

# Parallel Computations

Timo Heister, Clemson University  
[heister@clemson.edu](mailto:heister@clemson.edu)

2015-08-05  
deal.II workshop 2015



## Introduction

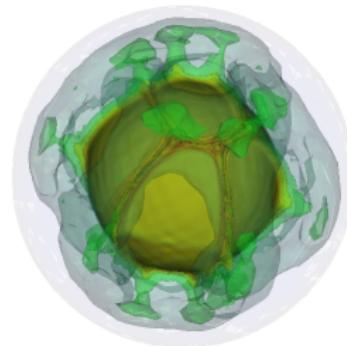
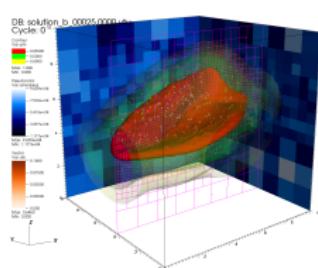
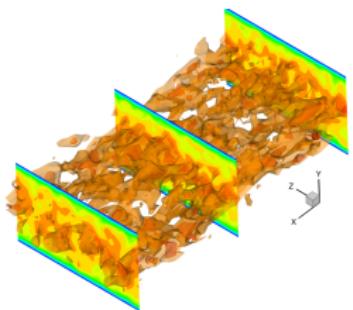
- Parallel computations with deal.II: Introduction
  - Applications
  - Parallel, adaptive, geometric Multigrid
  - Ideas for the future: parallelization

# My Research

1. Parallelization for large scale, adaptive computations
2. Flow problems: stabilization, preconditioners
3. Many other applications



IBM Sequoia, 1.5 million cores,  
source: [nextbigfuture.com](http://nextbigfuture.com)



# Parallel Computing

	<b>Before</b>	<b>Now (2012)</b>
Scalability	ok up to 100 cores	16,000+ cores
# unknowns	maybe 10 million	5+ billion

## Ideas:

- ❖ Fully parallel, scalable
- ❖ Keep flexibility(!)
- ❖ Abstraction for the user
- ❖ Reuse existing software

## Available in deal.II, but described in a generic way:



Bangerth, Burstedde, Heister, and Kronbichler.

Algorithms and Data Structures for Massively Parallel Generic Finite Element Codes.

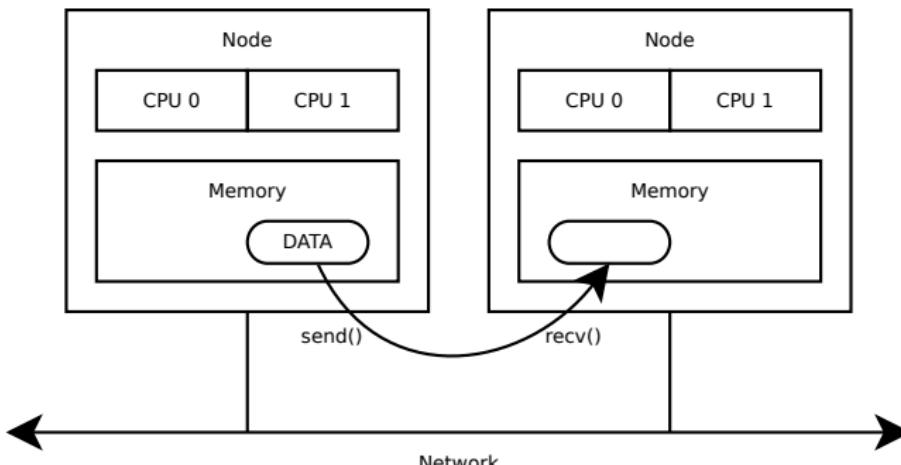
*ACM Trans. Math. Softw.*, 38(2), 2011.

# Parallel Computing Model

- System:  
nodes connected via fast network
- Model: MPI
- we here ignore:  
multithreading and vectorization



IBM Sequoia, 1.5 million cores,  
source: [nextbigfuture.com](http://nextbigfuture.com)



# Parallel Computations: How To? Why?

- ❖ Required: split up the work!
- ❖ Goal: get solutions faster, allow larger problems
- ❖ Who needs this?
  - ❖ 3d computations?
  - ❖ > 500, 000 unknowns?
- ❖ From laptop to supercomputer!

# Scalability

What is scalability?

(you should know about weak/strong scaling, parallel efficiency, hardware layouts, NUMA, interconnects, . . . )

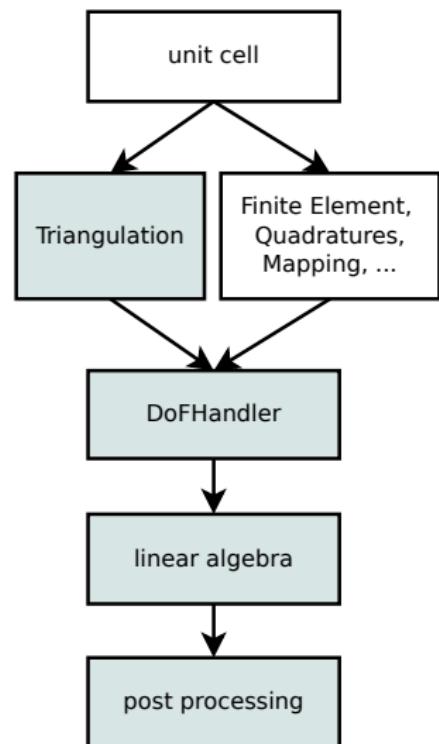
Required for Scalability:

