deal.II:
a numerical library to tackle realistic challenges from industry and academia

Luca Heltai
SISSA – International School for Advanced Studies

Adapted from a talk by

Wolfgang Bangerth
Texas A&M University

In collaboration with many others around the world.

Motivations

Many of the big problems in scientific computing are described by partial differential equations (PDEs):

- Structural statics and dynamics
  – Bridges, roads, cars, …
- Fluid dynamics
  – Ships, pipe networks, …
- Aerodynamics
  – Cars, airplanes, rockets, …
- Plasma dynamics
  – Astrophysics, fusion energy
- But also in many other fields: Biology, finance, epidemiology, ...

Numerics for PDEs

There are 3 standard tools for the numerical solution of PDEs:

- Finite element method (FEM)
- Finite volume method (FVM)
- Finite difference method (FDM)

Common features:

- Split the domain into small volumes (cells)
- Define balance relations on each cell
- Obtain and solve very large (non-)linear systems

Problems:

- Every code has to implement these steps
- There is only so much time in a day
- There is only so much expertise anyone can have

Ideal Characteristics

Examples of what we would like to have:

- Adaptive meshes
- Realistic, complex geometries

- Quadratic or even higher order elements

- Multigrid solvers
- Scalability to 1000s of processors
- Efficient use of current hardware

- Graphical output suitable for high quality rendering

Q: How can we make all of this happen in a single code?

Main Question!

Q: How can we make all of this happen in a single code?

Not a question of feasibility but of how we develop software:

- Is every student developing their own software?
- Or are we re-using what others have done?

- Do we insist on implementing everything from scratch?
- Or do we build our software on existing libraries?

There has been a major shift on how we approach the second question in scientific computing over the past 10-15 years!

MHPC
Master in High Performance Computing

The bitter reality…

Research software today:

- Typically written by graduate students
  – without a good overview of existing software
  – with little software experience
  – with little incentive to write high quality code
- Often maintained by postdocs
  – with little time
  – need to consider software a tool to write papers
- Advised by faculty
  – with no time
  – oftentimes also with little software experience

SISSA ICTP

About software

MHPC
Master in High Performance Computing

Observation 1:

Most research software is not of high quality.

How does this affect our field? (Reproducibility? Archival? "Standing on the shoulders of giants"?)

SISSA ICTP

About software

Observation 2:

There is a complexity limit to what we can get out of a student.

About software

Solutions:

- Creating software is an art and science. So:

What makes software successful?
(Best practices? Lessons learned?)

The secret to good scientific software is (re)using existing libraries!

Existing Software

There is excellent software for almost every purpose!

Basic linear algebra (dense vectors, matrices):

- BLAS
- LAPACK

Parallel linear algebra (vectors, sparse matrices, solvers):

- PETSc
- Trilinos

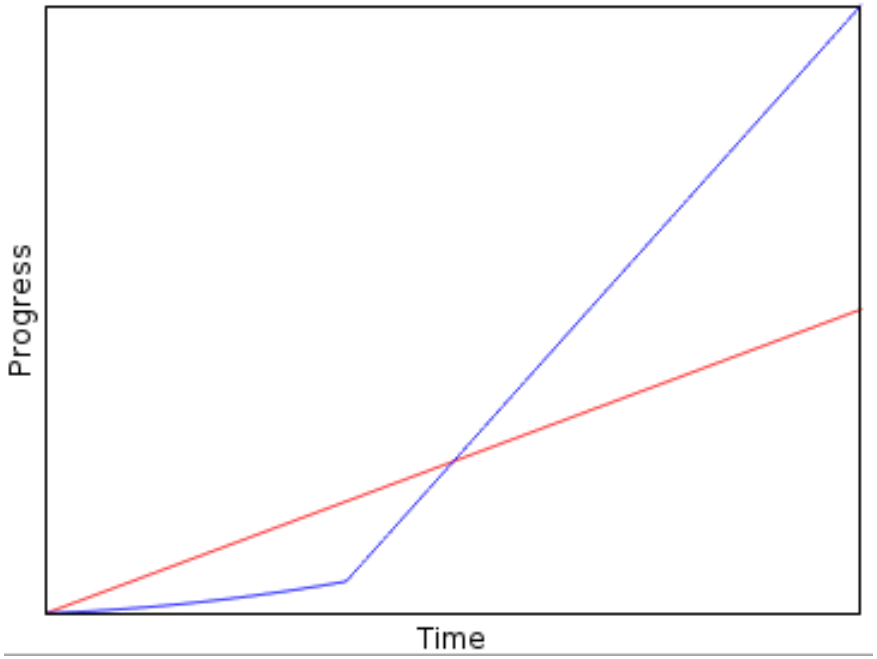
Meshes, finite elements, etc:

- deal.II – the topic of this class
- …

Visualization, dealing with parameter files, ...

Existing Software

Progress over time:

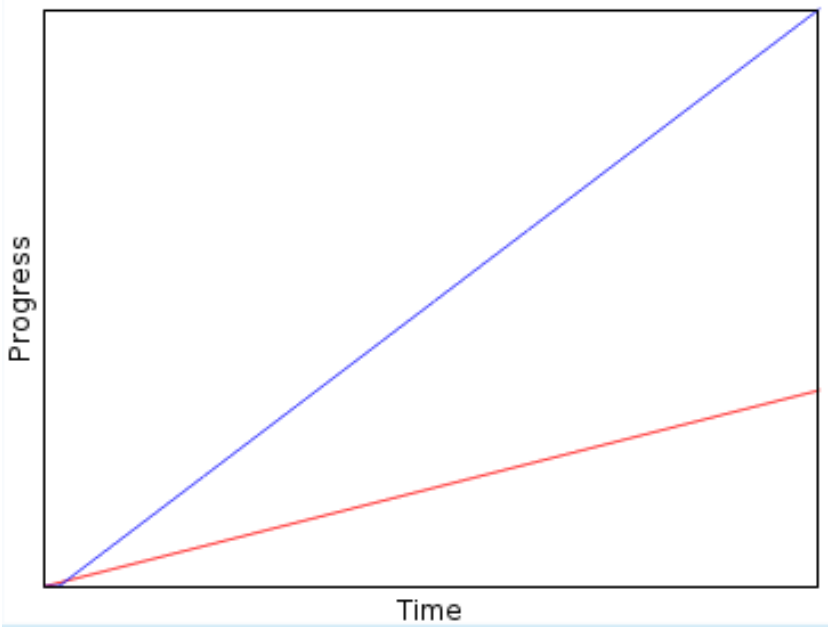


Red: Do it yourself. Blue: Use existing software.

Question: Where is the cross-over point?

Existing Software

Progress over time, the real picture:



Red: Do it yourself. Blue: Use existing software.

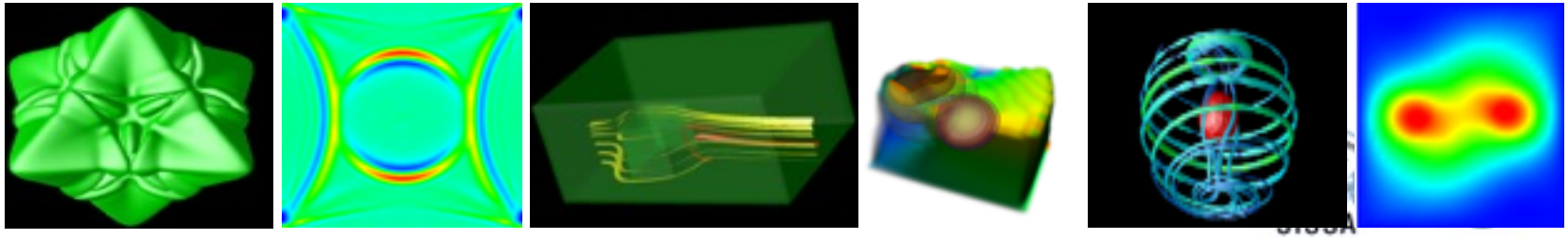
Answer: Cross-over is after 2–4 weeks! A PhD takes 3–4 years.

MHPC
Master in High Performance Computing

The Number 1 strategy… LIBRARIES!

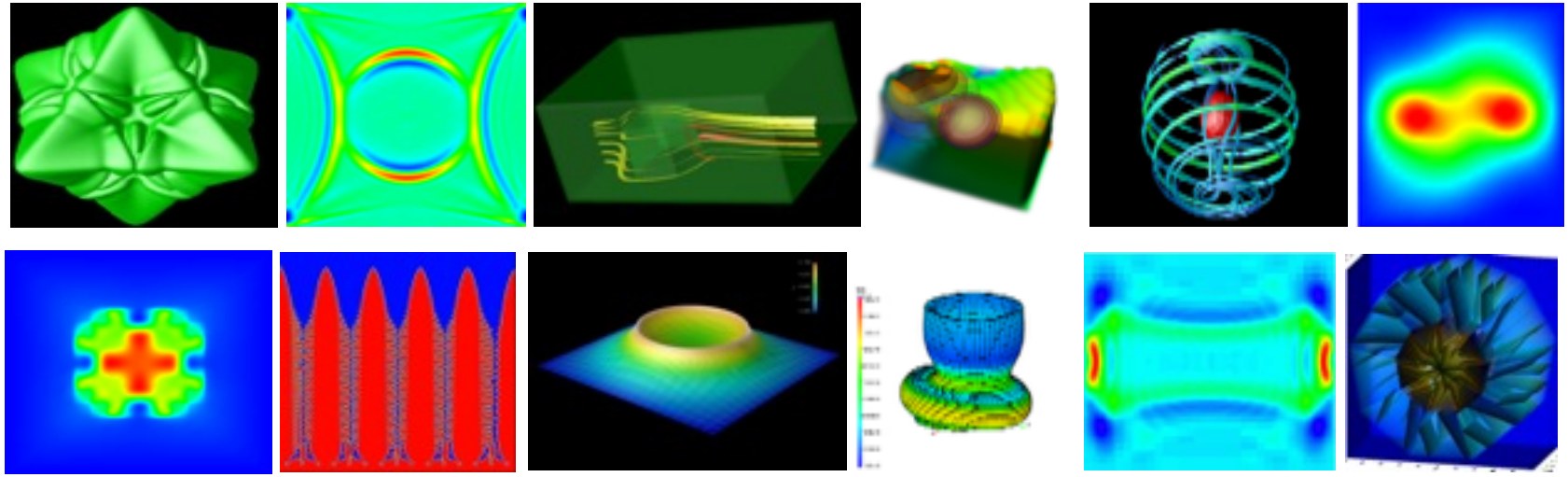
Our example today:

The *deal.II* library



deal.II

A library for finite element computations that supports...



...a large variety of PDE applications
tailored to non-experts.

deal.II

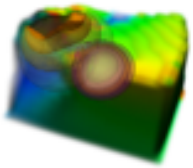
We want a library that:

- Supports complex computations in many fields
- Is general (not area-specific)
- Has fully adaptive, dynamically changing 3d meshes
- Scales to 10,000s of processors
- Is efficient on today's multicore machines

Fundamental premise:
Provide building blocks that can be used in many different ways, not a rigid framework.

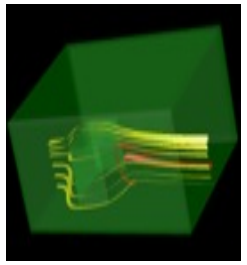
MHPC
Master in High Performance Computing

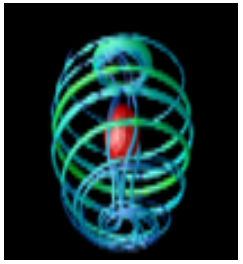
deal.II



deal.II provides:

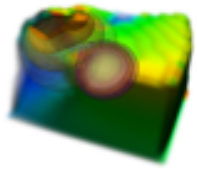
- Adaptive meshes in 1d, 2d, and 3d
- Interfaces to all major graphics programs
- Standard refinement indicators built in
- Many standard finite element types (continuous, discontinuous, mixed, Raviart-Thomas, ...)
- Low and high order elements
- Support for multi-component problems
- Its own sub-library for dense + sparse linear algebra
- Interfaces to PETSC, Trilinos, UMFPACK, ARPACK, SLEPc, MUMPS, SCALAPACK…
- Supports SMP + cluster systems

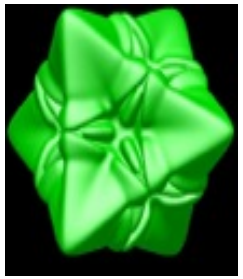


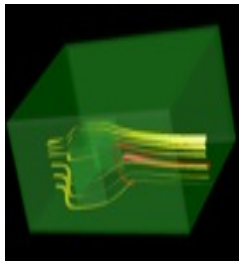


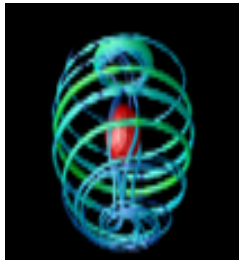
SISSA ICTP

deal.II







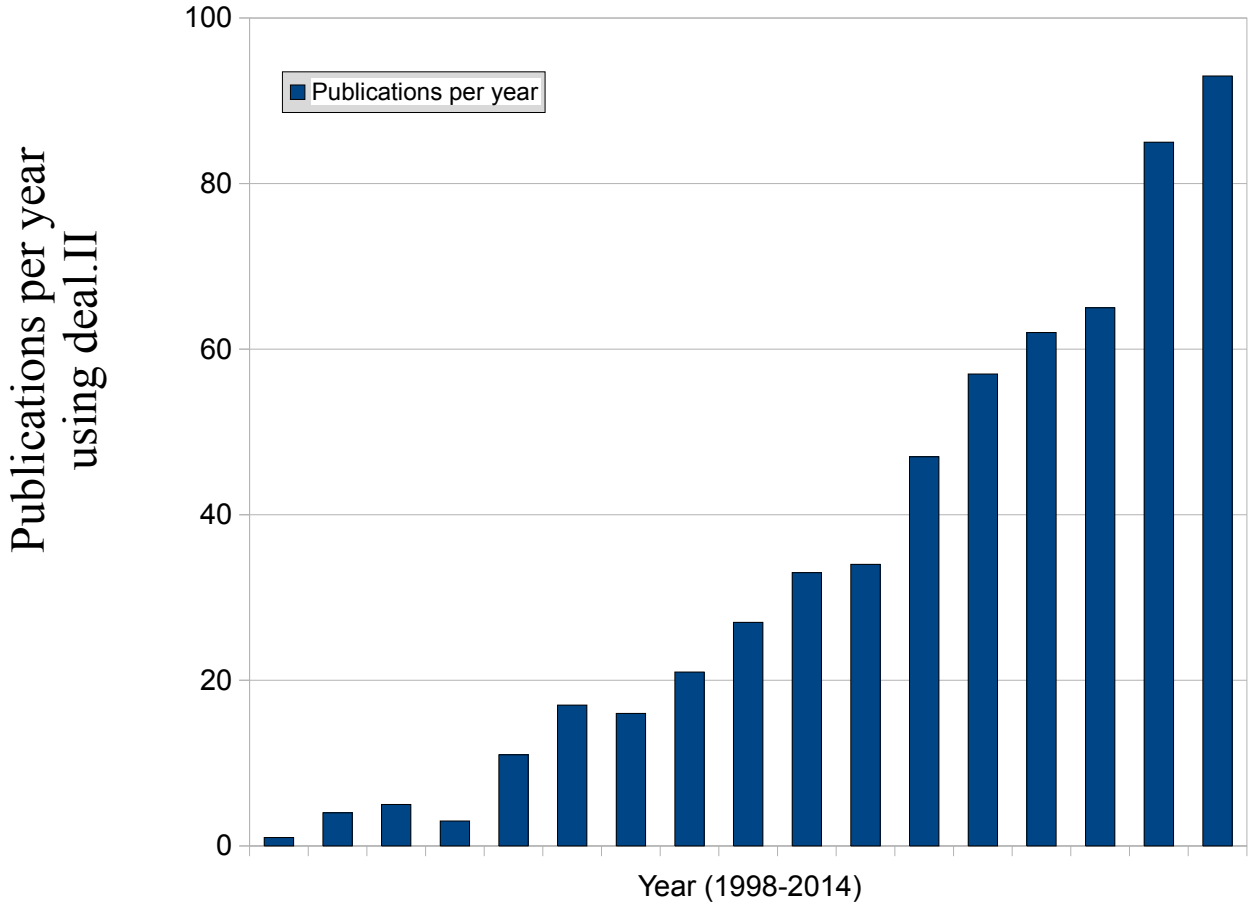


Status today:

- 500+ downloads per month
- 600,000 lines of code
- 10,000+ pages of documentation
- Portable build environment
- Used in teaching at several universities

MHPC
Master in High Performance Computing

Publications per year using deal.II:



SISSA ICTP

MHPC
Master in High Performance Computing

Examples

Examples of what can be done with deal.II (2013 only):

- Biomedical imaging
- Brain biomechanics
- E-M brain stimulation

- Microfluidics
- Oil reservoir flow
- Fuel cells
- Transonic aerodynamics
- Foam modeling
- Fluid-structure interactions
- Atmospheric sciences

- Quantum mechanics
- Neutron transport
- Nuclear reactor modeling

- Numerical methods research

- Fracture mechanics
- Damage models
- Solidification of alloys
- Laser hardening of steel
- Glacier mechanics
- Plasticity
- Contact/lubrication models

- Electronic structure
- Photonic crystals

- Financial modeling

- Chemically reactive flow
- Flow in the Earth mantle

SISSA ICTP

What makes such projects successful?

