High performance computing and the Finite Element Method

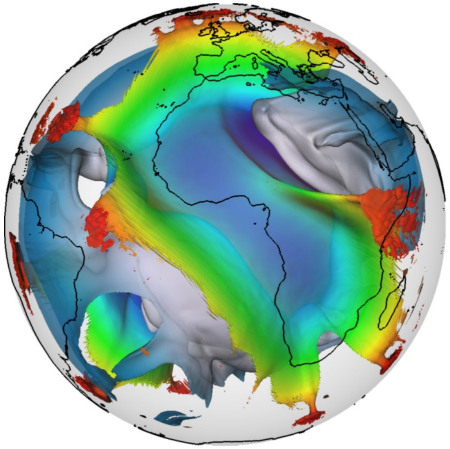
MATH 9830

Timo Heister
(heister@clemson.edu)

http://www.math.clemson.edu/~heister/

About this class

- Finite Element solutions used across sciences
- High performance is critical:
  how to achieve it? Understand the details!
- Class modality: online only



Hello, my name is … Timo Heister

- Associate Professor at Clemson
- Started in 2013 in Clemson
- Born in Germany
- PhD from Göttingen, Germany
- Postdoc at Texas A&M

Introductions

- Your name
- Where are you from?
- Master or Phd? Adviser?
- Research project in one sentence (elevator!)
- Something about you: Hobby or Random fact?

Class structure

- Some lectures/demos
- some flipped classrooms with reading/videos at home and discussion during class time
- Assessment:
  - Homework: 25%
  - Journal & library contributions: 25%
  - Project presentation & report: 50%

Topics

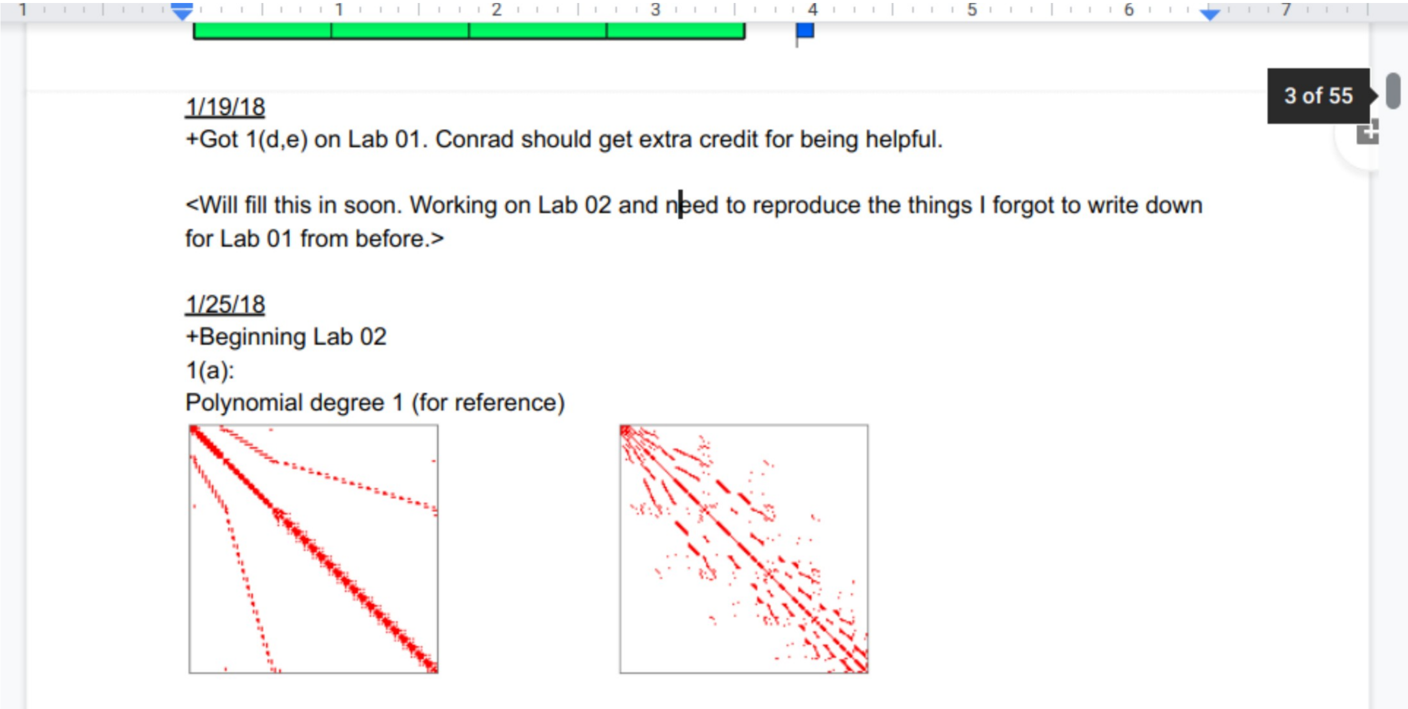
- Implementation aspects of FEM
- High-performance Computing and FEM
- Introduction to the deal.II library: Using/extending the library
- Parallel FEM using MPI
- Student projects
- And:
  - Software development tools
  - C++ software design (templates…)
- Potentially:
  - Multithreading
  - Discontinous Galerkin
  - Solvers, Preconditioners

