

Elmer/Ice Updates: A high-resolution coupled permafrost model

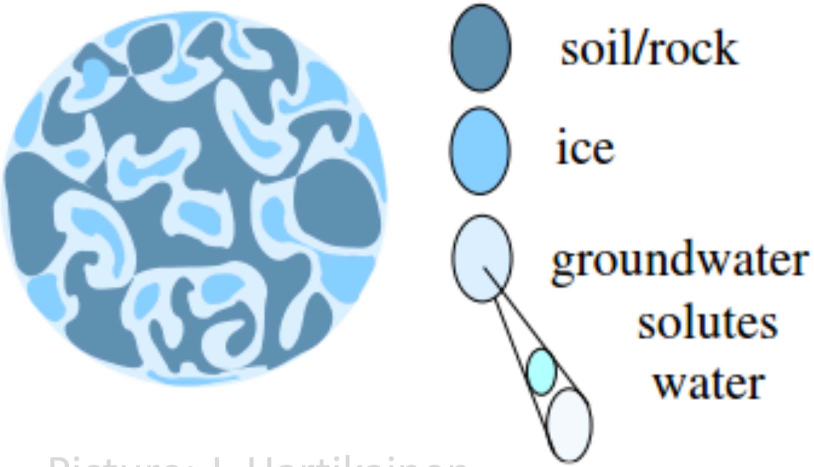
Thomas Zwinger[1], **Juha Hartikainen**[2], **Denis Cohen**[3], and **Peter Råback**[1]

[1] CSC-IT Centre for Science, Espoo, Finland
[2] Tampere University of Technology, Tampere, Finland
[3] New Mexico Tech, Socorro, NM, USA


Permafrost model

- **Saturated porous medium that consists of skeleton of rock or soil, ice and groundwater of water and dissolved salts** :

1. Heat transfer
2. Groundwater flow of saturated aquifer (Darcy)
3. Solute transport within groundwater
4. Deformation of bedrock (porosity)

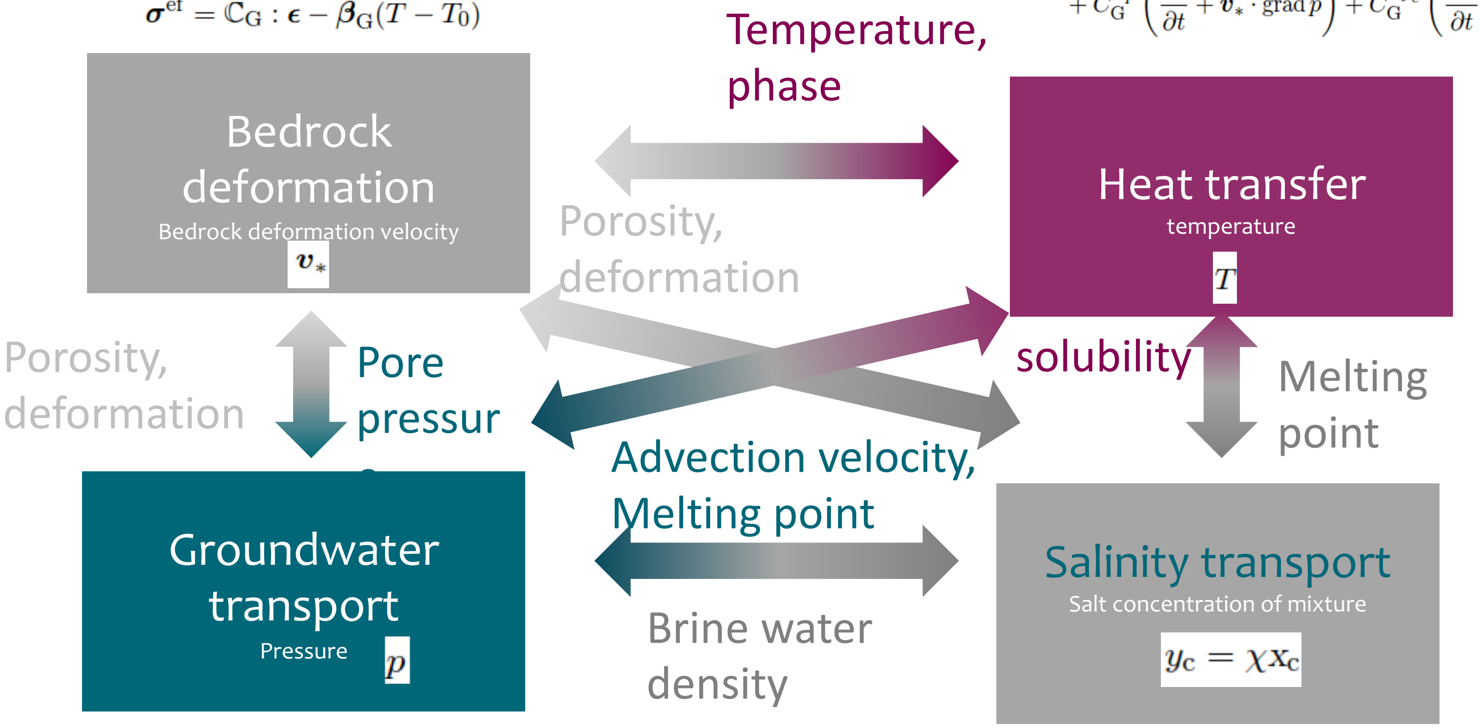


Picture: J. Hartikainen

Permafrost model

$$-\mathrm{div}\left(\boldsymbol{\sigma}^{\mathrm{ef}} - p\mathbf{I}\right) = \rho_{\mathrm{G}}\boldsymbol{g},$$
$$\boldsymbol{\sigma}^{\mathrm{ef}} = \mathbb{C}_{\mathrm{G}} : \boldsymbol{\epsilon} - \boldsymbol{\beta}_{\mathrm{G}}(T - T_0)$$

$$C_{\mathrm{G}}^{TT}\left(\frac{\partial T}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, T\right) + C_{\mathrm{gw}}^{TT}\,\mathrm{grad}\, T \cdot \boldsymbol{J}_{\mathrm{gw}}^{\mathrm{D}} + \mathrm{div}\, \boldsymbol{J}_{\mathrm{G}}^{\mathrm{H}} + \\ + C_{\mathrm{G}}^{Tp}\left(\frac{\partial p}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, p\right) + C_{\mathrm{G}}^{Ty_{\mathrm{c}}}\left(\frac{\partial T}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, y_{\mathrm{c}}\right) = S_{\mathrm{G}}$$