General observations:

Success or failure of scientific software projects
is not decided on technical merit alone.

The *true* factors are beyond the code!
It is not enough to be a good programmer!

In particular, what counts:

- Utility and quality
- Documentation
- Community

All of the big libraries provide this for their users.

Ease of use

Take utility as an example:

- Lots of error checking in the code
- Extensive testsuites
- Meaningful error messages and assertions rather than cryptic error codes

- Cataloged use cases
- FAQs
- Well documented examples of debugging common problems

Documentation/Education

Take documentation and education as an example:

- Installation instructions/README
- Within-function comments
- Function interface documentation
- Class-level documentation
- Module-level documentation
- Worked “tutorial” programs
- Recorded, interactive demonstrations

Example: deal.II has 10,000+ HTML pages. 170,000 lines of code are actually documentation (~10 man years of work). There are 48 recorded video lectures on YouTube.

Examples

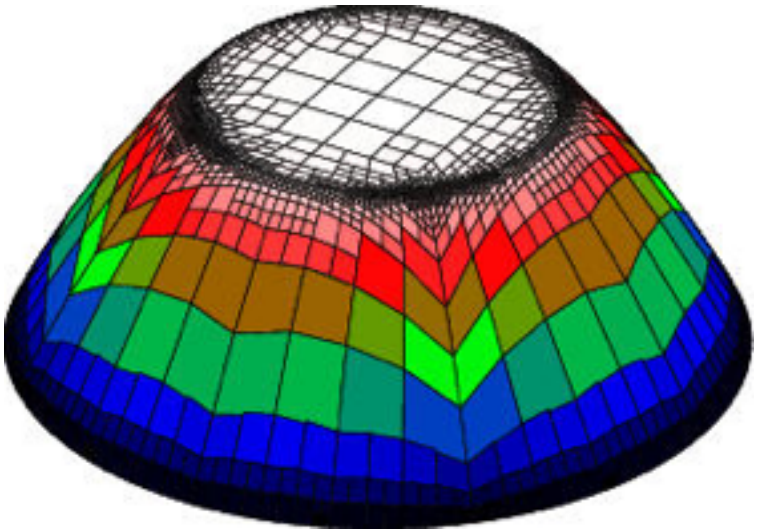
deal.II comes with ~60 tutorial programs:

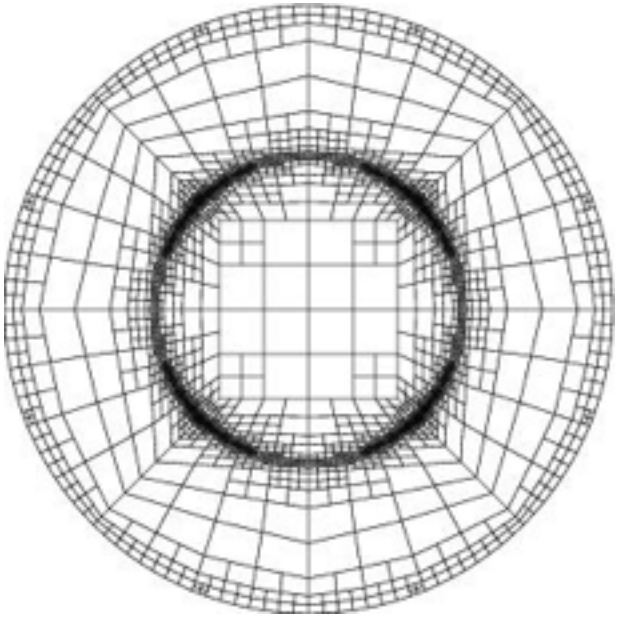
- From small Laplace solvers (~100s of lines)
- To medium-sized applications (~1000s of lines)
- Intent:
  – teach deal.II
  – teach advanced numerical methods
  – teach software development skills

Examples

Step-6:

- Laplace equation, variable coefficient
- Adaptive mesh refinement
- 118 lines of code

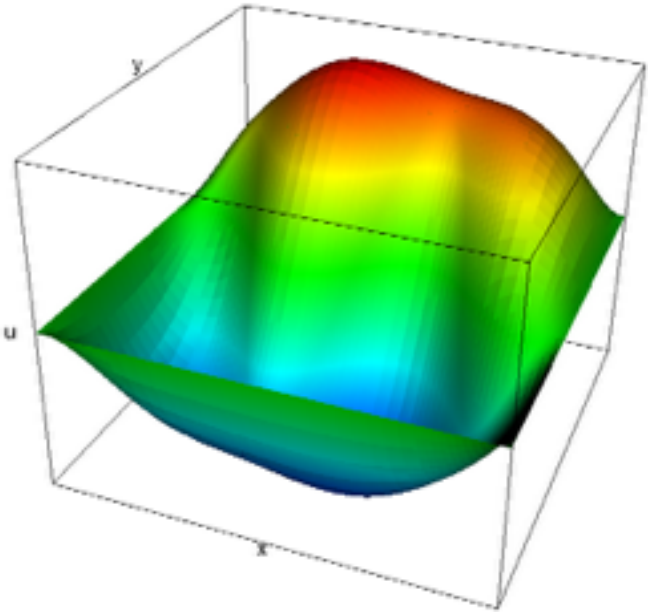


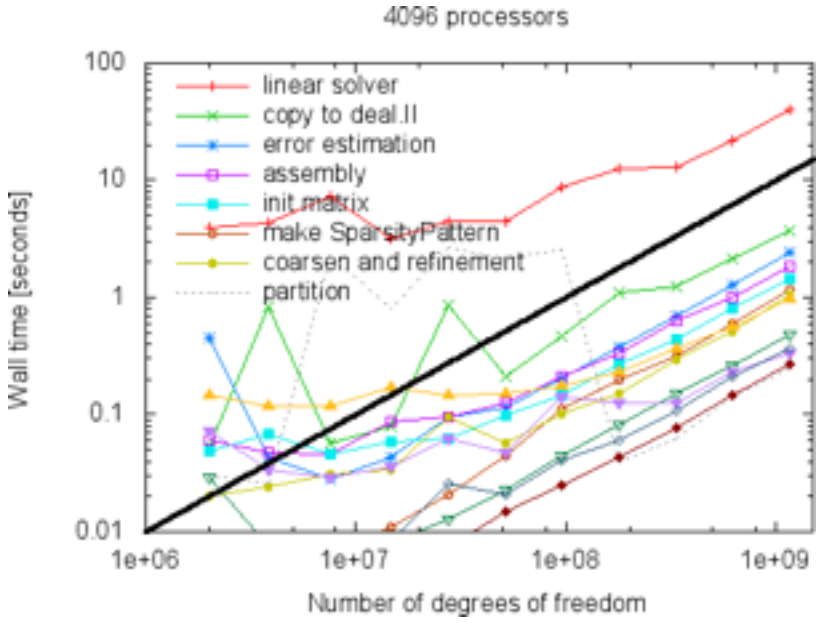


Examples

Step-40:

- Laplace equation, variable coefficient, 2d or 3d
- Adaptive mesh refinement
- Massively parallel: runs on 16k cores
- 138 lines of code

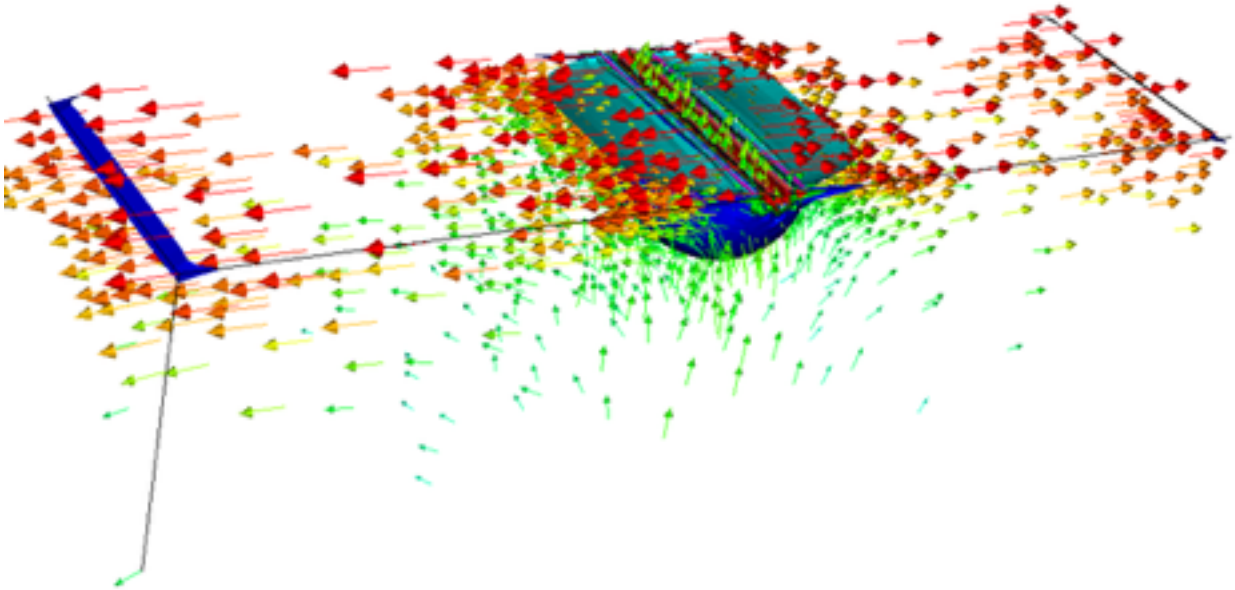




Examples

Step-22:

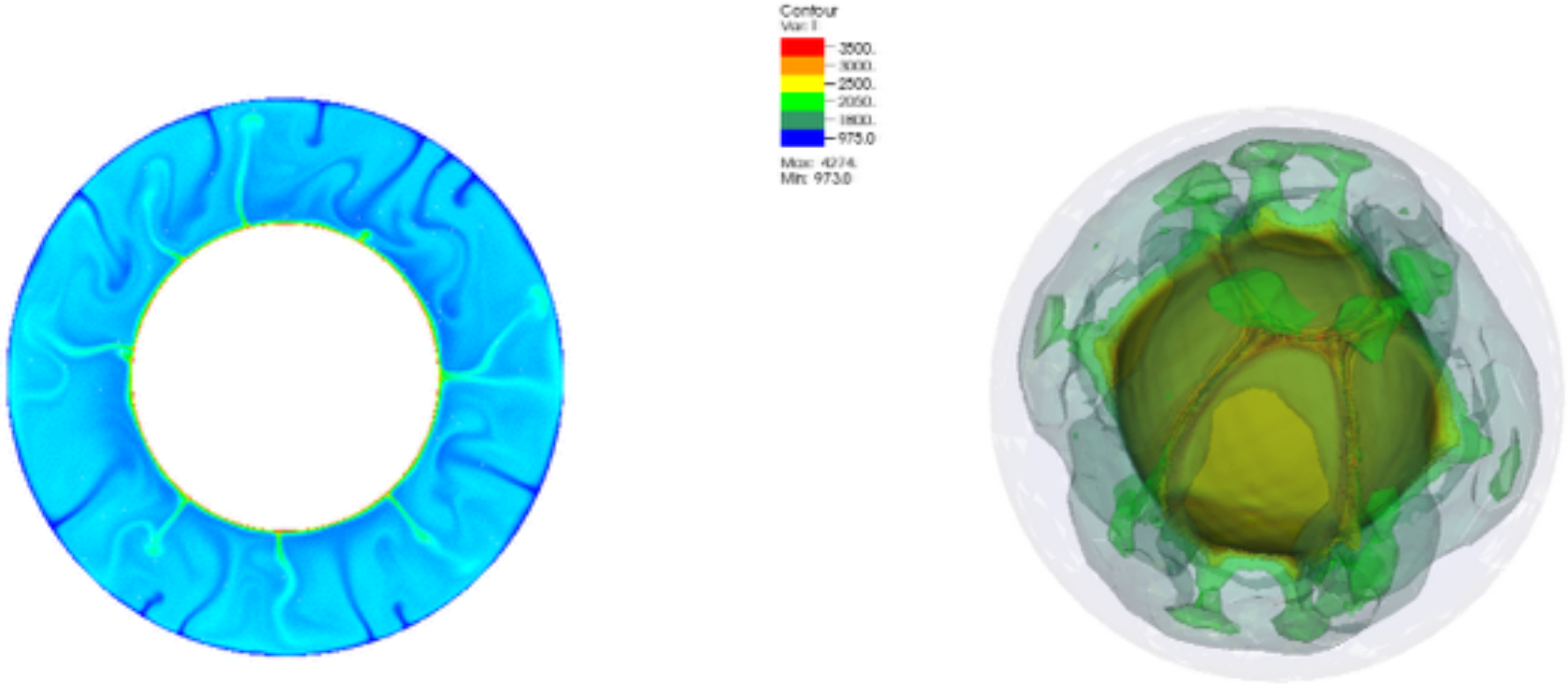
- Stokes equations, “interesting” boundary conditions, 2d/3d
- Adaptive mesh refinement, advanced solvers
- 206 lines of code



Examples

Step-31/32:

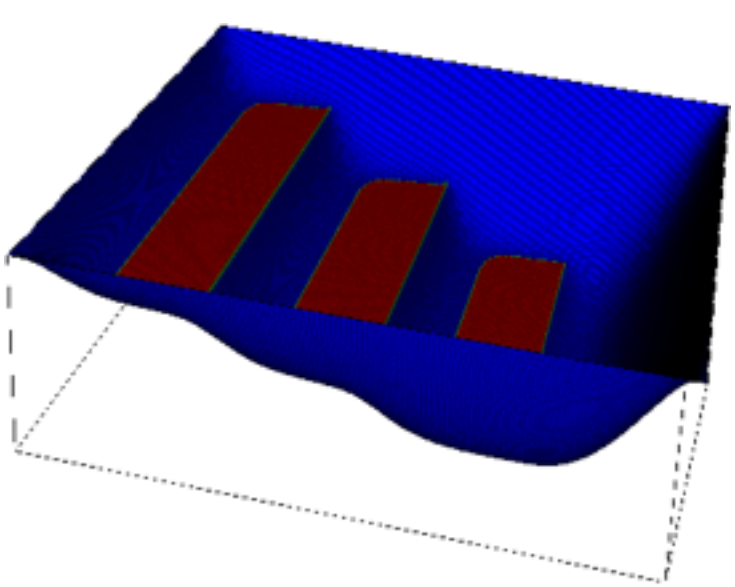
- Boussinesq equations, realistic material models, 2d/3d
- Adaptive mesh refinement, advanced solvers, parallel
- 864 lines of code