Goals

- Outcomes:
  - Solve your own FEM problems, appreciate sophisticated libraries, work on your research
  - Useful skills: research, national labs, companies, ...
- Different skill levels, so:
  - Help others
  - Improve deal.II (documentation, typo fixes, tutorials, functionality)

Tasks for you

- Communicate:
  - just lecturing is boring for all of us, please be active
- Give feedback
- Keep your journal going!



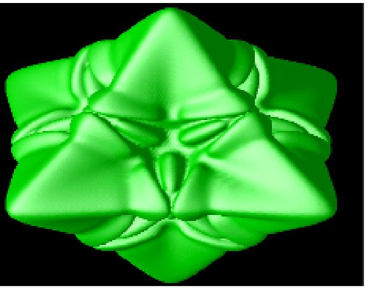
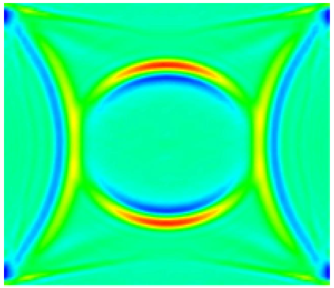
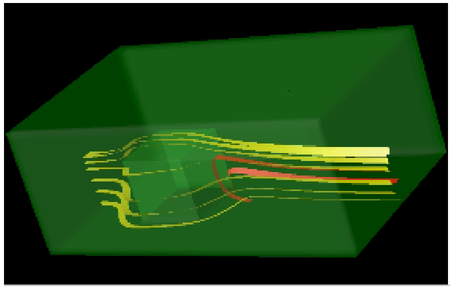
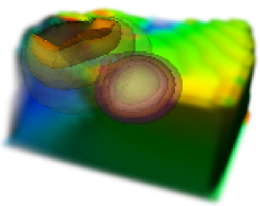
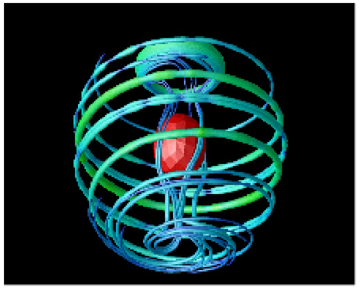
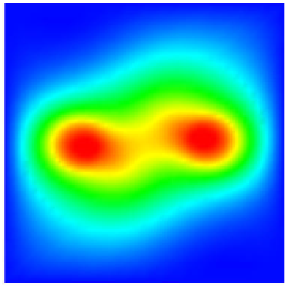
1/19/18
+Got 1(d,e) on Lab 01. Conrad should get extra credit for being helpful.