Temperature, phase

Bedrock deformation

Bedrock deformation velocity

$\boldsymbol{v}_*$

Heat transfer

temperature

T

Porosity, deformation

solubility

Porosity, deformation

Pore pressure

Melting point

Advection velocity, Melting point

Groundwater transport

Pressure p

Salinity transport

Salt concentration of mixture

$y_{\mathrm{c}} = \chi X_{\mathrm{c}}$

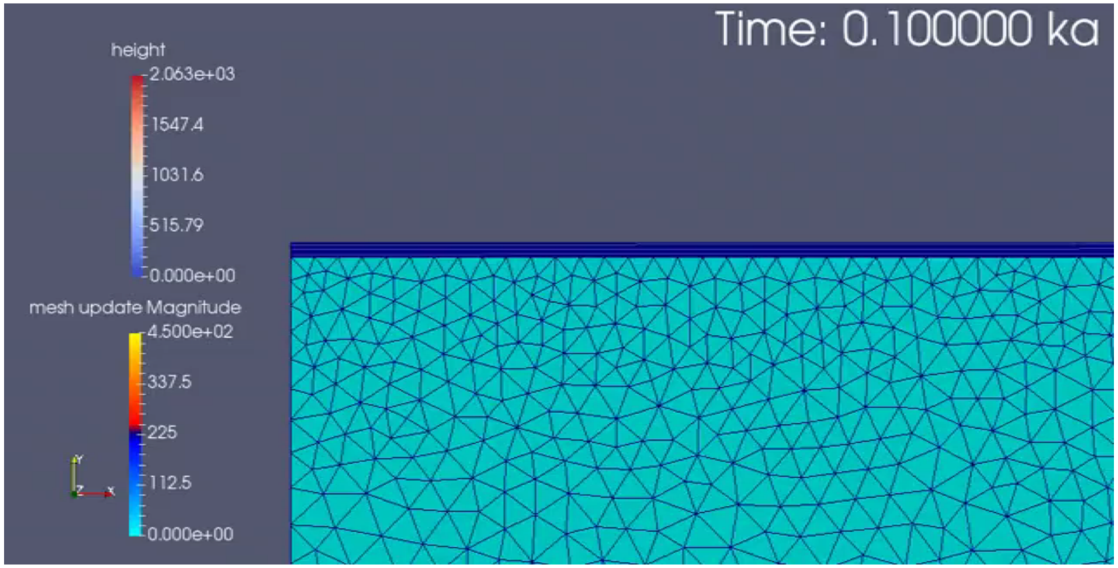
Brine water density

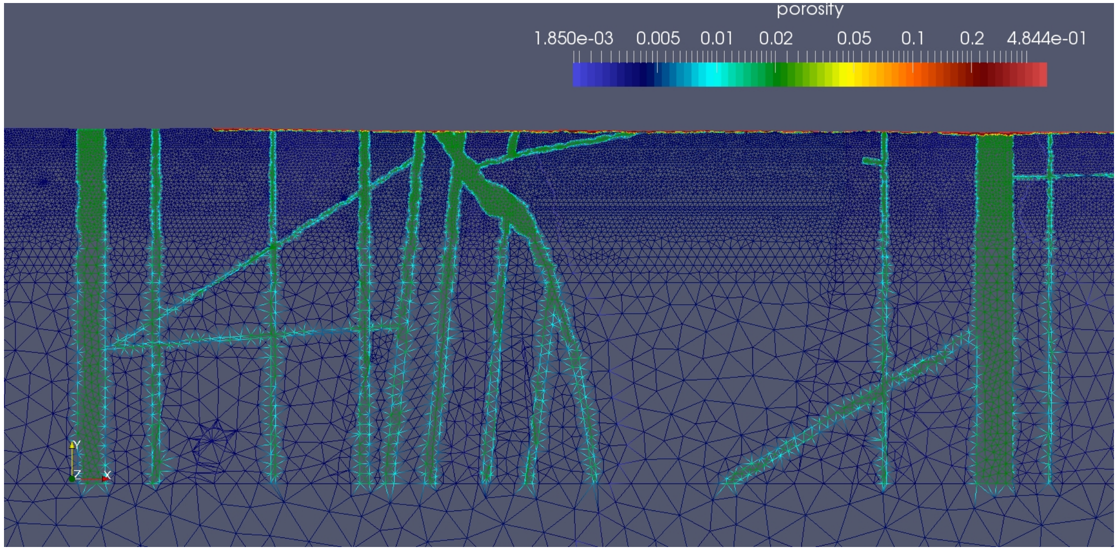
$$C_{\mathrm{gw}}^{pp}\left(\frac{\partial p}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, p\right) + \mathrm{div}\left(\varrho_{\mathrm{gw}}\boldsymbol{J}_{\mathrm{gw}}^{\mathrm{D}}\right) + C_{\mathrm{gw}}^{pT}\left(\frac{\partial T}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, T\right) + \\ + C_{\mathrm{gw}}^{py_{\mathrm{c}}}\left(\frac{\partial y_{\mathrm{c}}}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, y_{\mathrm{c}}\right) + \mathrm{div}\left[\eta\left(\varrho_{\mathrm{c}} - \varrho_{\mathrm{w}}\right)\boldsymbol{J}_{\mathrm{c}}^{\mathrm{F}}\right] - C_{\mathrm{gw}}^{pI_1}\left(\frac{\partial I_1}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, I_1\right) = S_{\mathrm{gw}}$$

$$C_{\mathrm{c}}^{y_{\mathrm{c}}y_{\mathrm{c}}}\left(\frac{\partial y_{\mathrm{c}}}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, y_{\mathrm{c}}\right) + \mathrm{div}\left(\frac{y_{\mathrm{c}}}{\chi}\varrho_{\mathrm{c}}\boldsymbol{J}_{\mathrm{gw}}^{\mathrm{D}}\right) + \mathrm{div}\left(\eta\varrho_{\mathrm{c}}\boldsymbol{J}_{\mathrm{c}}^{\mathrm{F}}\right) + \\ + C_{\mathrm{c}}^{y_{\mathrm{c}}T}\left(\frac{\partial T}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, T\right) + C_{\mathrm{c}}^{y_{\mathrm{c}}p}\left(\frac{\partial p}{\partial t} + \boldsymbol{v}_* \cdot \mathrm{grad}\, p\right) = S_{\mathrm{c}}$$

