure = "DummyRoutine" "DummyRoutine"
  Variable = -dofs 1 "P"
  Variable Output = False
  Exec Solver = "before simulation"
  Element = "p:1 b:4"
  Bubbles in Global System = False
  Linear System Symmetric = True
  Linear System Scaling = True
  Linear System Solver = iterative
  Linear System Iterative Method = CG
  Linear System Max Iterations = 1000
  Linear System Convergence Tolerance = 1.0e-6
  Linear System Preconditioning = Diagonal
  Linear System Residual Output = 10
  Skip Compute Nonlinear Change = Logical True
  Back Rotate N-T Solution = Logical False Linear
  System Timing = True
End
```

- Dummy solver, just to allocate the matrix block
- Defines the solution parameters that are taken over by ParStokes

Block pre-conditioner

$$P = \begin{pmatrix} A & B^{(T)} \\ 0 & M \end{pmatrix}$$

```
Solver 3
 Equation = "Stokes"
 Procedure = "ParStokes" "StokesSolver"
 Element = "p:1 b:4"
 Bubbles in Global System = False
 Variable = FlowVar
 Variable Dofs = 4
 Convective = Logical False
 Block Diagonal A = Logical True
 Use Velocity Laplacian = Logical False
 !Keywords related to the block preconditioning
 Block Preconditioning = Logical True
 Linear System Scaling = Logical True
 Linear System Row Equilibration = Logical True
 Linear System Solver = "Iterative"
 Linear System GCR Restart = Integer 200
 Linear System Max Iterations = 200
 Linear System Convergence Tolerance = 1.0e-6
 Nonlinear System Max Iterations = 100
 Nonlinear System Convergence Tolerance = 1.0e-5
 Nonlinear System Newton After Tolerance = 1.0e-3
End
```

- The outer iteration of the saddle-point problem
- Pre-defined GCR method
- Needs the 2 dummy-solver (uses memory)

Mass consistent normals

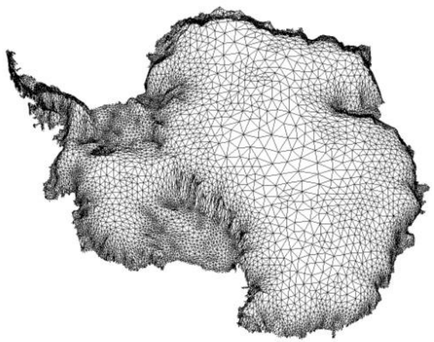
- Especially on noisy bed with linear test functions normal vectors are discontinuous
  - Artificial sink/source of mass
- New way to deduce mass consistent nodal normal vectors from elements

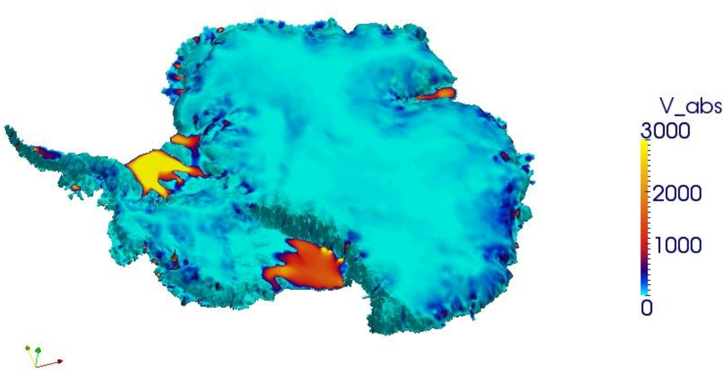
$$\vec{n}_j = \frac{1/N_j \sum_{i=1}^{N_j} \int_{\Omega_k} \vec{n}(\vec{x}_j)\varphi_k \, dV}{\|1/N_j \sum_{i=1}^{N_j} \int_{\Omega_k} \vec{n}(\vec{x}_j)\varphi_k \, dV\|}$$

```
Mass consistent normals = Logical True
```

M. A. Walkley, et al. , *On the calculation of normals in free-surface flow problems*, Comm. Num. Meth. Engrg., 20, 2004

Results AAIS





- 2D footprint reordered by YAMS (60k elements)
- Internally extruded using ALBMAP dataset (NetCDF reader + StructMeshMapper)
- Temperatures interpolated from Pattyn-output
- Sliding values from inversion on coarser mesh
- Block-Preconditioner as solution

Results AAIS