<Will fill this in soon. Working on Lab 02 and need to reproduce the things I forgot to write down for Lab 01 from before.>

1/25/18
+Beginning Lab 02
1(a):
Polynomial degree 1 (for reference)

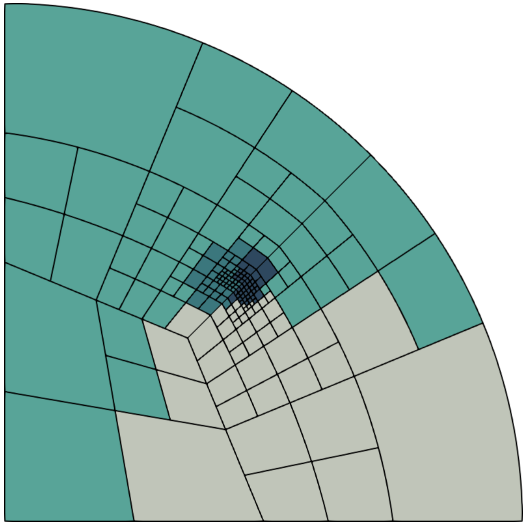
deal.II

- “A Finite Element **D**ifferential **E**quations **A**nalysis **L**ibrary”
- Open source, community project, written in C++
- I am one of the 11 principal developers
- One of the most widely used libraries:
  - 1400+ papers using and citing deal.II
  - 100+ people have contributed in the past 15 years
  - 600,000+ lines of code
  - 10,000+ pages of documentation
- Website: www.dealii.org

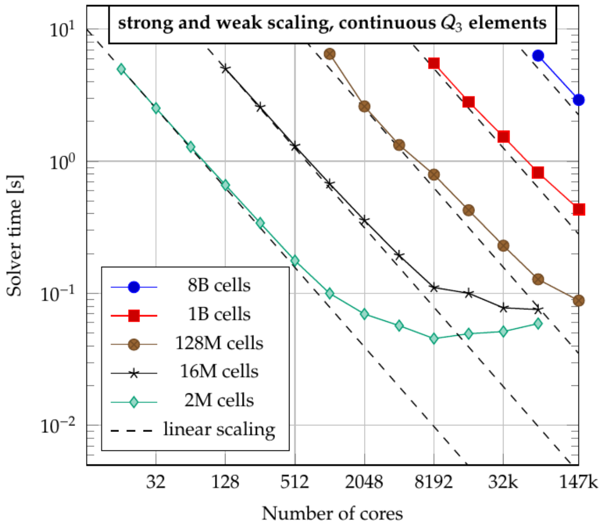







Features

- 1d, 2d, 3d computations, adaptive mesh refinement (on quads/hexas only*)
- Finite element types:
  - Continuous and DG Lagrangian elements
  - Higher order elements, hp adaptivity
  - Raviart-Thomas, Nedelec, …
  - And arbitrary combinations
- Excellent documentation:
  - 70+ tutorial “steps”, detailed documentation

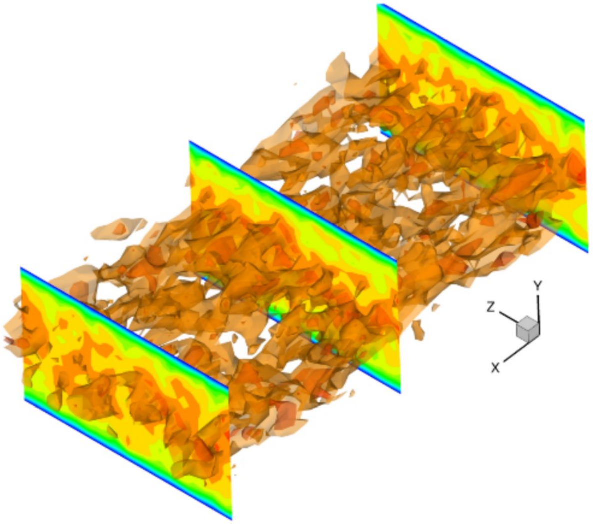


Features: Performance

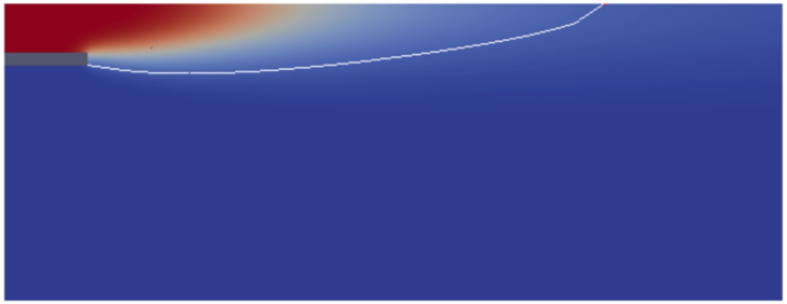


- Linear Algebra
  - Own sparse and dense library
  - Interfaces to PETSc, Trilinos, UMFPACK, BLAS, ..
  - Matrix-free linear algebra routines
- Parallelization
  - MPI: 100,000+ processors
  - 100+ billion unknowns
  - State of the art: matrix-free, SIMD, multi-threading, MPI