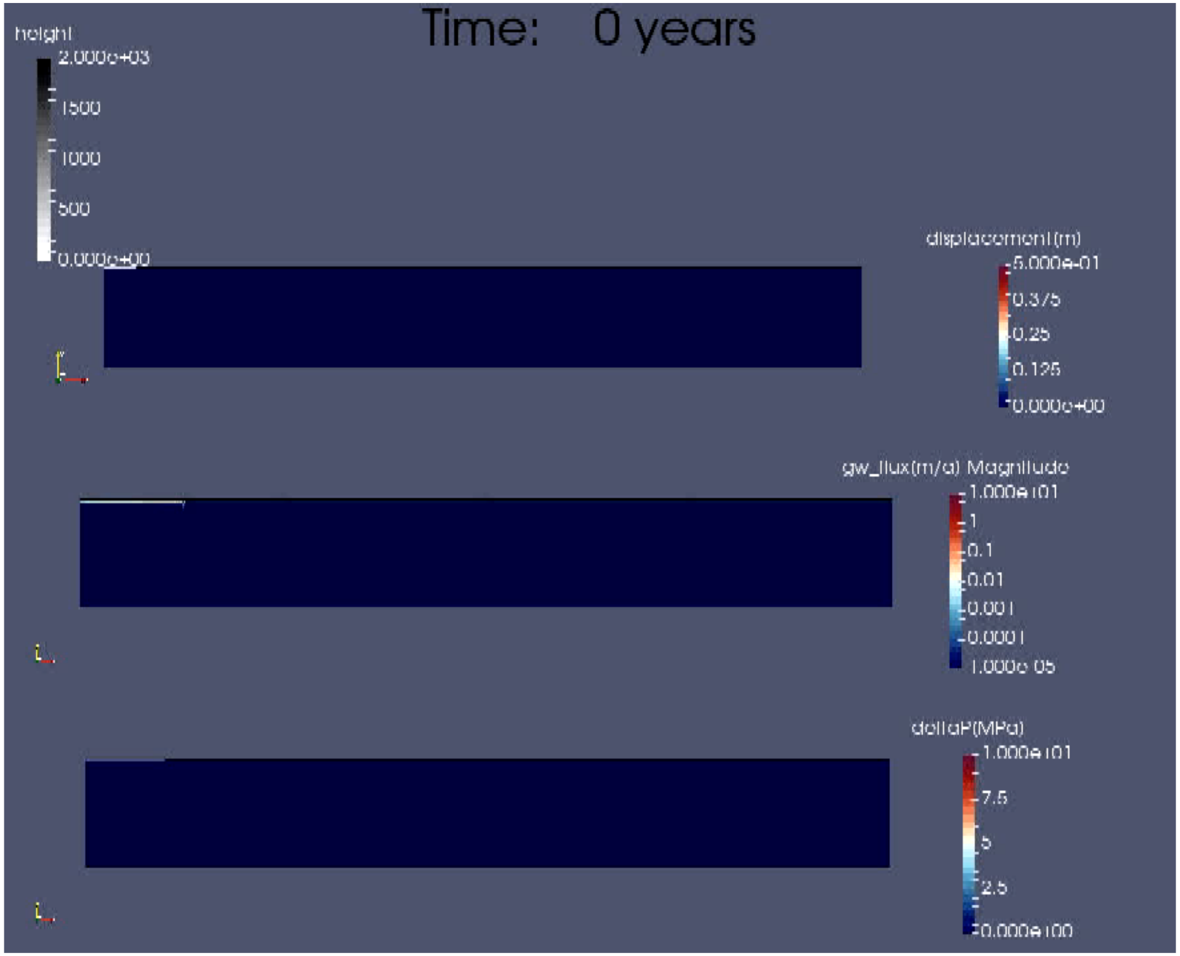
Permafrost Model

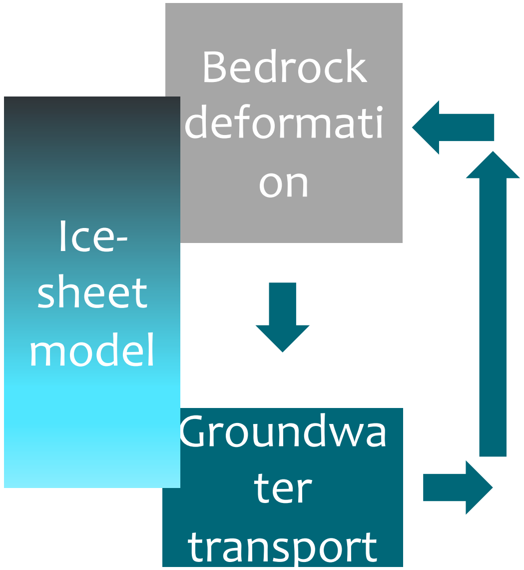
- Multiple bodies
- Different mesh-concepts:
    - **Ice-sheet:** structured, layered mesh
    - **Bedrock:** unstructured, in places high-resolution mesh
    - Offset for displacement: Model for glacial isostatic adjustment (LLRA)





Ice-sheet advance

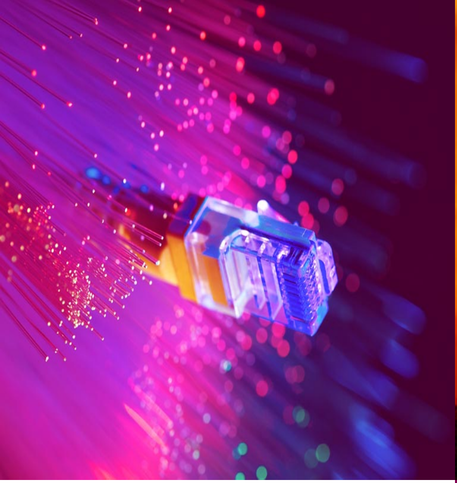




Bedrock deformation

Ice-sheet model

Groundwater transport



Elmer/Ice Updates: A versatile visco-elastic Earth deformation model

Thomas Zwinger[1], **Juha Ruokolainen**[1], **Grace Nield**[2,3], and **Matt King**[3]

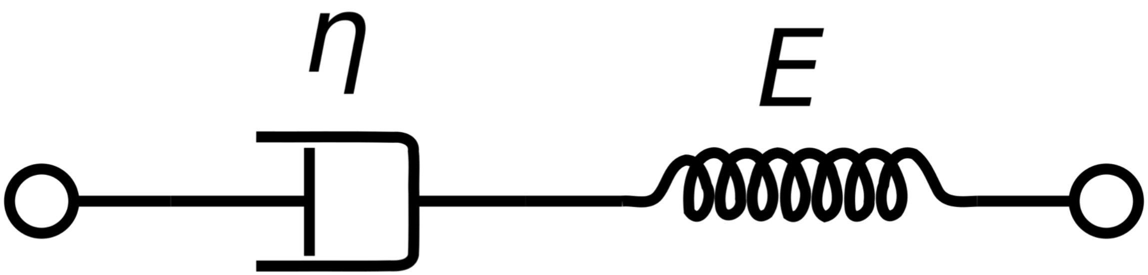
[1] CSC-IT Centre for Science, Espoo, Finland, Europe
[2] Durham Univ., Durham, UK, (soon not) Europe
[3] UTAS, Hobart, Tasmania, Australia