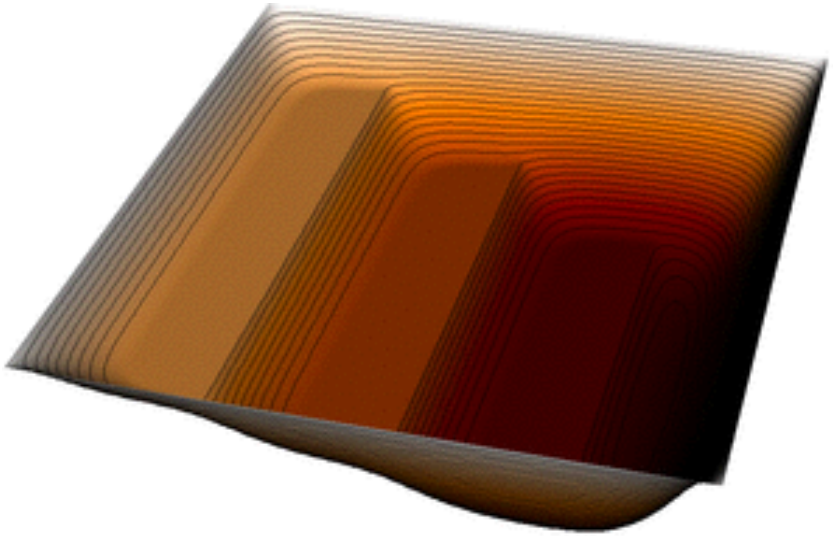


Examples

Step-41:

- Contact problem: Membrane over an obstacle
- Active set/Newton solver
- 177 lines of code

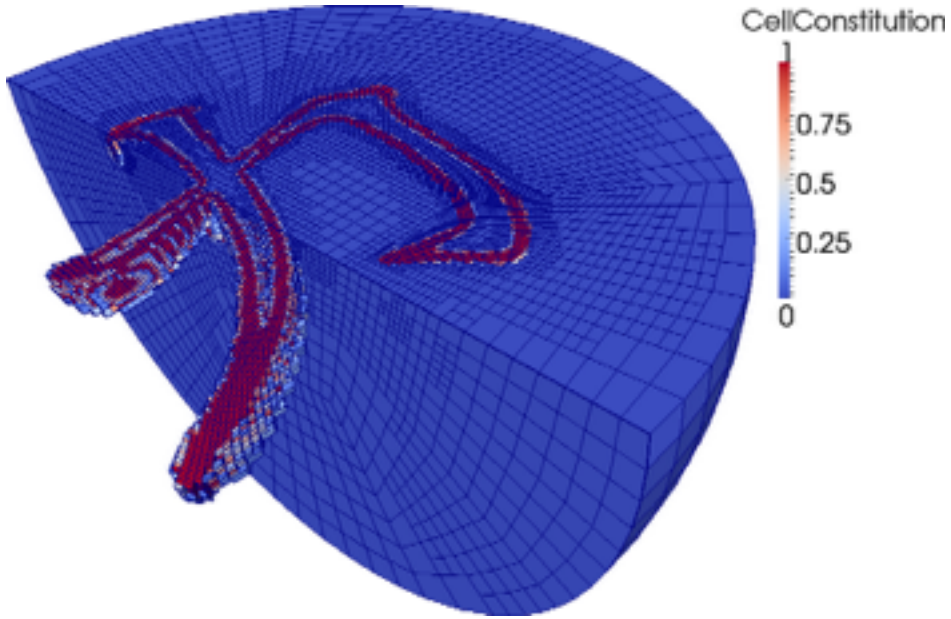


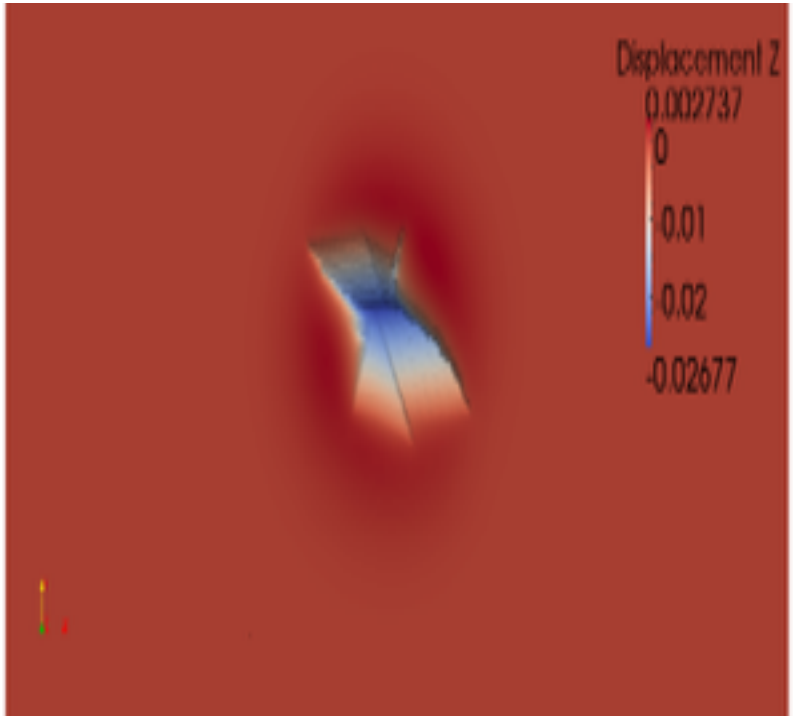


Examples

Step-42:

- Elasto-plastic contact problem
- Active set/Newton solver, multigrid, 3d, parallel
- 586 lines of code





What can you expect?

Experience:

It is realistic for a student developing numerical methods to have a code at the end of a PhD time that:

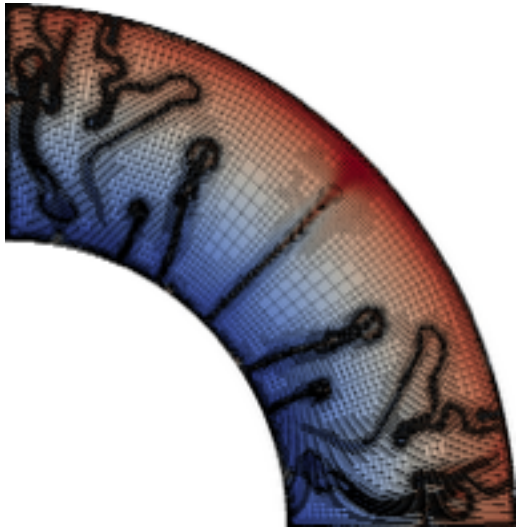
- Works in 2d and 3d
- On complex geometries
- Uses higher order finite element methods
- Uses multigrid solvers or preconditioners
- Solves a nonlinear, time dependent problem

Doing this from scratch would take 10+ years.

Examples

There are also large applications (not part of deal.II):

- *Aspect:* Advanced Solver for Problems in Earth Convection
  – ~50,000 lines of code
  – Open source: http://aspect.dealii.org/



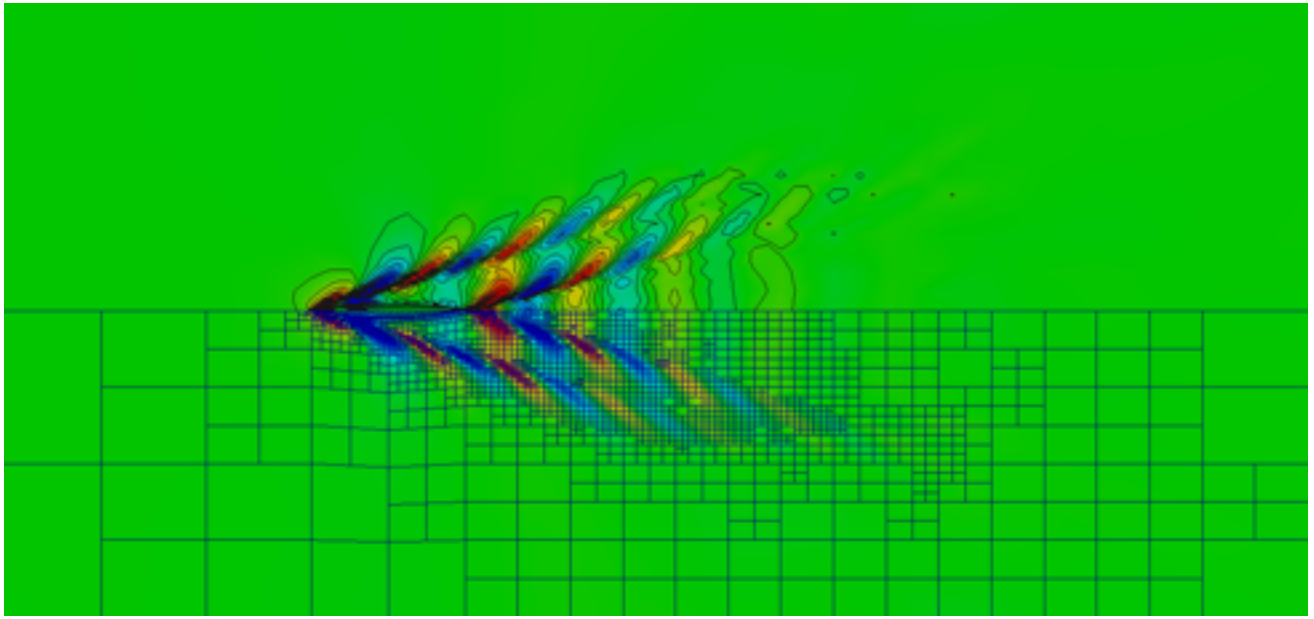
- *OpenFCST:* A fuel cell simulation package
  – Supported by an industrial consortium
  – Open source: http://www.openfcst.org/

- *WaveBEM*: A nonlinear solver for ship-wave interaction

– Supported by a mixed consortium (OpenViewSHIP)

– ~80,000 lines of code

WaveBEM



WaveBEM:

Nonlinear solver for ship-wave interaction using potential flow

(Mola, Heltai, Giuliani, DeSimone, @ SISSA)

WaveBEM – Approaches

Among the mathematical techniques we use are:

- Higher order time stepping schemes (SUNDIALS)
- Direct interface with CAD data structures (OpenCASCADE)
- Fully adaptive, dynamically changing 3d meshes (deal.II)
- Newton's method + Line search for the nonlinearity (TRILINOS)
- Parallelization using MPI (TRILINOS and PETSC)

To make the code usable by the community:

- Use object-oriented programming
- Make it modular, separate concerns
- Semi-Extensive documentation
- Semi-Extensive and frequent testing

Effects

What this development model means for us:

- We can solve problems that were previously intractable

- Methods developers can demonstrate applicability

- Applications scientists can use state of the art methods

- Our codes become far smaller:
  – less potential for error
  – less need for documentation

  – lower hurdle for “reproducible” research (publishing the code along with the paper)

- More impact/more citations when publishing one's code

Effects

What this development model means for our community:

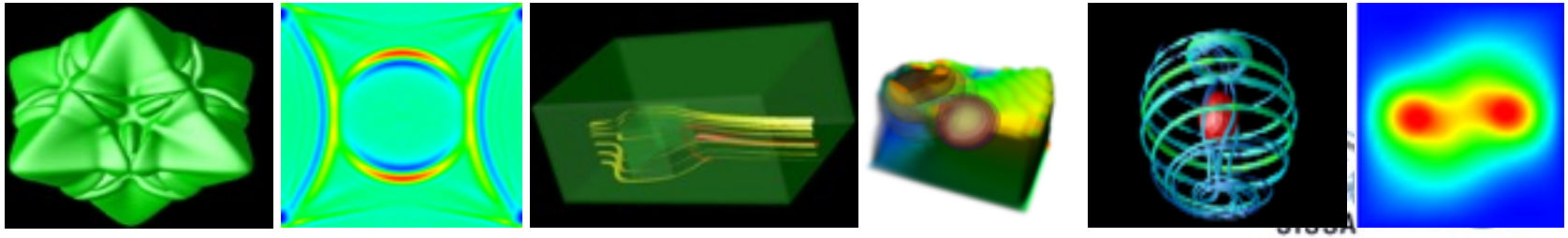
- Faster progress towards “real” applications
- Leveling the playing field – excellent online resources are there for *all*
- Raising the standard in research:
  – can't get 2d papers published any more (almost true…)
  – reviewers can require state-of-the-art solvers
  – allows for easier comparison of methods

Conclusions

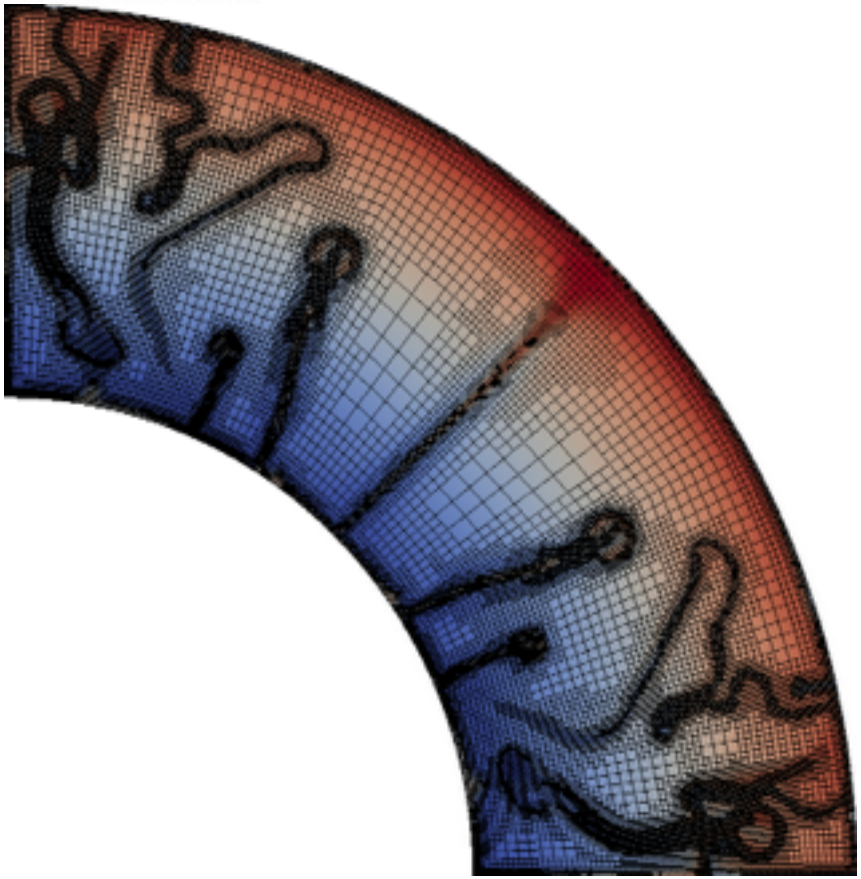
Computational science has spent too much time where everyone writes their own software.

By building on existing, well written and well tested, software packages:

- We build codes *much* faster
- We build better codes
- We can solve more realistic problems



MHPC
Master in High Performance Computing

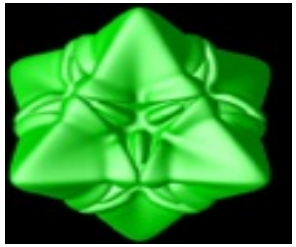
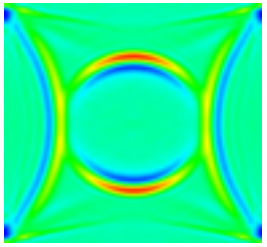
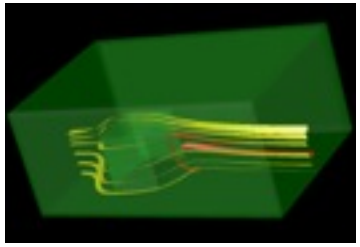
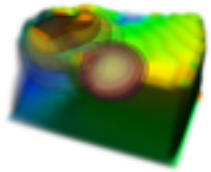
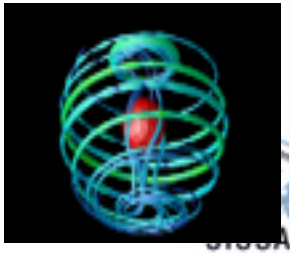
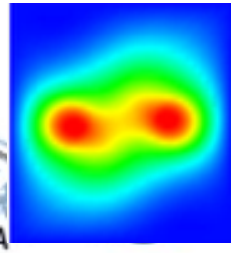


More information:

http://www.dealii.org/

http://aspect.dealii.org/

http://www.openship.it/

Part II:
from serial to massively parallel FEM in deal.II

A working Example…

Brief re-hash of the FEM, using the Poisson equation:

We start with the strong form:

$$\begin{aligned} -\Delta u &= f \quad && \text{in } \Omega \\ u &= 0 \quad && \text{on } \partial\Omega \end{aligned}$$

Basis of FEM!

Brief re-hash of the FEM, using the Poisson equation:

We start with the strong form:

$$-\Delta u = f$$

...and transform this into the weak form by multiplying *from the left* with a test function:

$$(\nabla \phi, \nabla u) = (\phi, f) \quad \forall \phi$$

The solution of this is a function *u(x)* from an infinite-dimensional function space.