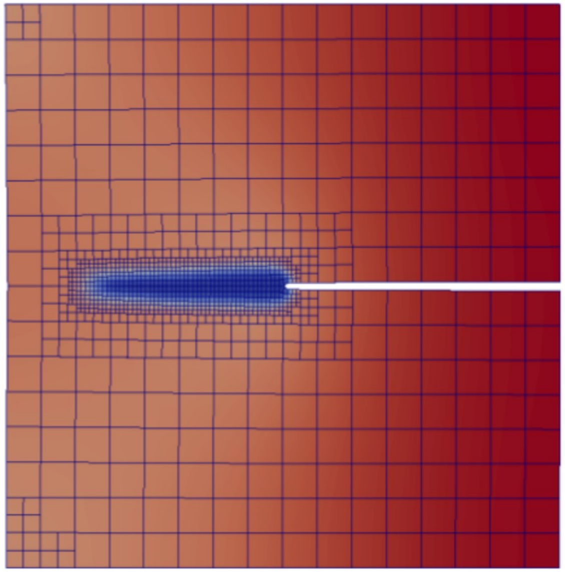
Some examples



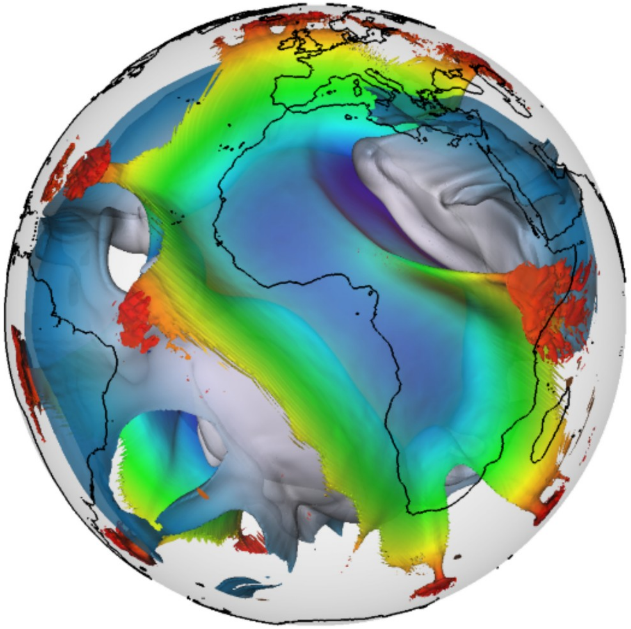
Turbulent flows



Laminar flames

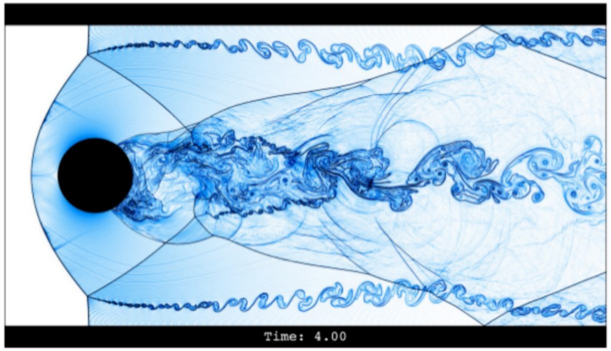


crack propagation

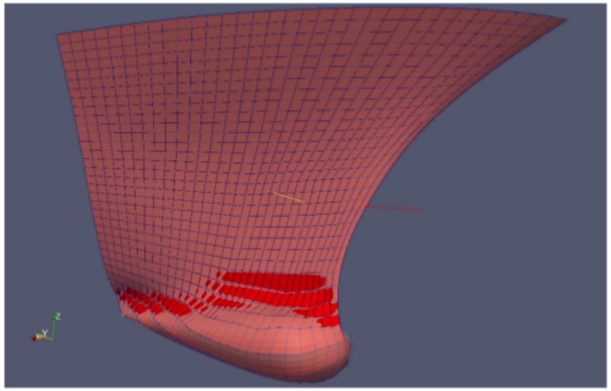


Convection in the Earth's mantle
(ASPECT code)