CSC – Suomalainen tutkimuksen, koulutuksen, kulttuurin ja julkishallinnon ICT-osaamiskeskus

Maxwell rheology

- Standard FE linear elasticity: $\vec{\nabla} \cdot \overline{\overline{\tau}} = 0,$
- Elastic rheology: stress as a function of reversible deformation
- Visco-elastic: (partly non-reversible) deformation as a function of

viscous and elastic contribution



By Pekaje at English Wikipedia - Transferred from en.wikipedia to Commons., Public Domain

Implementation into Elmer

- Introduction of visco-elastic stress (Wu 2004)

$$\partial_t \overline{\overline{\tau}} = \partial_t \overline{\overline{\tau}}^0 - \frac{\mu}{\nu}\left(\overline{\overline{\tau}} - \Pi \overline{\overline{I}}\right),$$

$$\overline{\overline{\tau}}^0 = \lambda\theta \overline{\overline{I}} + 2\mu\overline{\overline{\varepsilon}},$$

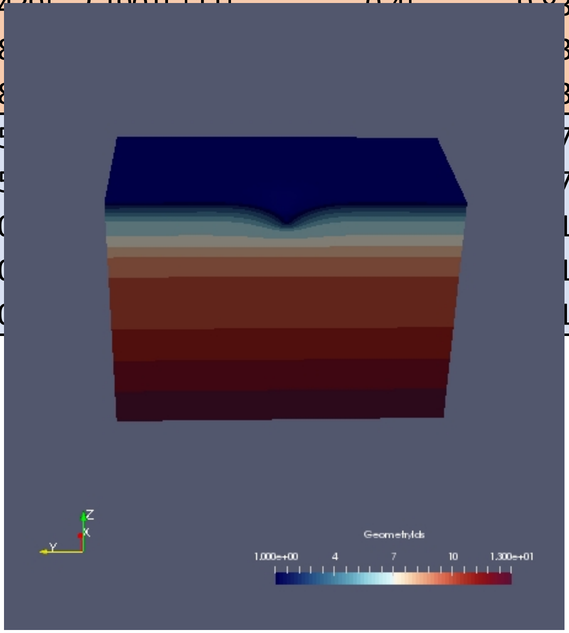
- At the same time we introduce a pressure Π to enable incompressibility
- Additional term accounting for restoring force by specific weight gradient

$$\vec{\nabla} \cdot \overline{\overline{\tau}} - \rho_o g_o \vec{\nabla} w = 0,$$

- This is not standard in commercial FE packages, hence needs to be "cheated" around by putting jump-conditions on inter-layer boundaries (Wrinkler foundations)
- In Elmer we can include this, which introduces the right boundary condition naturally over

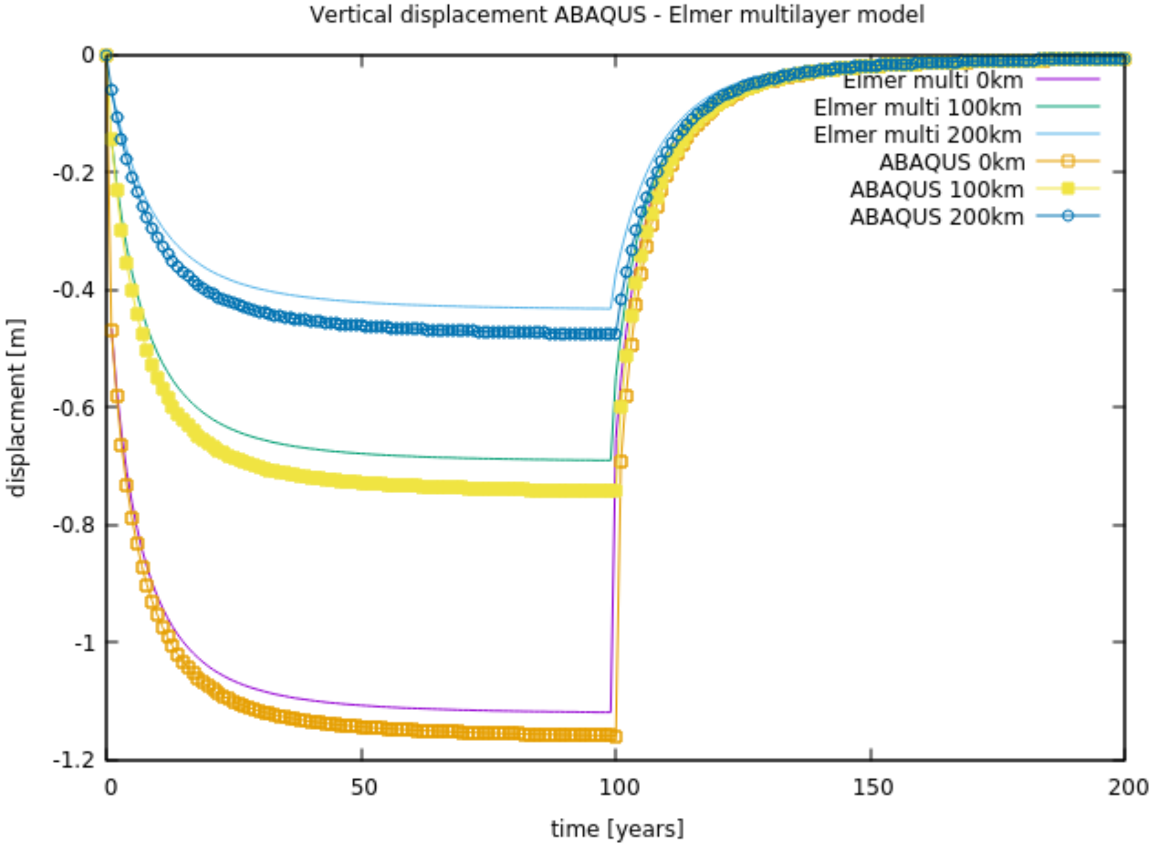
GIA benchmark model

Layer	Layer top (radius, km)	Layer base (radius, km)	Thickness (km)	Viscosity	Density	Young's Modulus	Poisson's Ratio	Gravitational Acceleration
Lithosphere	6371	6336	35	1x10^44	3196	1.8148E+11	0.4	9.7852
Lithosphere	6336	6301	35	1x10^44	3196	1.8148E+11	0.4	9.7852
Lithosphere	6301	6251	50	1x10^44	3196	1.8148E+11	0.4	9.7852
Upper Mantle	6251	6201	50	1x10^18	3439	2.1901E+11	0.4	9.8367
Upper Mantle	6201	6141	60	1x10^18	3439	2.1901E+11	0.4	9.8367
Upper Mantle	6141	5971	170	1x10^18	3439	2.1901E+11	0.4	9.8367
Upper Mantle	5971	5835	136	1x10^18	38			349
Upper Mantle	5835	5701	134	1x10^18	38			349
Lower Mantle	5701	5450	251	1x10^22	45			799
Lower Mantle	5450	4770	680	1x10^22	45			799
Lower Mantle	4770	4340	430	1x10^22	50			108
Lower Mantle	4340	3910	430	1x10^22	50			108
Lower Mantle	3910	3480	430	1x10^22	50			108