MHPC
Master in High Performance Computing

Basis of FEM!

Since computers can't handle objects with infinitely many coefficients, we seek a finite dimensional function of the form

$$u_h = \sum_{j=1}^{N} U_j \phi_j(x)$$

To determine the *N* coefficients, test with the *N* basis functions:

$$(\nabla \phi_i, \nabla u_h) = (\phi_i, f) \quad \forall i = 1 \ldots N$$

If basis functions are linearly independent, this yields *N* equations for *N* coefficients.
This is called the *Galerkin* method.

Basis of FEM!

Practical question 1: How to define the basis functions?

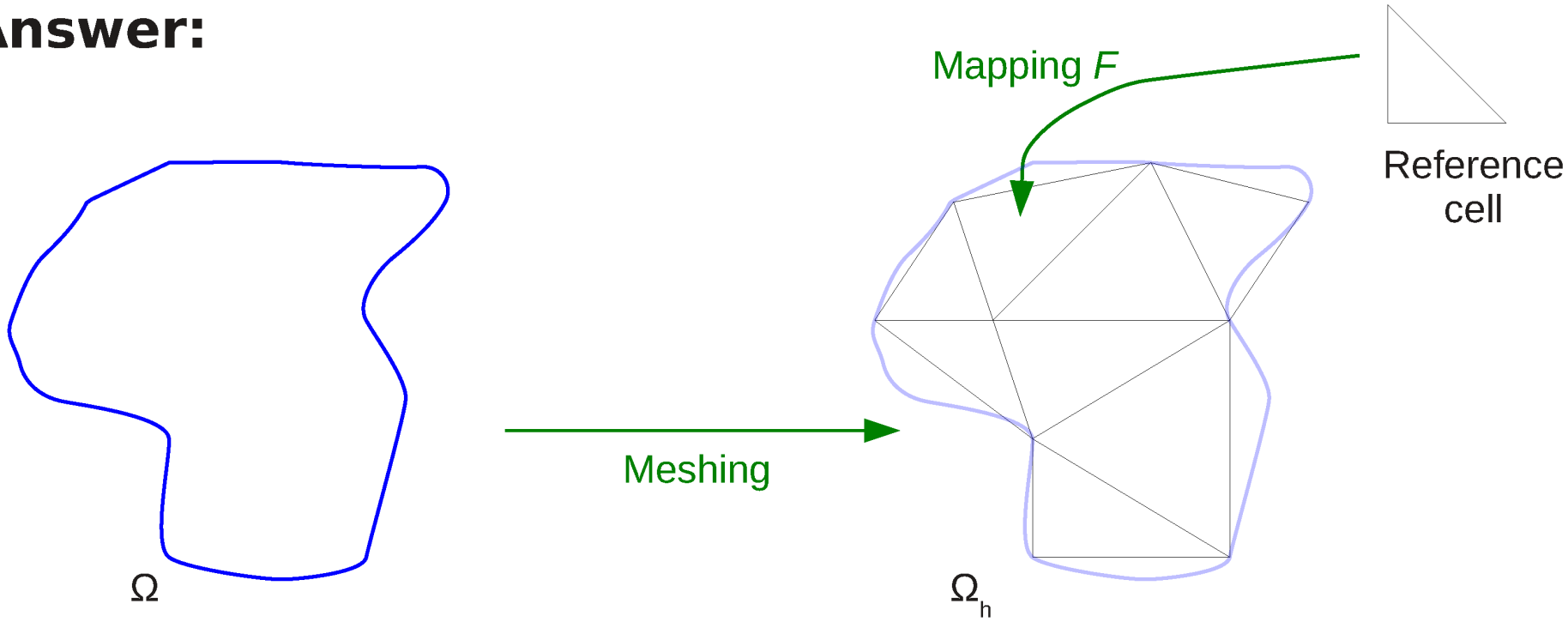
Answer: In the finite element method, this is done using the following concepts:

- Subdivision of the domain into a mesh
- Each cell of the mesh is a mapping of the reference cell
- Definition of basis functions on the reference cell
- Each shape function corresponds to a degree of freedom on the global mesh

Basis of FEM!

Practical question 1: How to define the basis functions?

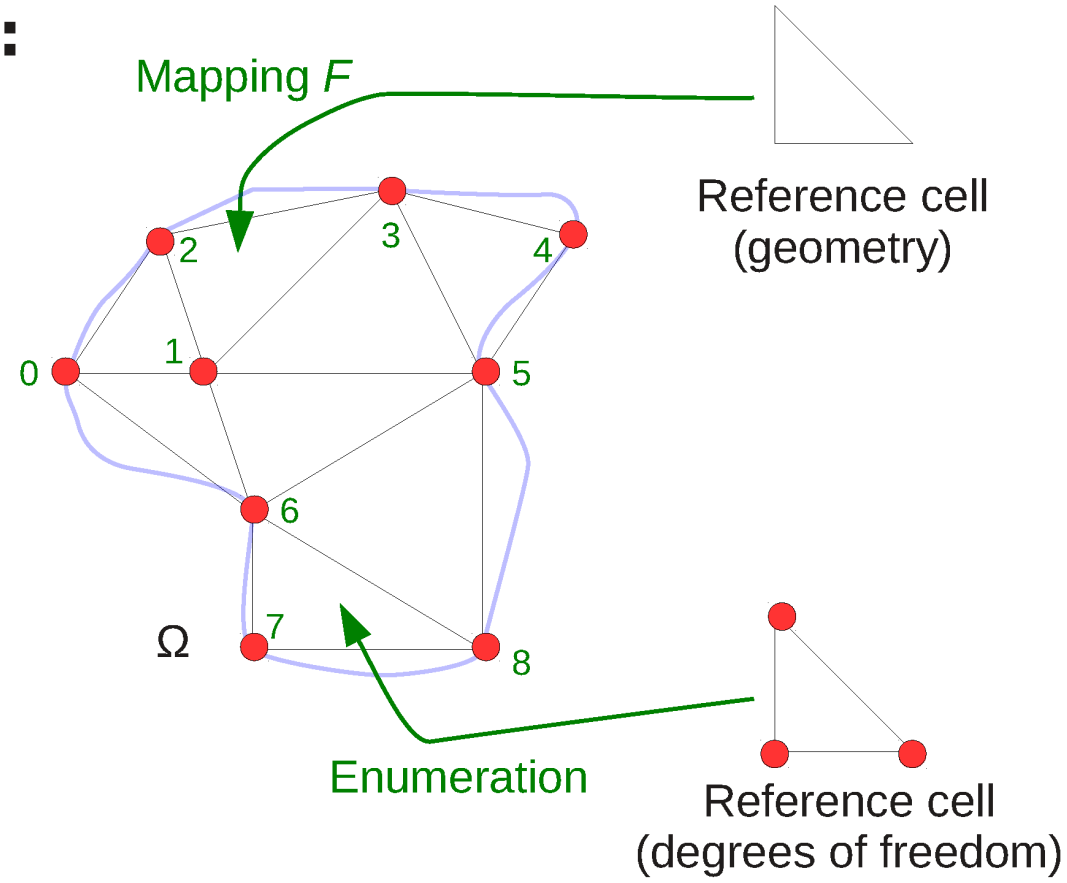
Answer:



Basis of FEM!

Practical question 1: How to define the basis functions?

Answer:



Basis of FEM!

Practical question 1: How to define the basis functions?

Answer: In the finite element method, this is done using the following concepts:

- Subdivision of the domain into a mesh
- Each cell of the mesh is a mapping of the reference cell
- Definition of basis functions on the reference cell
- Each shape function corresponds to a degree of freedom on the global mesh

Concepts in red will correspond to things we need to implement in software, explicitly or implicitly.

Basis of FEM!

Given the definition $u_h = \sum_{j=1}^{N} U_j \phi_j(x)$, we can expand the bilinear form

$$(\nabla \phi_i, \nabla u_h) = (\phi_i, f) \quad \forall i = 1 \ldots N$$

to obtain:

$$\sum_{j=1}^{N} (\nabla \phi_i, \nabla \phi_j) U_j = (\phi_i, f) \quad \forall i = 1 \ldots N$$

This is a linear system

$$A U = F$$

with

$$A_{ij} = (\nabla \phi_i, \nabla \phi_j) \qquad F_i = (\phi_i, f)$$

Basis of FEM!

Practical question 2: How to compute

$$A_{ij}=(\nabla\phi_i,\nabla\phi_j) \qquad F_i=(\phi_i,f)$$

Answer: By mapping back to the reference cell...

$$\begin{aligned} A_{ij} &= (\nabla\phi_i,\nabla\phi_j) \\ &= \sum_K \int_K \nabla\phi_i(x)\cdot\nabla\phi_j(x) \\ &= \sum_K \int_{\hat{K}} J_K^{-1}(\hat{x})\hat{\nabla}\hat{\phi}_i(\hat{x}) \cdot J_K^{-1}(\hat{x})\hat{\nabla}\hat{\phi}_j(\hat{x}) \ |\det J_K(\hat{x})| \end{aligned}$$

...and quadrature:

$$A_{ij} \approx \sum_K \sum_{q=1}^{Q} J_K^{-1}(\hat{x}_q)\hat{\nabla}\hat{\phi}_i(\hat{x}_q) \cdot J_K^{-1}(\hat{x}_q)\hat{\nabla}\hat{\phi}_j(\hat{x}_q) \ \underbrace{|\det J(\hat{x}_q)| \ w_q}_{=:\mathrm{JxW}}$$

Similarly for the right hand side *F*.

Basis of FEM!

Practical question 3: How to store the matrix and vectors of the linear system

$$AU = F$$

Answers:

- *A* is sparse, so store it in compressed row format
- *U,F* are just vectors, store them as arrays
- Implement efficient algorithms on them, e.g. matrix-vector products, preconditioners, etc.
- For large-scale computations, data structures and algorithms must be parallel

Basis of FEM!

Practical question 4: How to solve the linear system

$$AU = F$$

Answers: In practical computations, we need a variety of

- Direct solvers
- Iterative solvers
- Parallel solvers

Basis of FEM!

Practical question 5: What to do with the solution of the linear system

$$AU = F$$

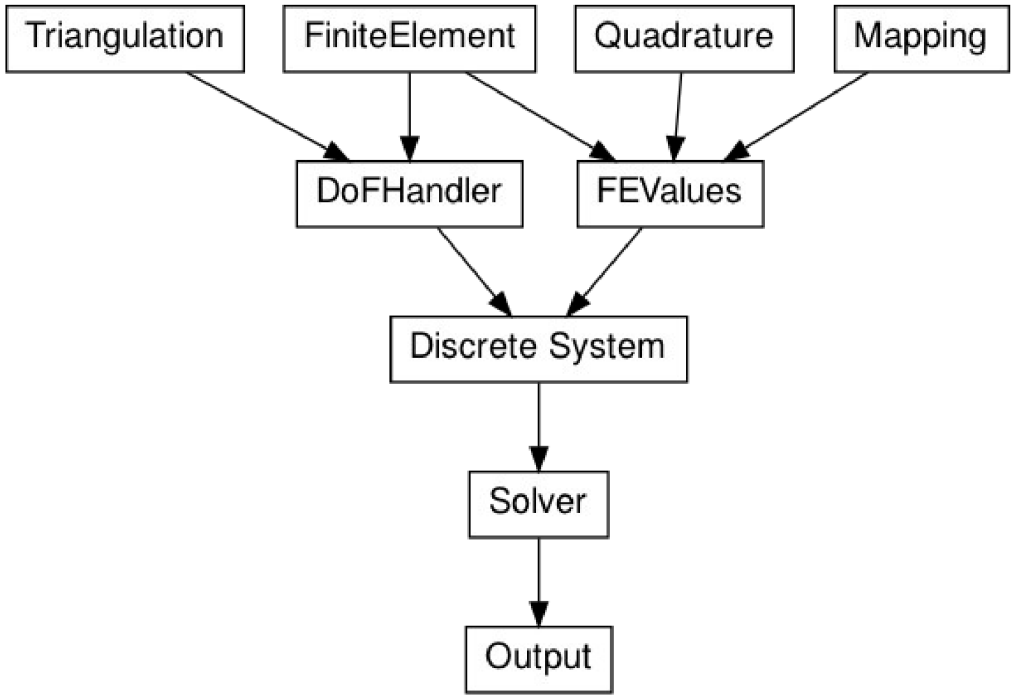
Answers: The goal is not to solve the linear system, but to do something with its solution:

- Visualize
- Evaluate for quantities of interest
- Estimate the error

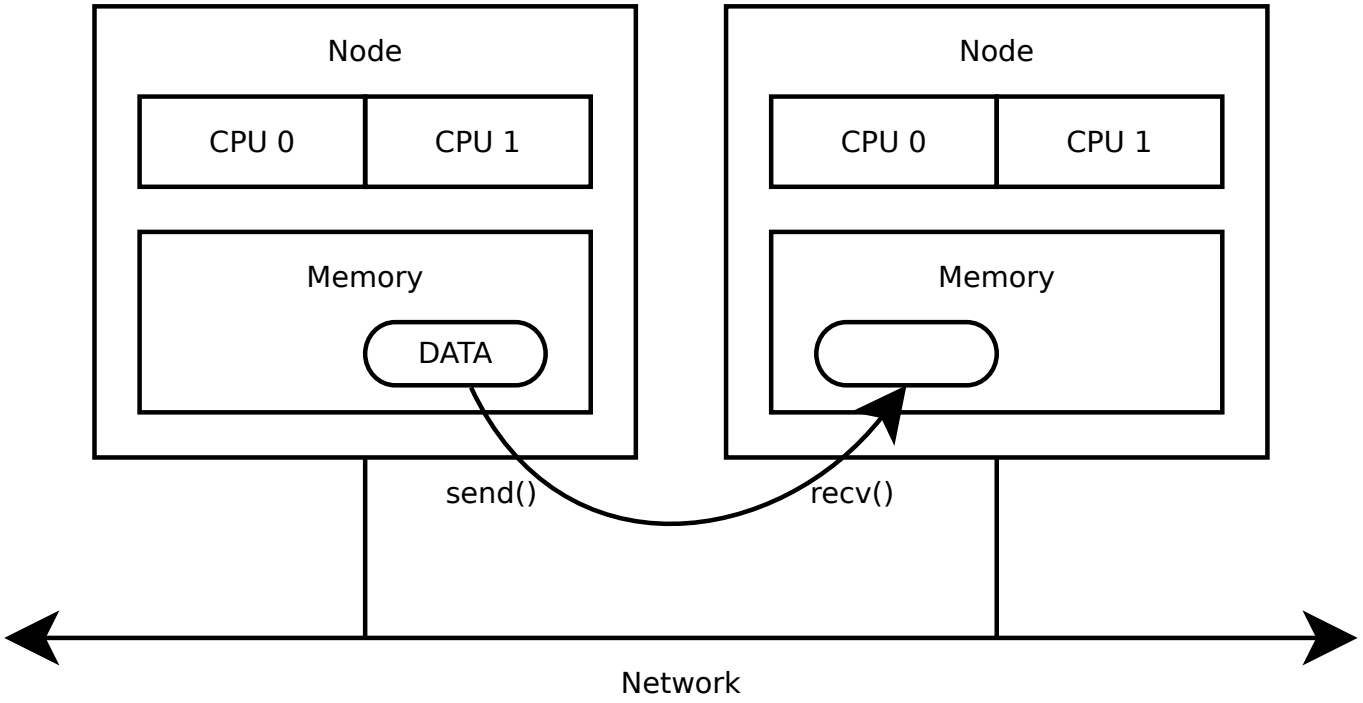
These steps are often called *postprocessing the solution*.

Basis of FEM!