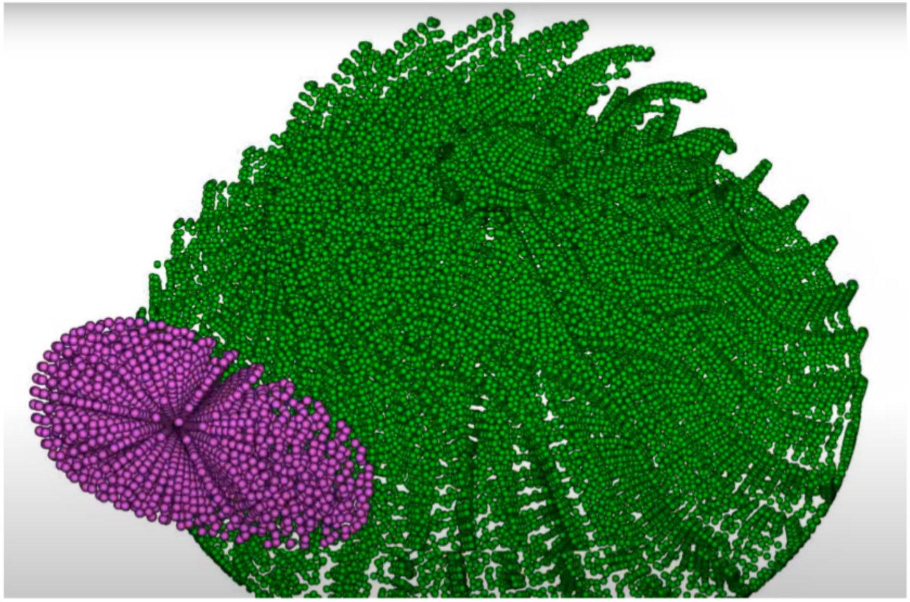
More examples



Euler flow (step-69)



BEM on ship hull
(base: step-54)



coupled fluid-structure interaction
(step-70)

Questions for you

- Open Questionnaire
- Show notes

Installation

- Ubuntu (or any other Linux) or MacOS or Windows: https://github.com/dealii/dealii/wiki/Getting-deal.II
- More details:
  - Version 9.2.0 is enough for now (maybe development version later)
  - Virtual machine works (easiest), but will be somewhat slow to do work in (maybe a good start?)
- Dependencies:
  - For now: nothing needed
  - Later: MPI, Trilinos
- I would suggest:
  WSL on Windows, .dmg on MacOS, Ubuntu packages on Linux

On MacOS

- follow instructions from wiki:

  https://github.com/dealii/dealii/wiki/MacOSX
- Installer:
  - .dmg installer at https://github.com/dealii/dealii/releases/tag/v9.2.0
  - Later manually: QtCreator, Eclipse, ParaView
- Alternative: install XCode as described on wiki, then install manually

On Windows

- Install WSL (Windows subsystem for Linux)
- This will give basically give you a Debian (or Ubuntu) Linux system where you can install everything needed
- https://github.com/dealii/dealii/wiki/Windows
- Alternative: Virtual Machine

Linux

- Without extra dependencies: easy
- Newest Ubuntu? Use the binary packages
- Later: QtCreator, ParaView

Install deal.II manually

- https://www.dealii.org/download.html
  https://www.dealii.org/current/readme.html
- Extract:

```
tar xf deal.II-9.2.0.tar.gz
```

- Build directory:

```
mkdir build; cd build
```

- Configuration:

```
cmake -D CMAKE_INSTALL_PREFIX=??? ../dealii-9.2.0/
```

  (where ??? is your installation directory, for example `../installed`)

- Compile (5-60 minutes):

```
make -j X install
```

  (where X is the number of cores you have)

- Test:

  `make test` (in build directory)