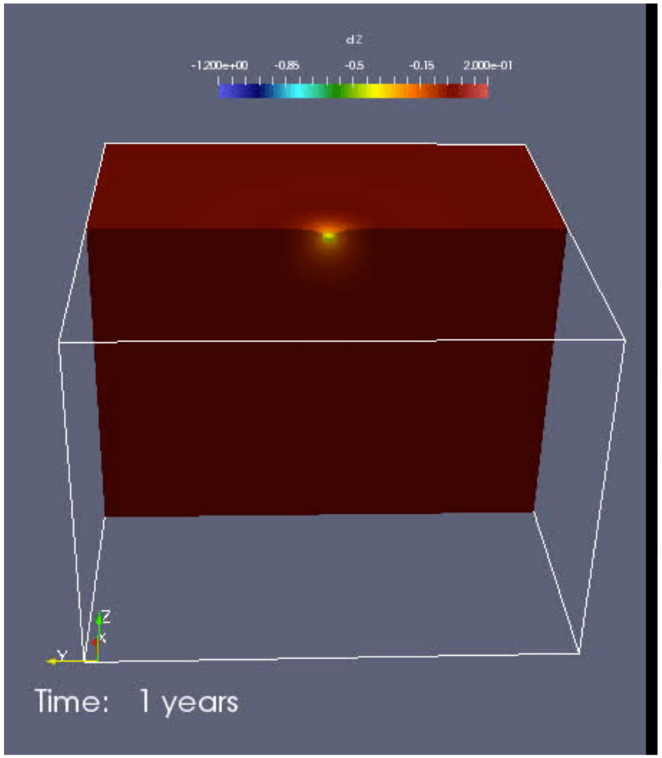


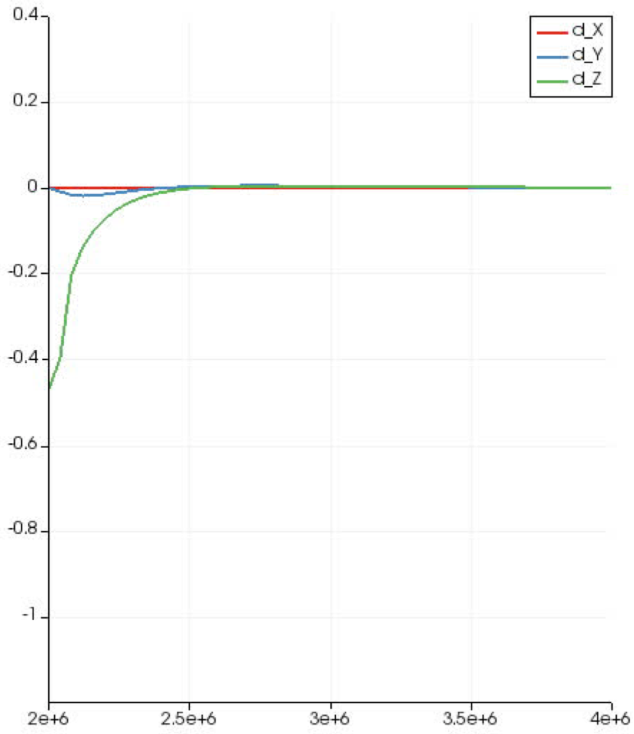
GIA benchmark model

- Total width 4000km (2000km each side of the ice load centre)
- Depth – surface to core (6371 – 3480km)
- Load:
  - Disc radius: 50km (dia 100km)
  - Disc thickness: 100m
  - Ice density: 917 kg/m3
  - Loading 100 years, unloading 100 years



Benchmark run: 100 km- diameter





Elmer/Ice Updates: A (even faster) scaling Stokes ice-flow solver

Elmer team + Intel (Intel Parallel Computing Center = IPCC)

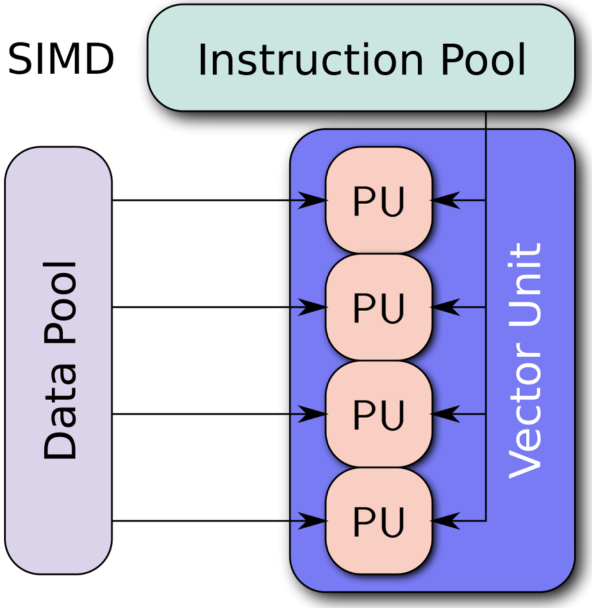
At UTAS: Chen Zhao for several MIPS

CSC – Suomalainen tutkimuksen, koulutuksen, kulttuurin ja julkishallinnon ICT-osaamiskeskus