Together, the concepts we have identified lead to the following components that all appear (explicitly or implicitly) in finite element codes:



Parallel computing model: MPI



General Considerations

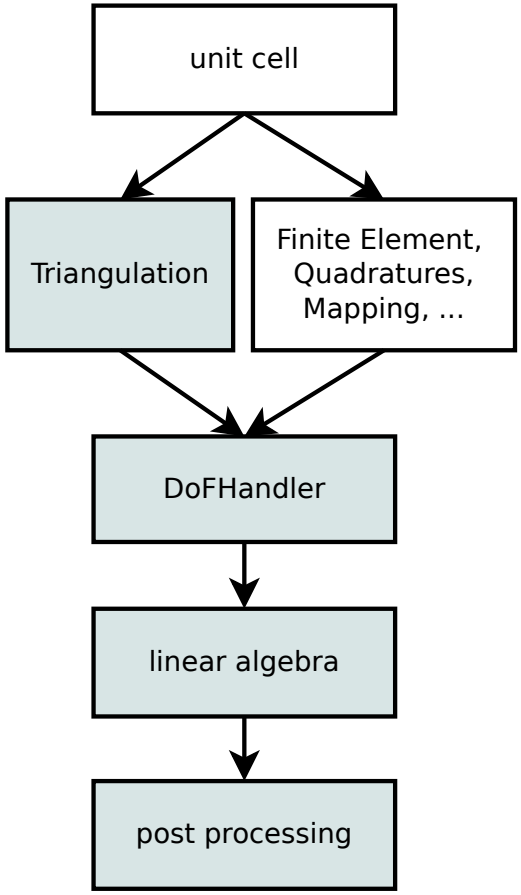
- Goal: get the solution faster!
- If FEM with <500.000 dofs, and 2d, use direct solver!

- If you need more, then you have to **SPLIT** the work
  - **Distributed data** storage everywhere
    - need special data structures
  - **Efficient algorithms**
    - not depending on total problem size
  - **“Localize” and “hide” communication**
    - point-to-point communication, nonblocking sends and receives

Data Structures

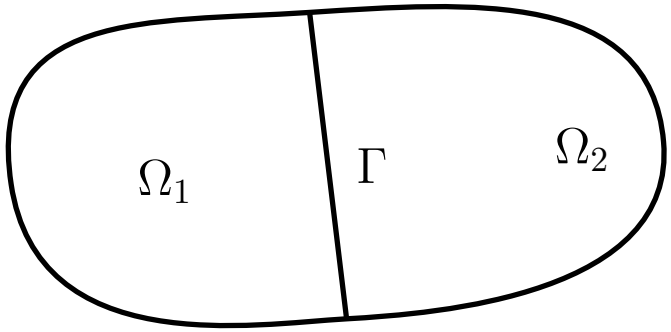
Needs to be parallelized:

1. Triangulation (mesh with associated data)
   — hard: distributed storage, new algorithms
2. DoFHandler (manages degrees of freedom)
   — hard: find global numbering of DoFs
3. Linear Algebra (matrices, vectors, solvers)
   — use existing library
4. Postprocessing (error estimation, solution transfer, output, . . .)
   — do work on local mesh, communicate



How to Parallelize?

Option 1: Domain Decomposition

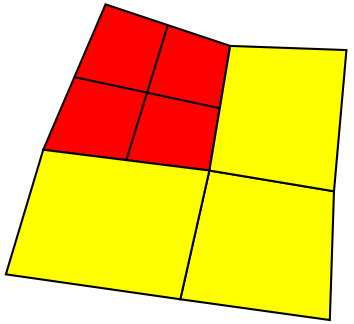


- Split up problem on PDE level
- Solve subproblems independently
- Converges against global solution
- Problems:
  - Boundary conditions are problem dependent:
    $\rightsquigarrow$ sometimes difficult!
    $\rightsquigarrow$ no black box approach!
  - Without coarse grid solver:
    condition number grows with # subdomains
    $\rightsquigarrow$ no linear scaling with number of CPUs!

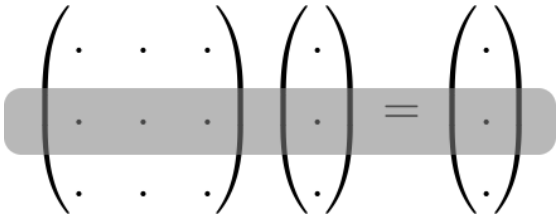
How to Parallelize?

Option 2: Algebraic Splitting

- Split up mesh between processors:



- Assemble logically global linear system (distributed storage):

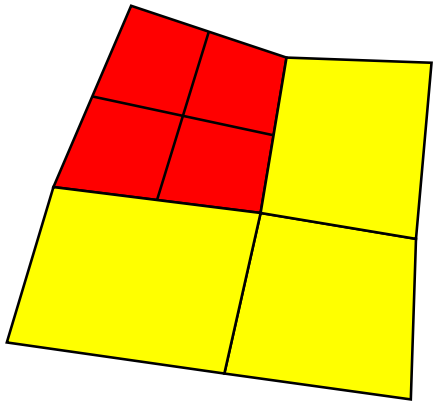


- Solve using iterative linear solvers in parallel
- Advantages:
  - Looks like serial program to the user
  - Linear scaling possible (with good preconditioner)

Partitioning

Optimal partitioning (coloring of cells):

- same size per region
  - $\rightsquigarrow$ even distribution of work
- minimize interface between region
  - $\rightsquigarrow$ reduce communication



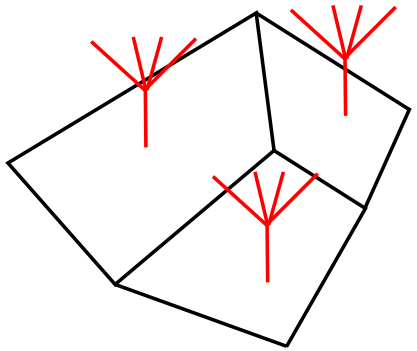
Optimal partitioning is an NP-hard graph partitioning problem.

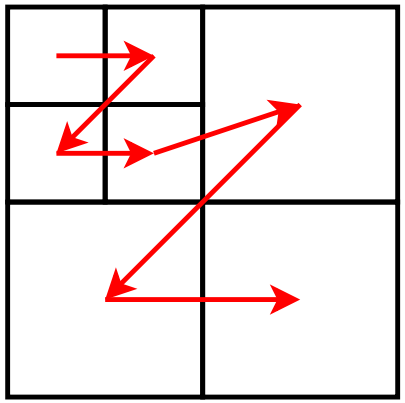
- Typically done: heuristics (existing tools: METIS)
- Problem: worse than linear runtime
- Large graphs: several minutes, memory restrictions

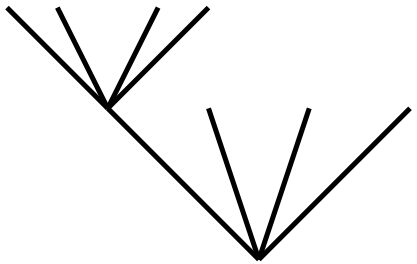
$\rightsquigarrow$ Alternative: avoid graph partitioning

Partitioning with “Space filling curves”

- *p4est* library: parallel quad-/octrees
- Store refinement flags from a base mesh
- Based on space-filling curves
- Very good scalability



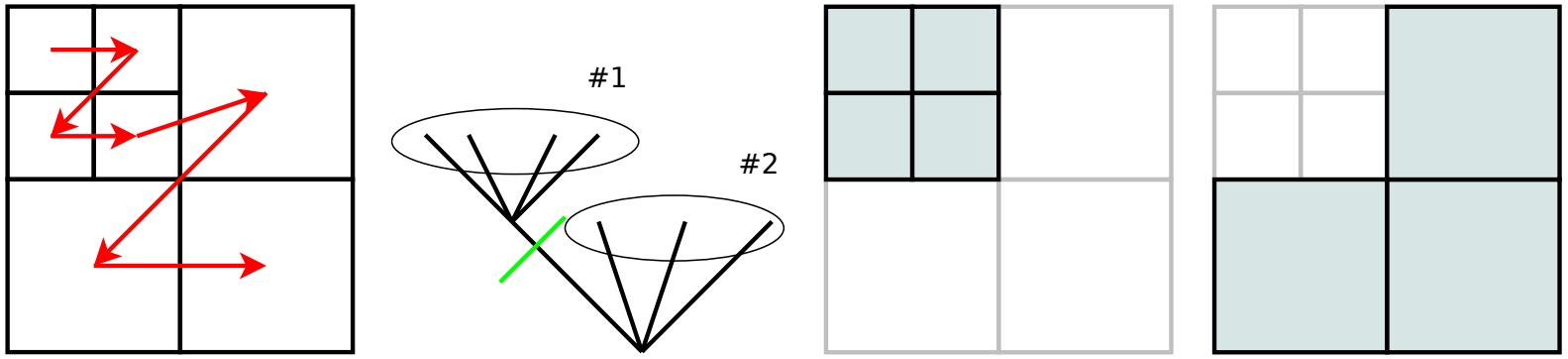




Burstedde, Wilcox, and Ghattas.
p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees.
SIAM J. Sci. Comput., 33 no. 3 (2011), pages 1103-1133.

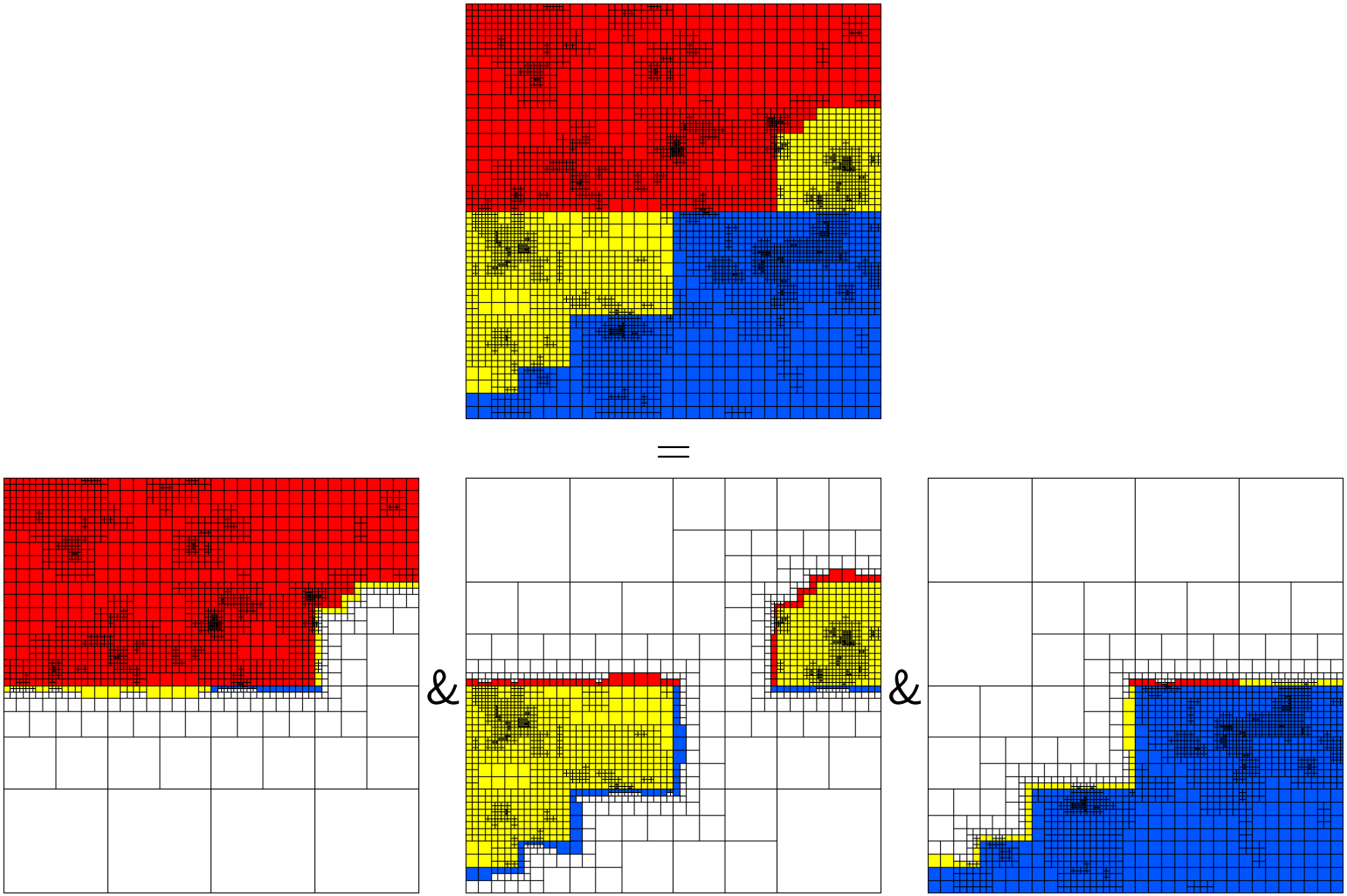
Triangulation

- Partitioning is cheap and simple:



- Then: take *p4est* refinement information
- Recreate rich *deal.II* Triangulation only for local cells (stores coordinates, connectivity, faces, materials, . . .)
- How? recursive queries to *p4est*
- Also create ghost layer (one layer of cells around own ones)

Example (color by CPU ID)



MHPC
Master in High Performance Computing

What's needed?

How to use?

- Replace `Triangulation` by `parallel::distributed::Triangulation`
- Continue to load or create meshes as usual
- Adapt with `GridRefinement::refine_and_coarsen*` and `tr.execute_coarsening_and_refinement()`, etc.
- You can only look at own cells and ghost cells: `cell->is_locally_owned()`, `cell->is_ghost()`, or `cell->is_artificial()`
- Of course: dealing with DoFs and linear algebra changes!

SISSA ICTP

MHPC
Master in High Performance Computing

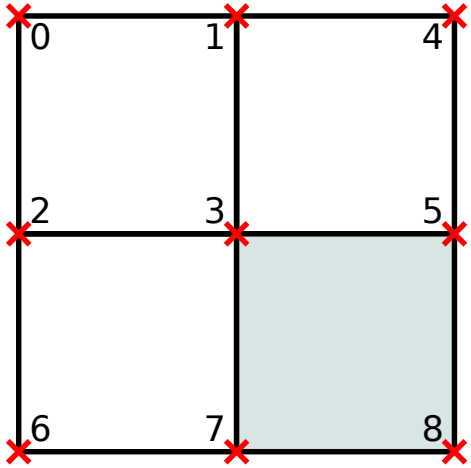
What's needed?

	serial mesh	dynamic parallel mesh	static parallel mesh
name	Triangulation	parallel::distributed ::Triangulation	(just an idea)
duplicated	everything	coarse mesh	nothing
partitioning	METIS	p4est: fast, scalable	offline, (PAR)METIS?
part. quality	good	okay	better?
hp?	yes	(planned)	yes?
geom. MG?	yes	in progress	?
Aniso. ref.?	yes	no	(offline only)
Periodicity	yes	in progress	?
Scalability	100 cores	16k+ cores	?

SISSA ICTP

Sketch...