- Test part two:

```
cd examples/step-1
cmake -D DEAL_II_DIR=??? .
make run
```

- Recommended layout:

```
deal.II/
  build          < build files
  installed      < your inst. dir
  dealii-9.2.0     < source
    examples     < the examples!
    include
    source
    ...
```

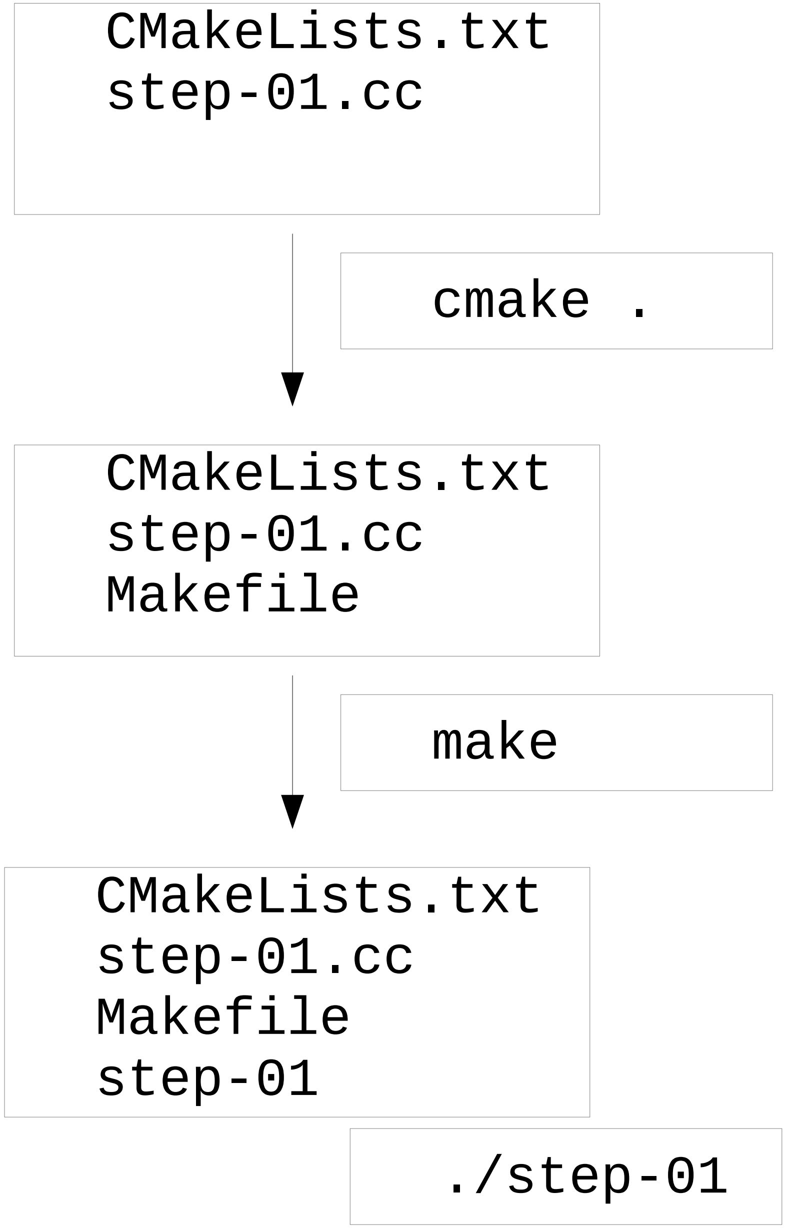
candi

- Automated installer under Linux including dependencies
- https://github.com/dealii/candi/
- (probably not needed for now though)

Palmetto Cluster

- deal.II is already installed by me. Instructions: http://www.math.clemson.edu/~heister/palmetto.html

Configuring/Compiling/Running C++ code



```
CMakeLists.txt
step-01.cc
```

```
cmake .
```

```
CMakeLists.txt
step-01.cc
Makefile
```

```
make
```

```
CMakeLists.txt
step-01.cc
Makefile
step-01
```

```
./step-01
```

- CMake:
  - Detect OS, compiler, dependencies, etc..
  - Generates `Makefile`
- Make:
  - Compile source (.cc) to executable (`step-01`)
- Finally:
  - `./step-01` to run
  - `make run` compiles and runs