Vectorized Stokes Solver

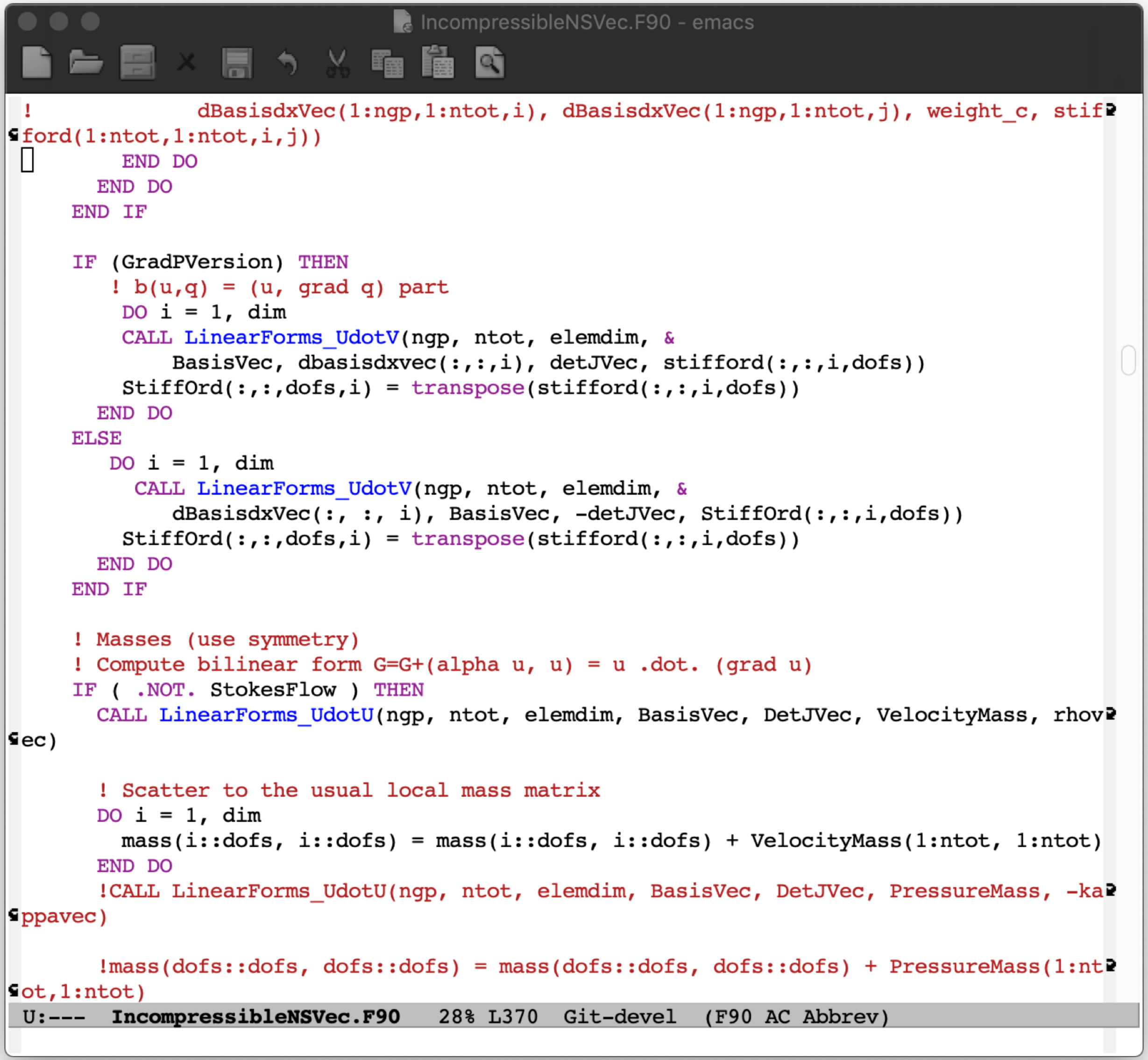
- New computer architectures use SIMD (=vector) units to do fast computations
- If you (on an Intel chip) don't utilize this, you a priori loose ¾ of your performance
- FEM: assembly = creating the matrix

  solution = solving it
- Until recently, assembly procedures in Elmer did not utilize SIMD
- New Stokes solver does!
- It also recently go the block-preconditioner functionality to increase solution efficiency



By Vadikus - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=39715273

Vectorized Stokes Solver

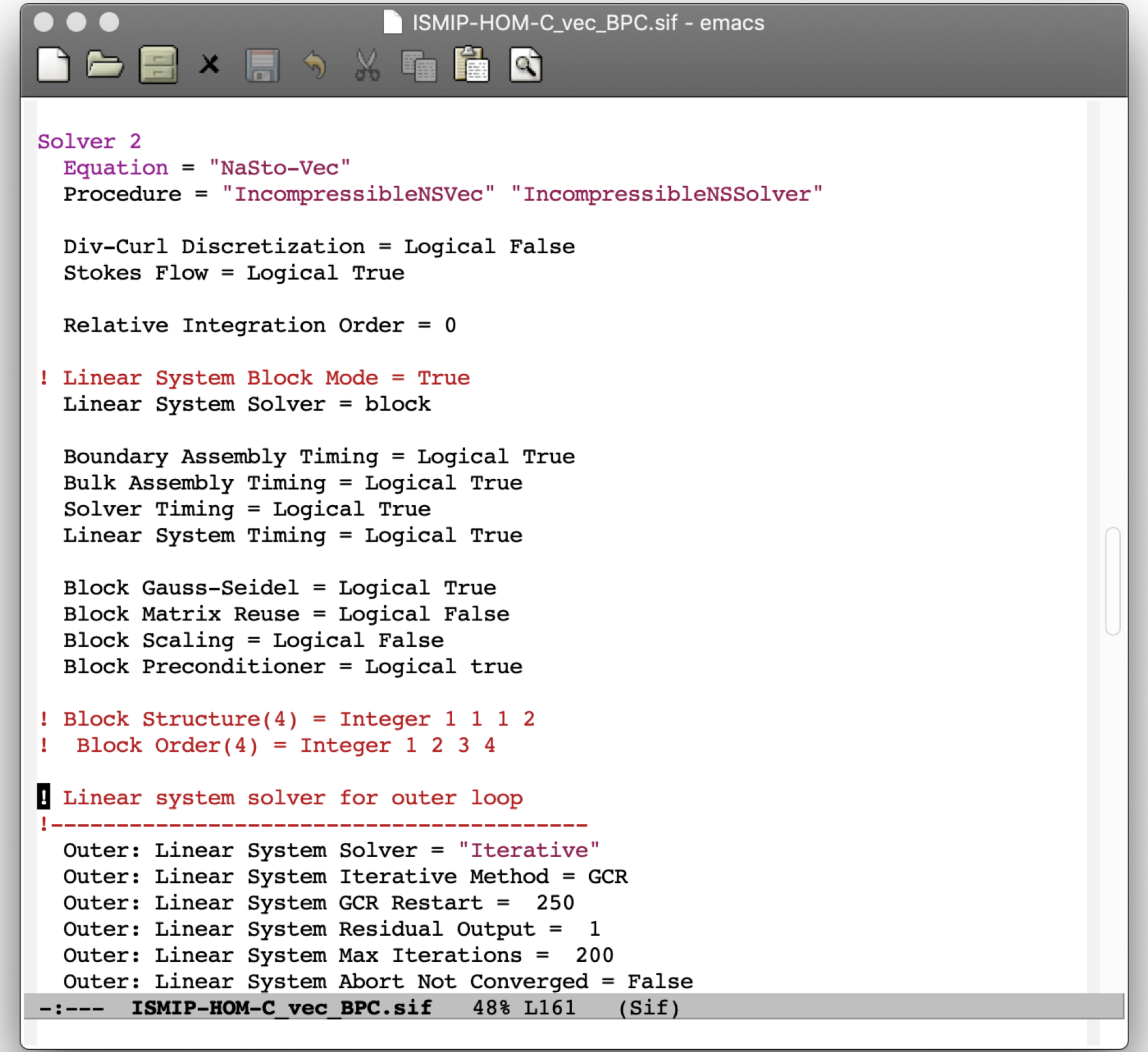


```
!             dBasisdxVec(1:ngp,1:ntot,i), dBasisdxVec(1:ngp,1:ntot,j), weight_c, stif
ford(1:ntot,1:ntot,i,j))
        END DO
      END DO
    END IF

    IF (GradPVersion) THEN
       ! b(u,q) = (u, grad q) part
        DO i = 1, dim
        CALL LinearForms_UdotV(ngp, ntot, elemdim, &
            BasisVec, dbasisdxvec(:,:,i), detJVec, stifford(:,:,i,dofs))
        StiffOrd(:,:,dofs,i) = transpose(stifford(:,:,i,dofs))
      END DO
    ELSE
       DO i = 1, dim
         CALL LinearForms_UdotV(ngp, ntot, elemdim, &
            dBasisdxVec(:, :, i), BasisVec, -detJVec, StiffOrd(:,:,i,dofs))
        StiffOrd(:,:,dofs,i) = transpose(stifford(:,:,i,dofs))
      END DO
    END IF

    ! Masses (use symmetry)
    ! Compute bilinear form G=G+(alpha u, u) = u .dot. (grad u)
    IF ( .NOT. StokesFlow ) THEN
      CALL LinearForms_UdotU(ngp, ntot, elemdim, BasisVec, DetJVec, VelocityMass, rhov
ec)

      ! Scatter to the usual local mass matrix
      DO i = 1, dim
        mass(i::dofs, i::dofs) = mass(i::dofs, i::dofs) + VelocityMass(1:ntot, 1:ntot)
      END DO
      !CALL LinearForms_UdotU(ngp, ntot, elemdim, BasisVec, DetJVec, PressureMass, -ka
ppavec)

      !mass(dofs::dofs, dofs::dofs) = mass(dofs::dofs, dofs::dofs) + PressureMass(1:nt
ot,1:ntot)
U:---  IncompressibleNSVec.F90    28% L370   Git-devel  (F90 AC Abbrev)
```