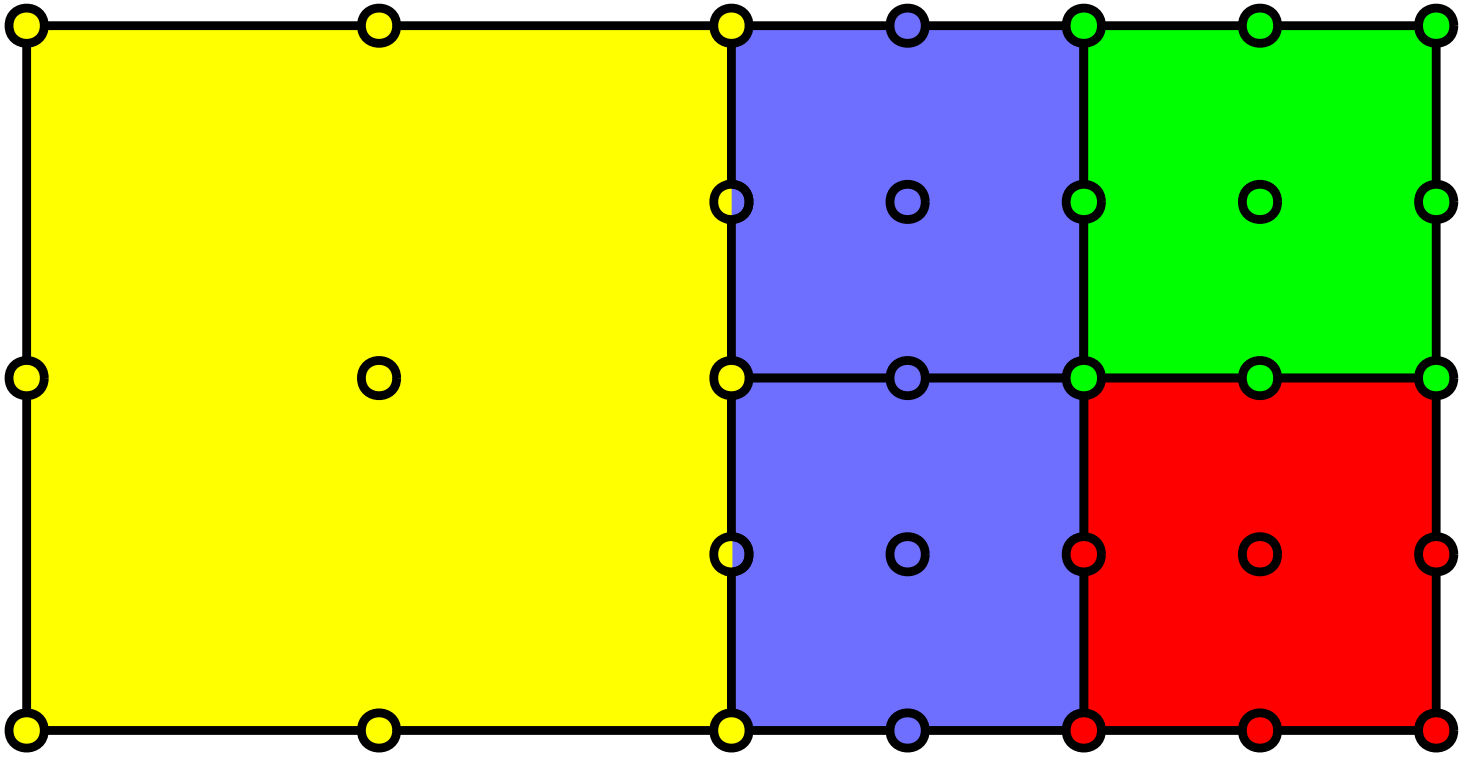
- Create global numbering for all DoFs
- Reason: identify shared ones
- Problem: no knowledge about the whole mesh



Sketch:

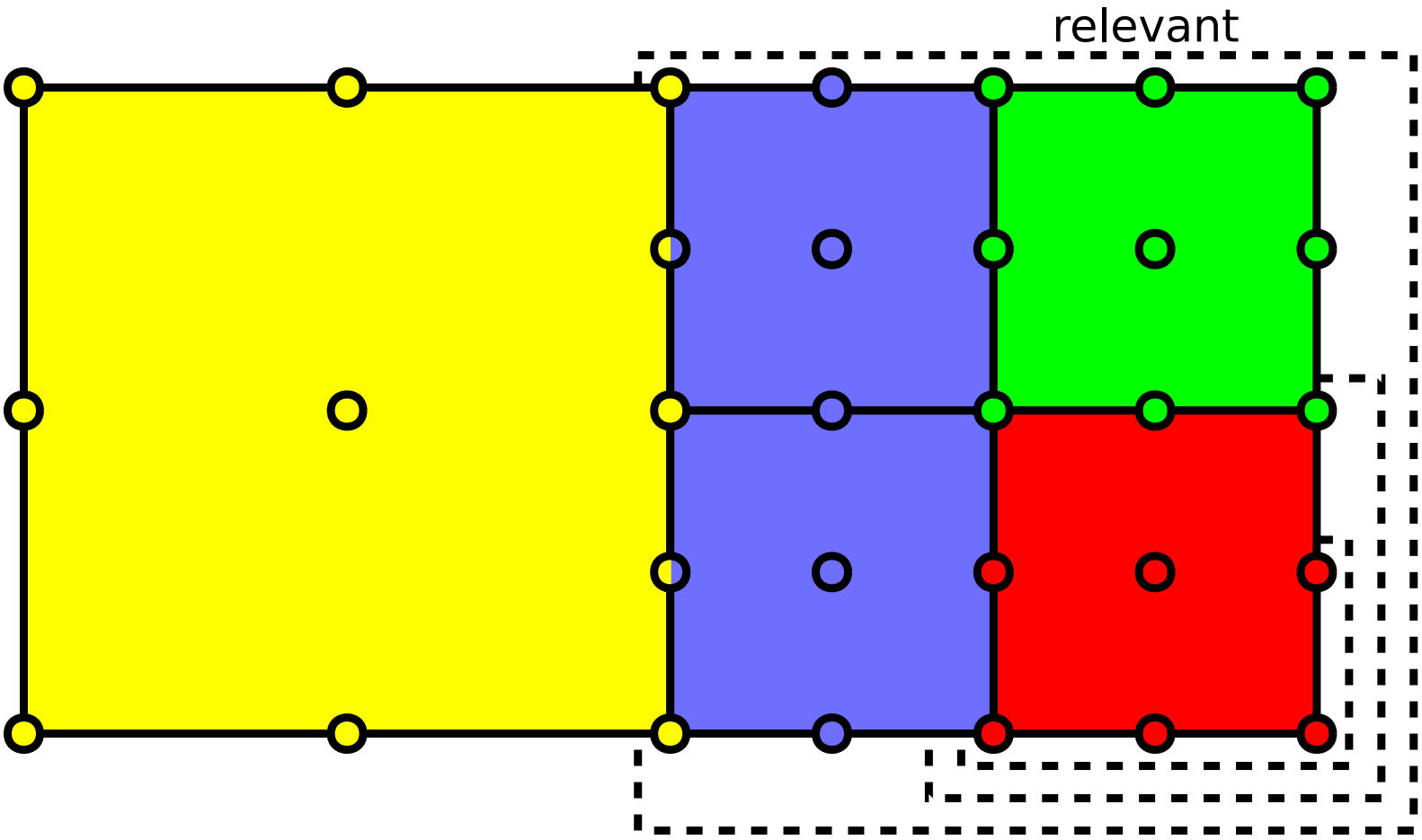
1. Decide on ownership of DoFs on interface (no communication!)
2. Enumerate locally (only own DoFs)
3. Shift indices to make them globally unique (only communicate local quantities)
4. Exchange indices to ghost neighbors

Sketch...



- Example: Q2 element and ownership of DoFs
- What might red CPU be interested in?

Sketch...



(perspective of the red CPU)

MHPC
Master in High Performance Computing

Sketch...

- Each CPU has sets:
  - `owned`: we store vector and matrix entries of these rows
  - `active`: we need those for assembling, computing integrals, output, etc.
  - `relevant`: error estimation
- These set are subsets of $\{0, \ldots,$`n_global_dofs`$\}$
- Represented by objects of type `IndexSet`
- How to get? `DoFHandler::locally_owned_dofs()`, `DoFTools::extract_locally_relevant_dofs()`, `DoFHandler::locally_owned_dofs_per_processor()`, ...

SISSA ICTP

MHPC
Master in High Performance Computing

Sketch...

- reading from owned rows only (for both vectors and matrices)
- writing allowed everywhere (more about `compress` later)
- what if you need to read others?
- Never copy a whole vector to each machine!
- instead: ghosted vectors

SISSA ICTP

Sketch...

- read-only
- create using
  `Vector(IndexSet owned, IndexSet ghost, MPI_COMM)`
  where ghost is `relevant` or `active`
- copy values into it by using operator=(Vector)
- then just read entries you need

Trilinos VS PETSc

What should I use?

- Similar features and performance
- Pro Trilinos: more development, some more features (automatic differentation, . . .), cooperation with deal.II
- Pro PETSc: stable, easier to compile on older clusters
- But: being flexible would be better! – "why not both?"
  - you can! Example: new `step-40`
  - can switch at compile time
  - need #ifdef in a few places (different solver parameters TrilinosML vs BoomerAMG)
  - some limitations, somewhat work in progress

Trilinos VS PETSc

```
#include <deal.II/lac/generic_linear_algebra.h>
#define USE_PETSC_LA // uncomment this to run with Trilinos

namespace LA
{
#ifdef USE_PETSC_LA
  using namespace dealii::LinearAlgebraPETSc;
#else
  using namespace dealii::LinearAlgebraTrilinos;
#endif
}

// ...
    LA::MPI::SparseMatrix system_matrix;
    LA::MPI::Vector solution;

// ...
    LA::SolverCG solver(solver_control, mpi_communicator);
    LA::MPI::PreconditionAMG preconditioner;

    LA::MPI::PreconditionAMG::AdditionalData data;

#ifdef USE_PETSC_LA
    data.symmetric_operator = true;
#else
    //trilinos defaults are good
#endif
    preconditioner.initialize(system_matrix, data);

// ...
```

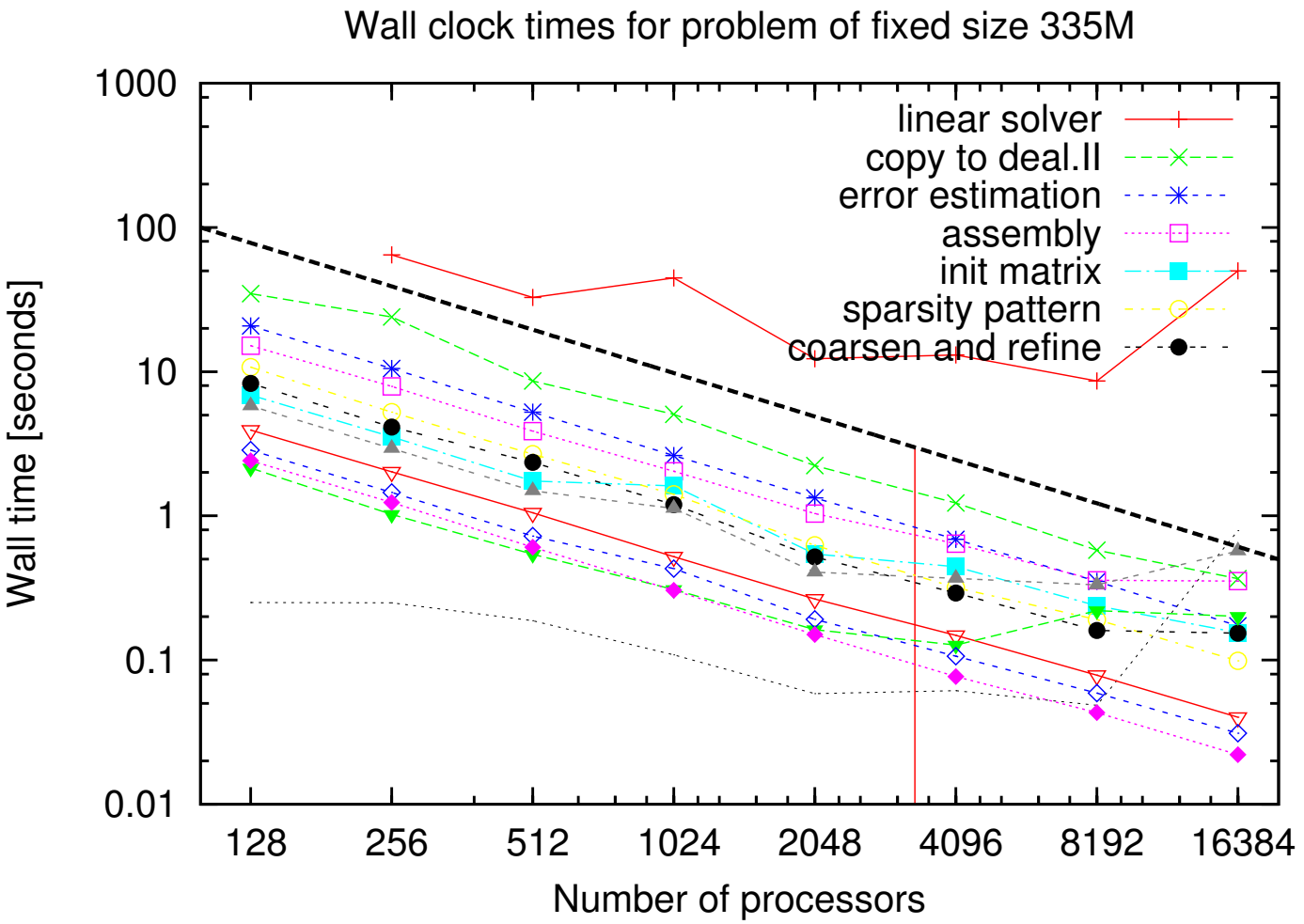
MHPC
Master in High Performance Computing

Solvers

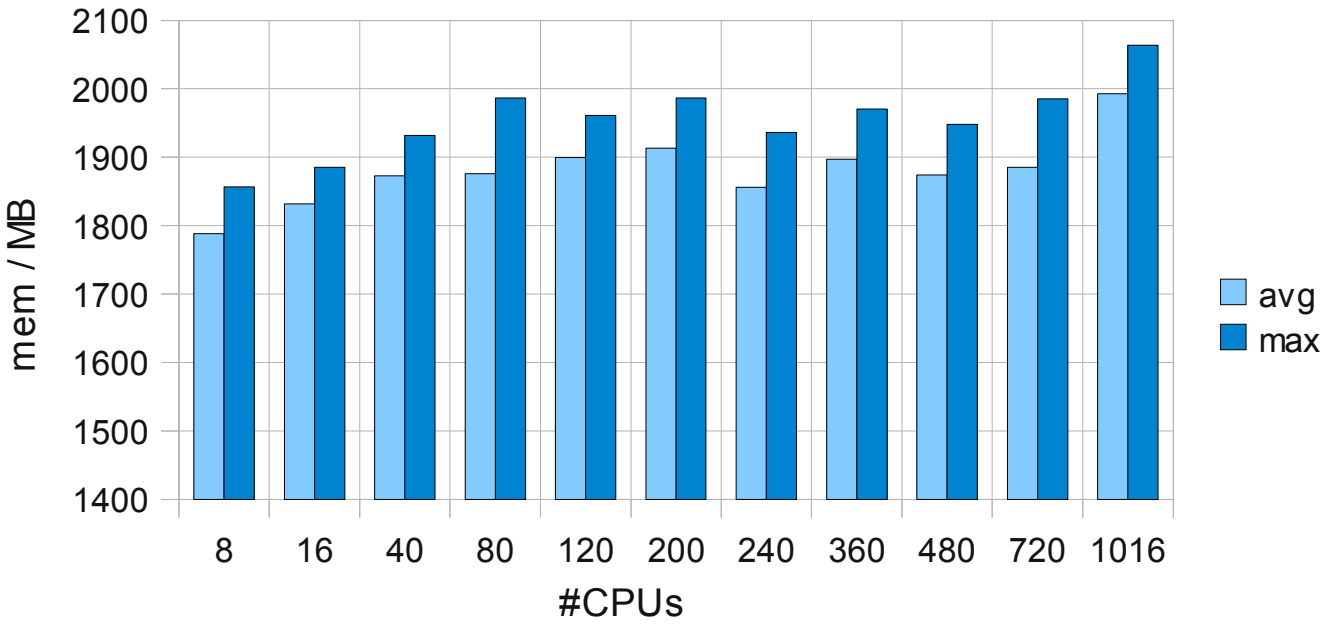
- Iterative solvers only need Mat-Vec products and scalar products
  $\rightsquigarrow$ equivalent to serial code
- Can use templated deal.II solvers like GMRES!
- Better: use tuned parallel iterative solvers that hide/minimize communication
- Preconditioners: more work, just operating on local blocks not enough