Vectorized Stokes Solver

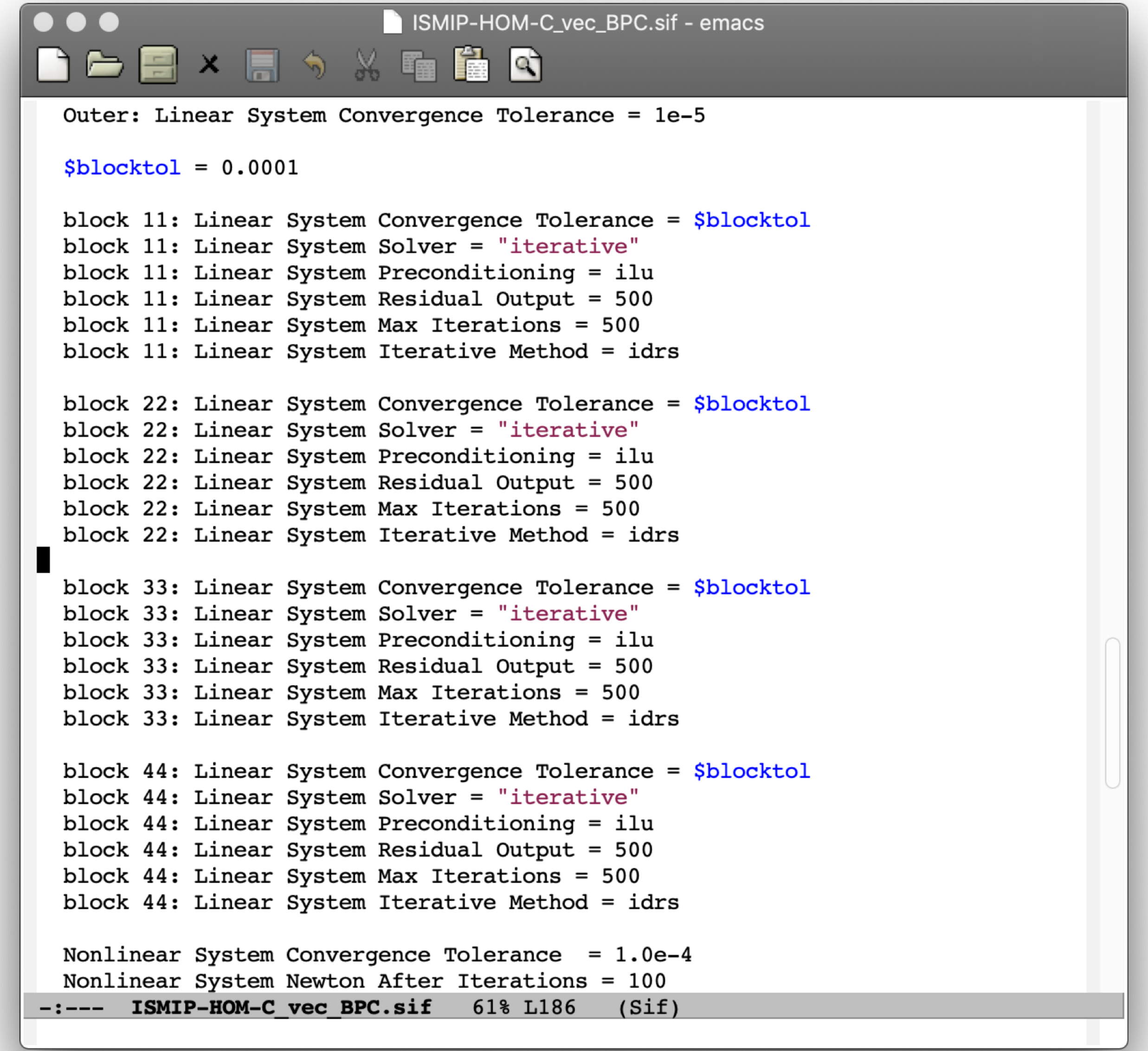
- Solver works basically like legacy solver, except for the assembly being SIMD parallel
- Switch off Div-curl discretization (else we have wrogn natural BC’s)
- You can (don’t have to) use the library version of the block-preconditioner
- Else, just use the iteration method of your choice
- ISMIP-HOM-C (solved with cPardiso in both cases) was about 1/3rd the solution time of a comparable legacy solver run

Vectorized Stokes Solver



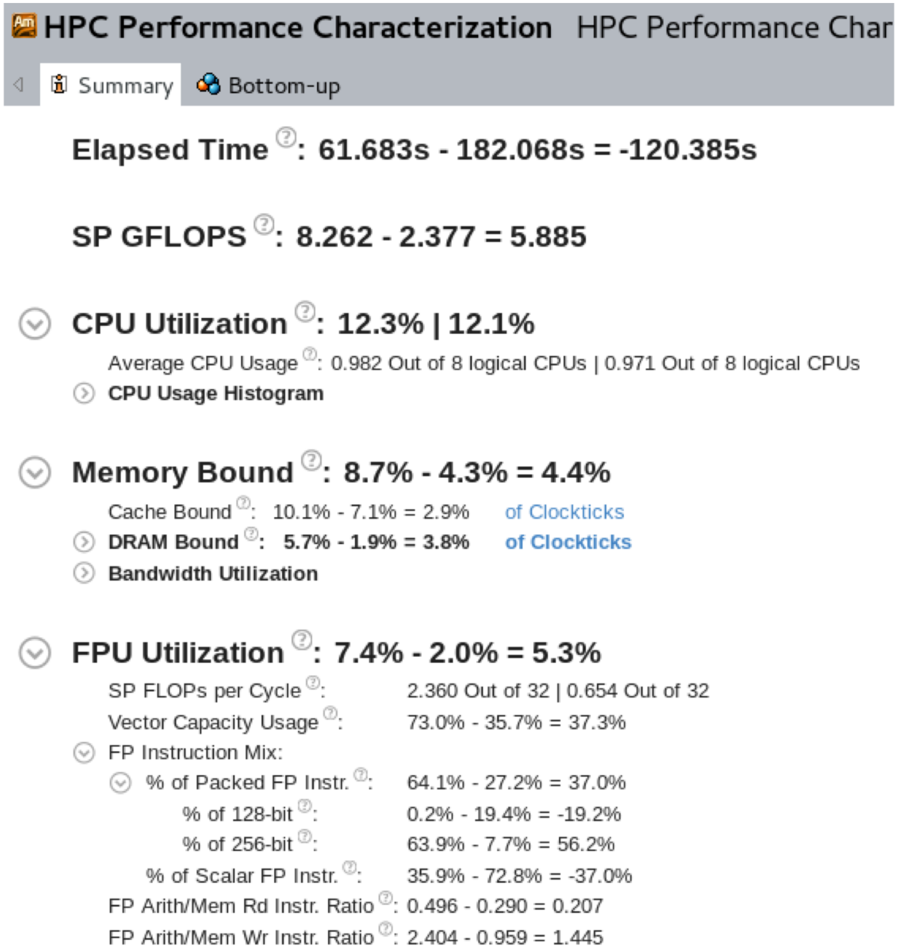
```
Solver 2
  Equation = "NaSto-Vec"
  Procedure = "IncompressibleNSVec" "IncompressibleNSSolver"

  Div-Curl Discretization = Logical False
  Stokes Flow = Logical True

  Relative Integration Order = 0

! Linear System Block Mode = True
  Linear System Solver = block

  Boundary Assembly Timing = Logical True
  Bulk Assembly Timing = Logical True
  Solver Timing = Logical True
  Linear System Timing = Logical True

  Block Gauss-Seidel = Logical True
  Block Matrix Reuse = Logical False
  Block Scaling = Logical False
  Block Preconditioner = Logical true

! Block Structure(4) = Integer 1 1 1 2
!  Block Order(4) = Integer 1 2 3 4

! Linear system solver for outer loop
!-----------------------------------------
  Outer: Linear System Solver = "Iterative"
  Outer: Linear System Iterative Method = GCR
  Outer: Linear System GCR Restart =  250
  Outer: Linear System Residual Output =  1
  Outer: Linear System Max Iterations =  200
  Outer: Linear System Abort Not Converged = False
-:---  ISMIP-HOM-C_vec_BPC.sif    48% L161    (Sif)
```

Vectorized Stokes Solver



```
Outer: Linear System Convergence Tolerance = 1e-5

$blocktol = 0.0001

block 11: Linear System Convergence Tolerance = $blocktol
block 11: Linear System Solver = "iterative"
block 11: Linear System Preconditioning = ilu
block 11: Linear System Residual Output = 500
block 11: Linear System Max Iterations = 500
block 11: Linear System Iterative Method = idrs

block 22: Linear System Convergence Tolerance = $blocktol
block 22: Linear System Solver = "iterative"
block 22: Linear System Preconditioning = ilu
block 22: Linear System Residual Output = 500
block 22: Linear System Max Iterations = 500
block 22: Linear System Iterative Method = idrs

block 33: Linear System Convergence Tolerance = $blocktol
block 33: Linear System Solver = "iterative"
block 33: Linear System Preconditioning = ilu
block 33: Linear System Residual Output = 500
block 33: Linear System Max Iterations = 500
block 33: Linear System Iterative Method = idrs

block 44: Linear System Convergence Tolerance = $blocktol
block 44: Linear System Solver = "iterative"
block 44: Linear System Preconditioning = ilu
block 44: Linear System Residual Output = 500
block 44: Linear System Max Iterations = 500
block 44: Linear System Iterative Method = idrs

Nonlinear System Convergence Tolerance  = 1.0e-4
Nonlinear System Newton After Iterations = 100
-:---  ISMIP-HOM-C_vec_BPC.sif   61% L186   (Sif)
```

Comparison vectorised/legacy Solver using Intel VTune



Elapsed Time: 61.683s - 182.068s = -120.385s

SP GFLOPS: 8.262 - 2.377 = 5.885

CPU Utilization: 12.3% | 12.1%

Average CPU Usage: 0.982 Out of 8 logical CPUs | 0.971 Out of 8 logical CPUs

CPU Usage Histogram

Memory Bound: 8.7% - 4.3% = 4.4%

Cache Bound: 10.1% - 7.1% = 2.9% of Clockticks

DRAM Bound: 5.7% - 1.9% = 3.8% of Clockticks

Bandwidth Utilization

FPU Utilization: 7.4% - 2.0% = 5.3%

SP FLOPs per Cycle: 2.360 Out of 32 | 0.654 Out of 32

Vector Capacity Usage: 73.0% - 35.7% = 37.3%

FP Instruction Mix:

- % of Packed FP Instr.: 64.1% - 27.2% = 37.0%
  - % of 128-bit: 0.2% - 19.4% = -19.2%
  - % of 256-bit: 63.9% - 7.7% = 56.2%
- % of Scalar FP Instr.: 35.9% - 72.8% = -37.0%

FP Arith/Mem Rd Instr. Ratio: 0.496 - 0.290 = 0.207

FP Arith/Mem Wr Instr. Ratio: 2.404 - 0.959 = 1.445

Comparison vectorised/legacy Solver using Intel VTune