SISSA ICTP

Strong Scaling: 2d Poisson



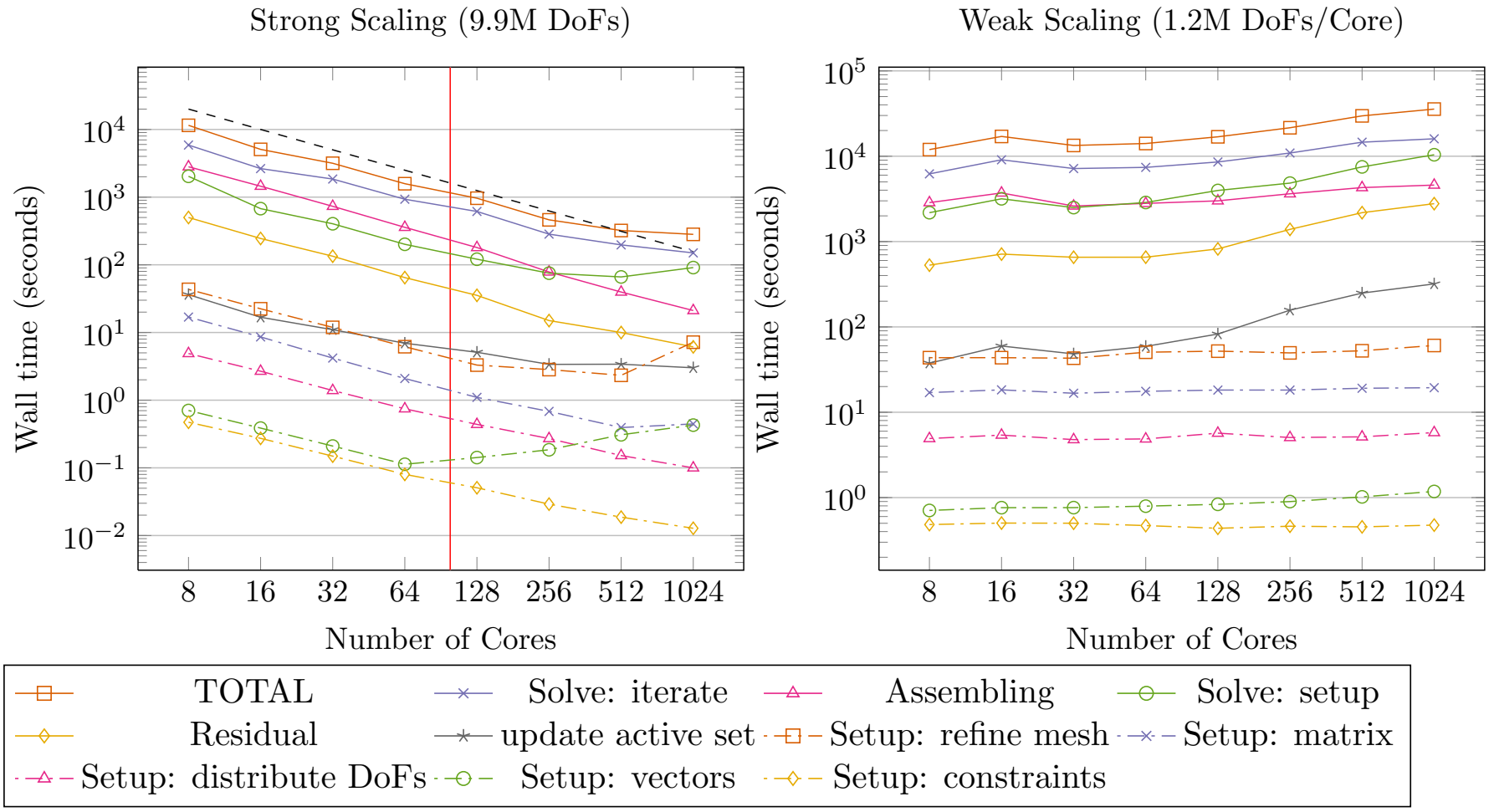
Memory Consumption



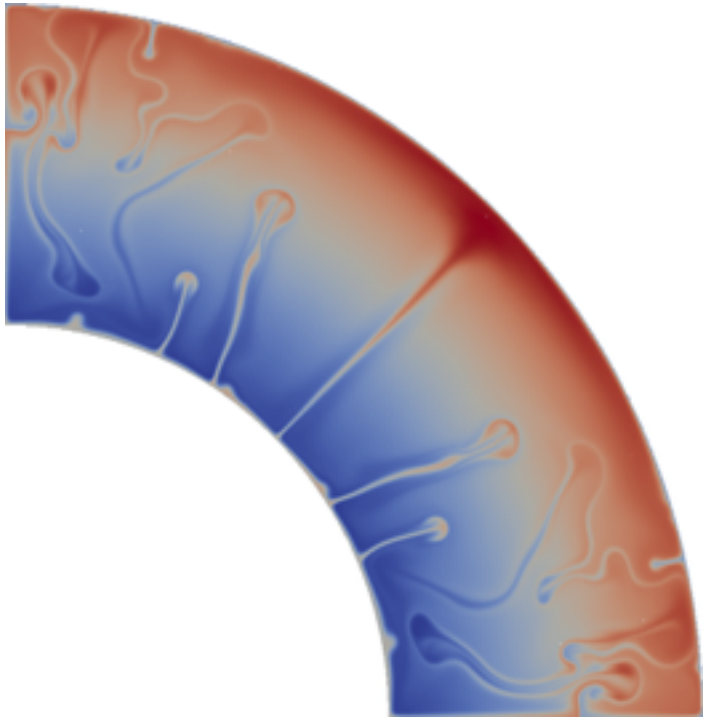
average and maximum memory consumption (VmPeak)
3D, weak scalability from 8 to 1000 processors with about 500.000 DoFs per processor (4 million up to 500 million total)

$\rightsquigarrow$ **Constant memory usage with increasing # CPUs & problem size**

Step 42


Strong Scaling (9.9M DoFs)
Wall time (seconds)
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
8
16
32
64
128
256
512
1024
Number of Cores
Weak Scaling (1.2M DoFs/Core)
Wall time (seconds)
10^5
10^4
10^3
10^2
10^1
10^0
8
16
32
64
128
256
512
1024
Number of Cores
TOTAL
Solve: iterate
Assembling
Solve: setup
Residual
update active set
Setup: refine mesh
Setup: matrix
Setup: distribute DoFs
Setup: vectors
Setup: constraints

A Real life Example: ASPECT



Aspect:
Advanced Solver for Problems in Earth ConvecTion

(Bangerth, Heister, and many others around the world; funding by CIG and NSF)

ASPECT – Approaches

Among the mathematical techniques we use are:

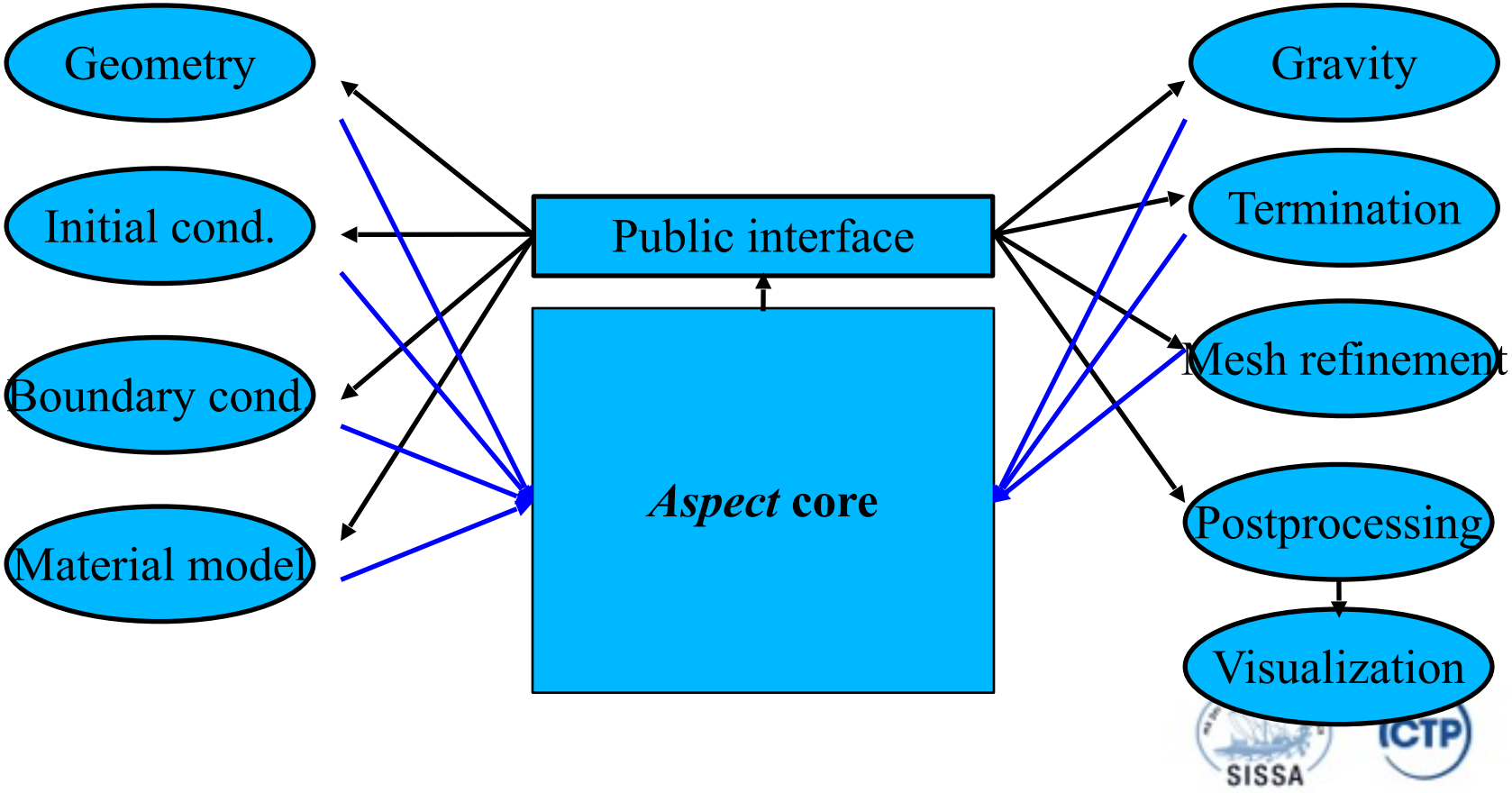
- Higher order time stepping schemes
- Higher order finite elements
- Fully adaptive, dynamically changing 3d meshes
- Newton's method for the nonlinearity
- Silvester/Wathen-style block preconditioners with FGMRES
- Algebraic multigrid for the elliptic part
- Parallelization using MPI, threads, and tasks

To make the code usable by the community:

- Use object-oriented programming
- Make it modular, separate concerns
- Extensive documentation
- Extensive and frequent testing

ASPECT – Approaches

***Aspect* is very modular:** It is extended by a number of isolated "plugin" sub-systems:



ASPECT – Approaches

How hard is this in practice:

- Core simulator:
  – ~5,000 lines of code (1,100 semicolons)
- Runtime parameters, checkpoint/restart, etc:
  – ~1,800 lines of code (350 semicolons)

- Problem statement (geometry, materials, boundary and initial conditions, postprocessing, etc):
  – ~43,000 lines of code (8,600 semicolons)

ASPECT – Approaches

How hard is this in practice:

- Core simulator:
  – ~5,000 lines of code (1,100 semicolons) – mostly stable
- Runtime parameters, checkpoint/restart, etc:
  – ~1,800 lines of code (350 semicolons) – mostly stable

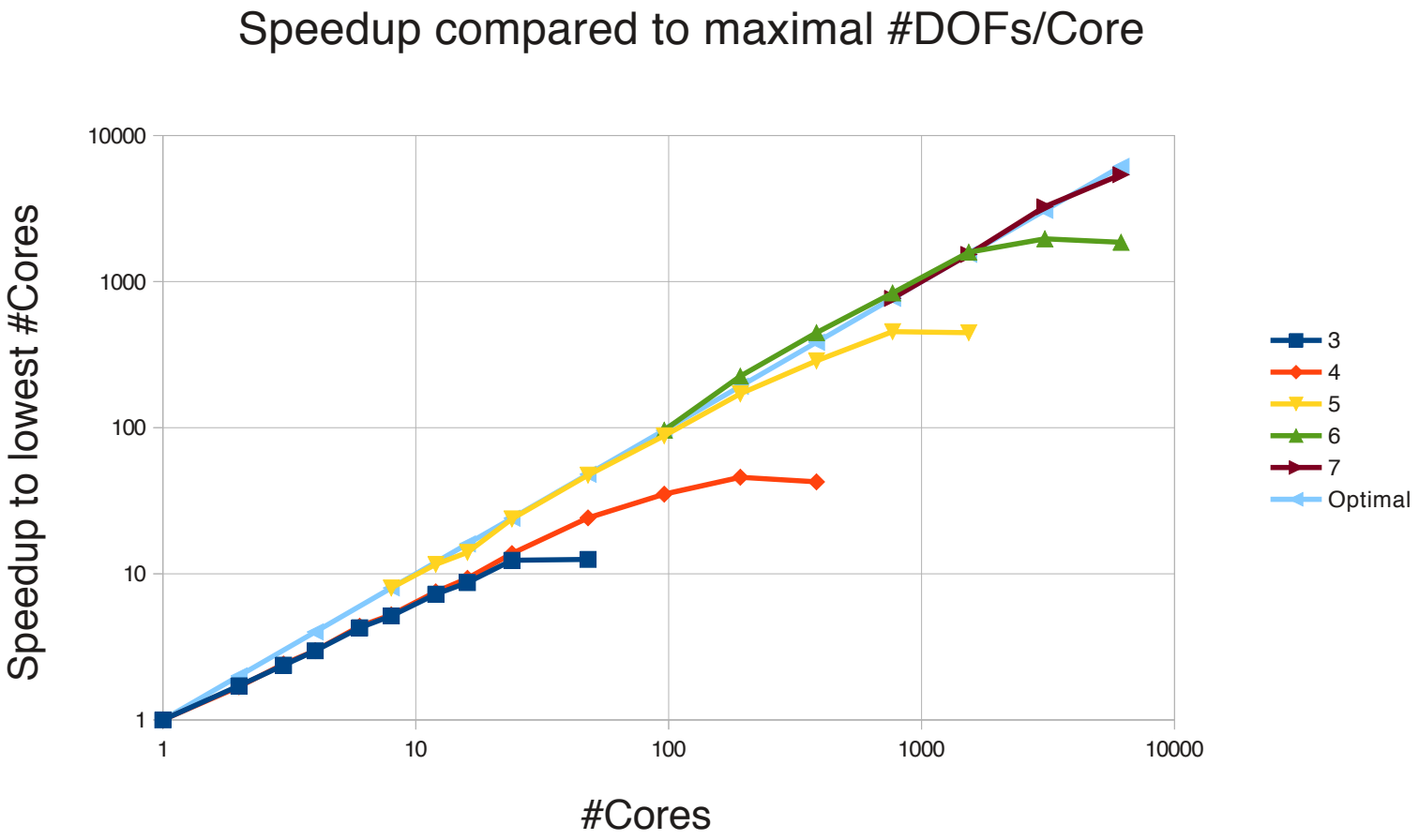
- Problem statement (geometry, materials, boundary and initial conditions, postprocessing, etc):
  – ~43,000 lines of code (8,600 semicolons) – growing

Conclusions:

- Using advanced numerical methods does not lead to prohibitive code growth
- Developers can focus on application specifics
- Most parts are self-contained and accessible to “newbies”

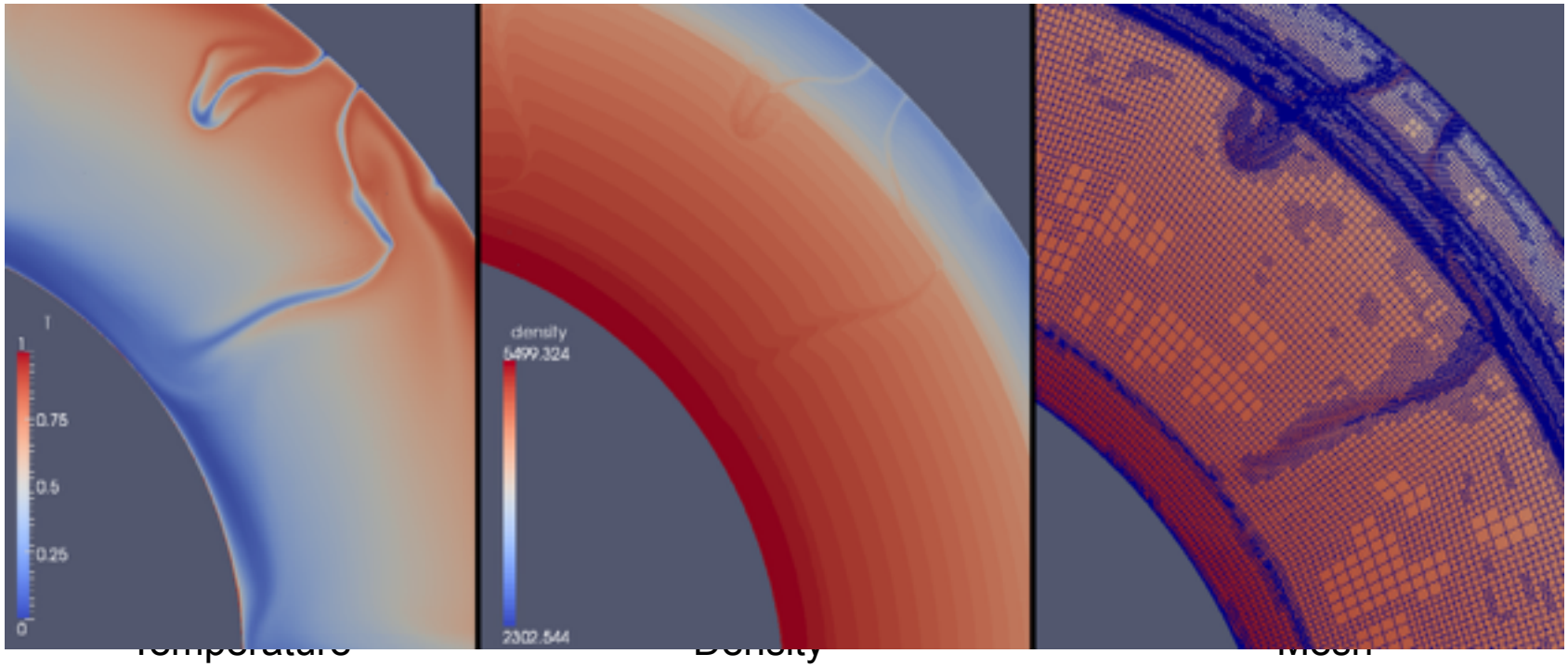
ASPECT – Results

Scalability:



ASPECT – Results

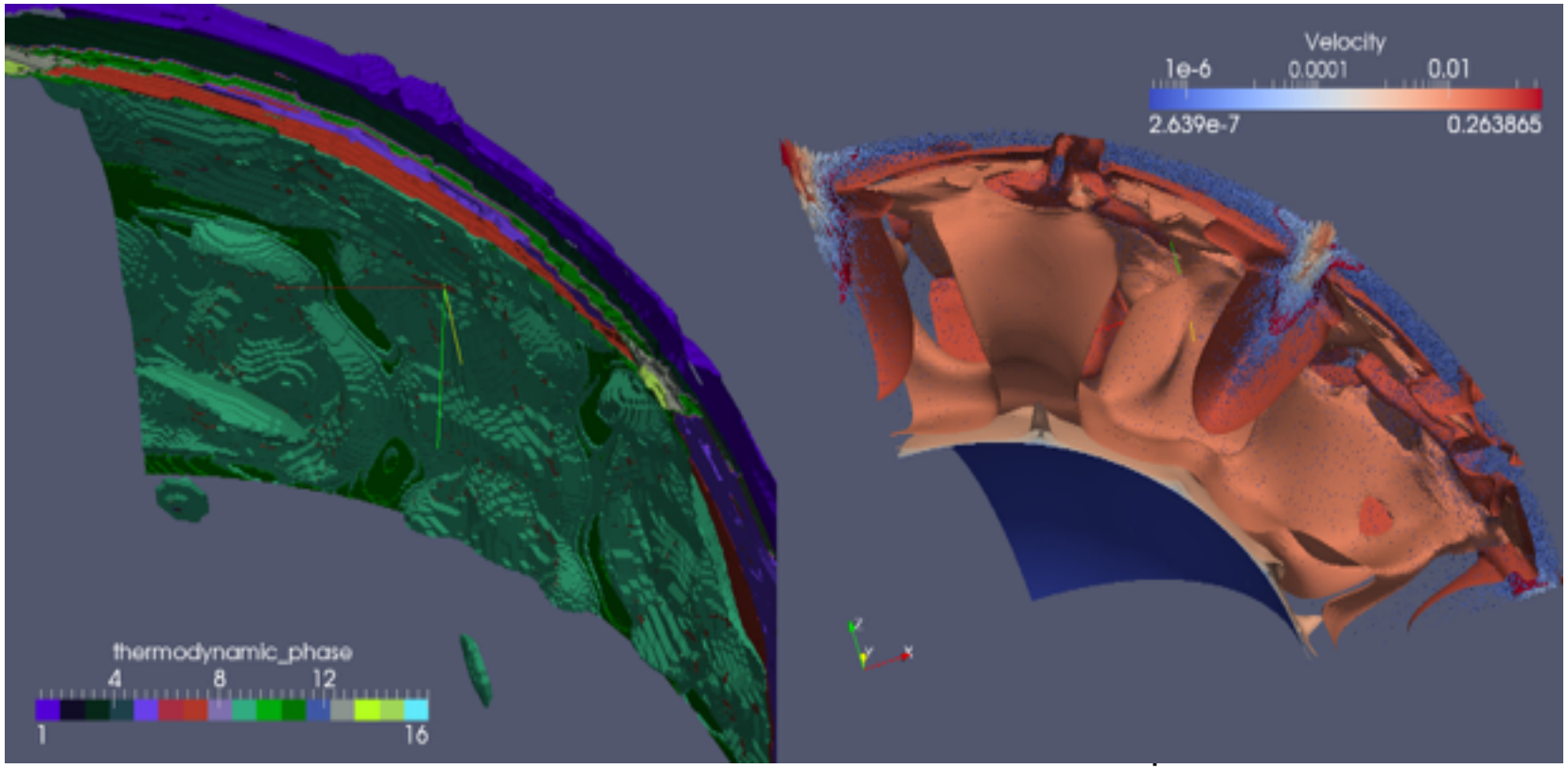
Example of a subducting slab lying on the 660km discontinuity:



Credit: Thomas Geenen, University of Utrecht

ASPECT – Results

Example of 3d computations:



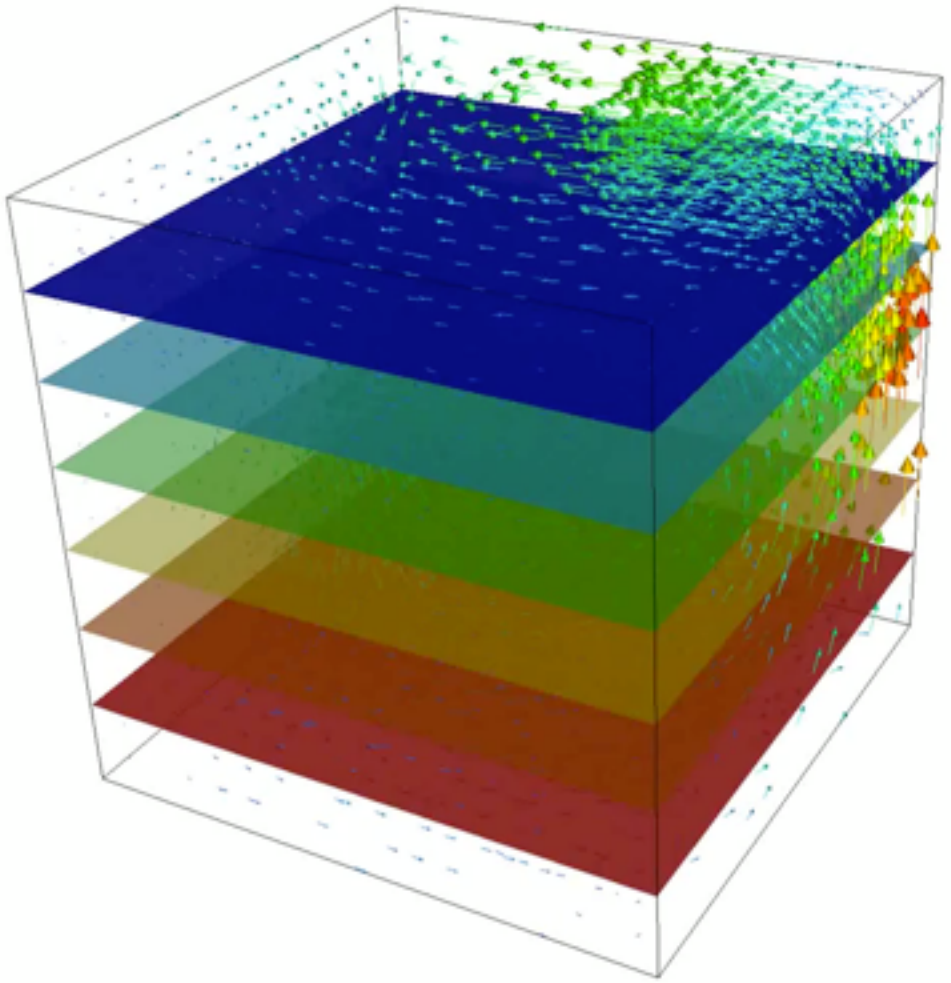
Credit: Thomas Geenen, University of Utrecht

MHPC
Master in High Performance Computing

A more “Academic” Example

Convection in a 3D Box

Source:
Wolfgang Bangerth,
Texas A&M University



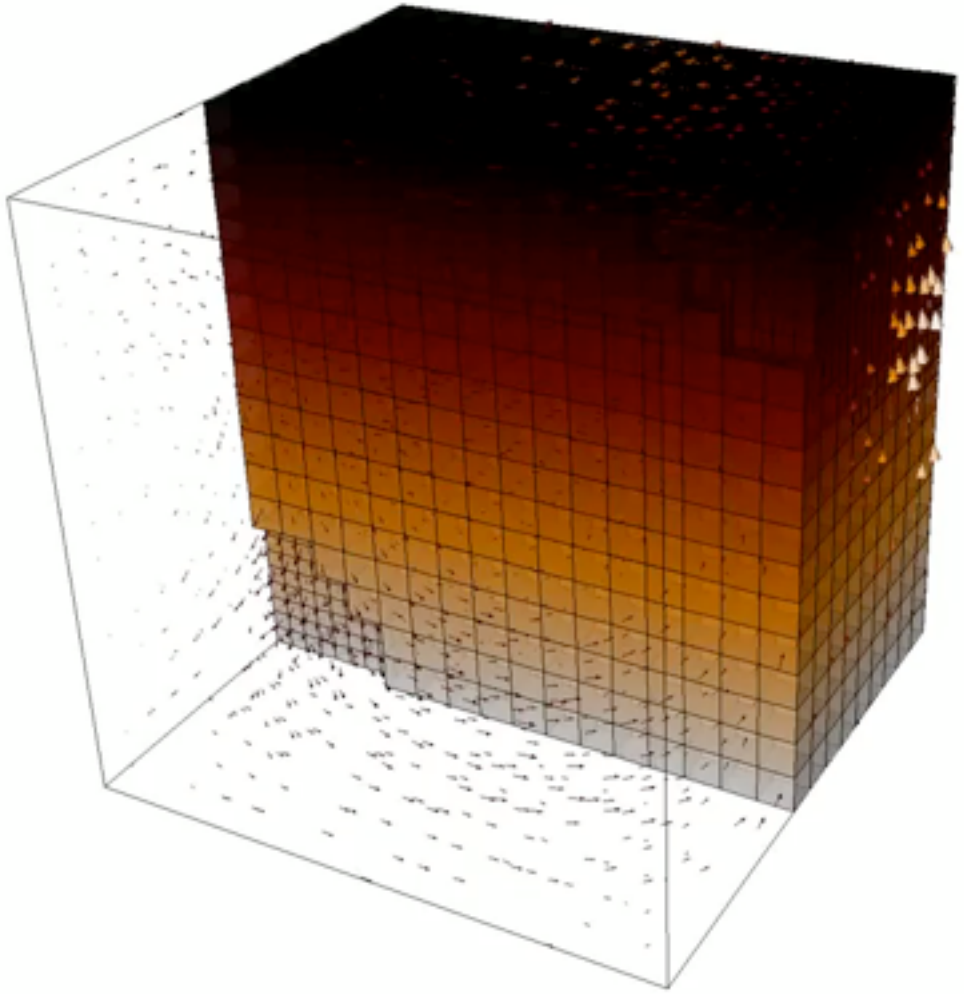
SISSA ICTP

MHPC
Master in High Performance Computing

A more “Academic” Example

Convection in a 3D Box

Source:
Wolfgang Bangerth,
Texas A&M University



SISSA ICTP

Effects

What this development model means for us:

- We can solve problems that were previously intractable

- Methods developers can demonstrate applicability

- Applications scientists can use state of the art methods

- Our codes become far smaller:
  – less potential for error
  – less need for documentation

  – lower hurdle for “reproducible” research (publishing the code along with the paper)

- More impact/more citations when publishing one's code

Effects

What this development model means for our community:

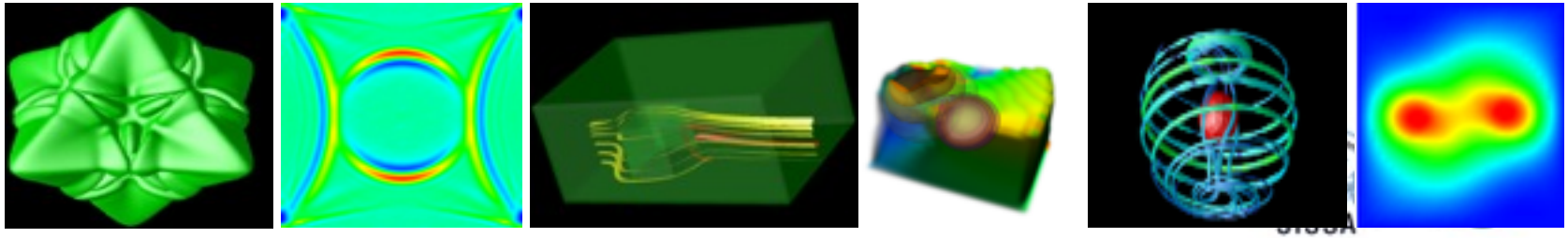
- Faster progress towards “real” applications
- Leveling the playing field – excellent online resources are there for *all*
- Raising the standard in research:
  – can't get 2d papers published any more
  – reviewers can require state-of-the-art solvers
  – allows for easier comparison of methods

Conclusions

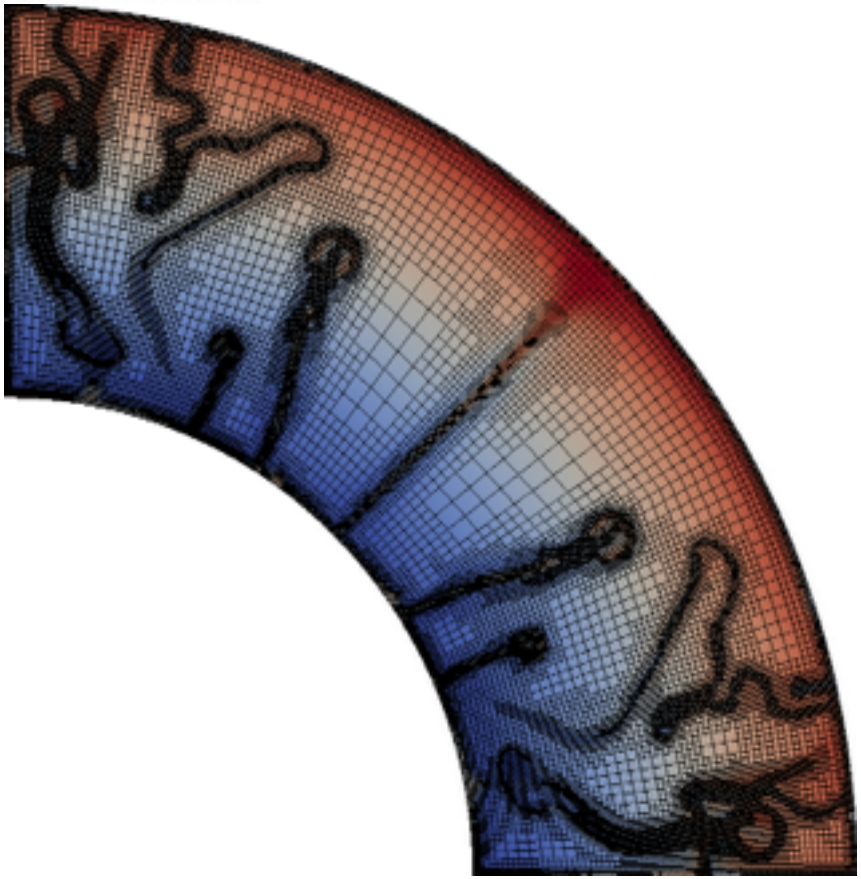
Computational science has spent too much time where everyone writes their own software.

By building on existing, well written and well tested, software packages:

- We build codes *much* faster
- We build better codes
- We can solve more realistic problems



MHPC
Master in High Performance Computing



More information:

http://www.dealii.org/

http://aspect.dealii.org/

http://www.openship.it/