- ❖ 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

# Overview of Data Structures and Algorithms

Needs to be parallelized:

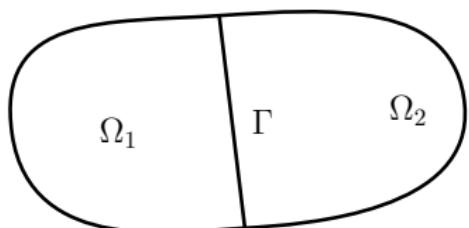
1. Triangulation (mesh with associated data)
  - hard: distributed storage, new algorithms
2. DoFHandler (manages degrees of freedom)
  - hard: find global numbering of DoFs
3. Linear Algebra (matrices, vectors, solvers)
  - use existing library
4. Postprocessing (error estimation, solution transfer,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



# How to do Parallelization?

## Option 1: Domain Decomposition

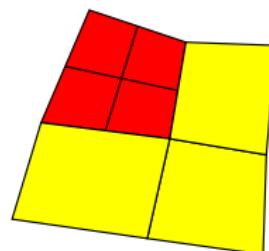
- ❖ Split up problem on PDE level
- ❖ Solve subproblems independently
- ❖ Converges against global solution
- ❖ Problems:
  - ❖ Boundary conditions are problem dependent:
    - ~~ sometimes difficult!
    - ~~ no black box approach!
  - ❖ Without coarse grid solver:  
condition number grows with # subdomains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



# How to do Parallelization?

## Option 2: Algebraic Splitting

- Split up mesh between processors:



- Assemble logically global linear system  
(distributed storage):

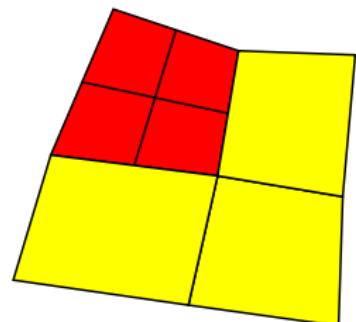
$$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix}$$

- 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
  - even distribution of work
- minimize interface between region
  - reduce communication

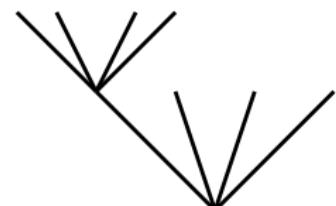
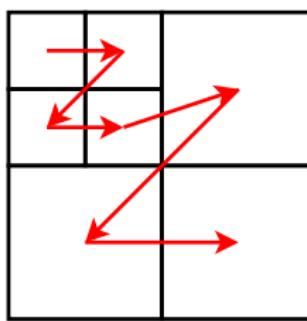
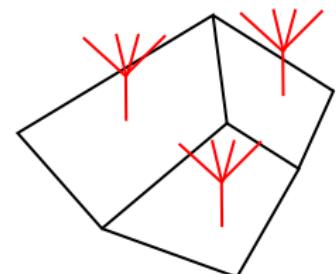


Optimal partitioning is an NP-hard  
**graph partitioning** problem.

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

# Partitioning using Space-Filling Curves

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



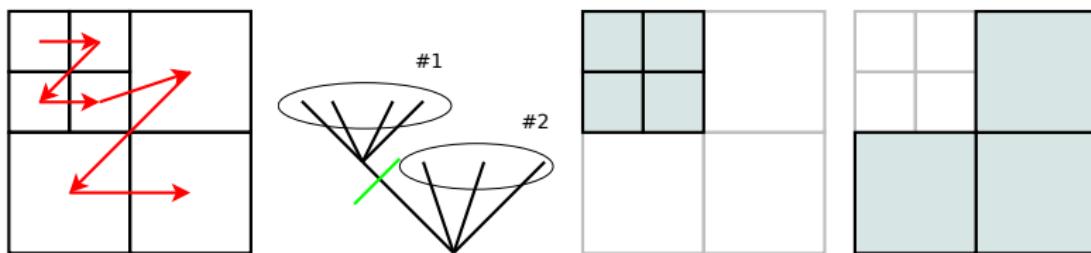
Burstedde, Wilcox, and Ghattas.

*p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees.*

*SIAM J. Sci. Comput.*, 33 no. 3 (2011), pages 1103-1133.

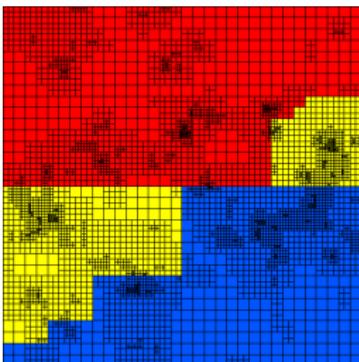
# Triangulation

- Partitioning is cheap and simple:

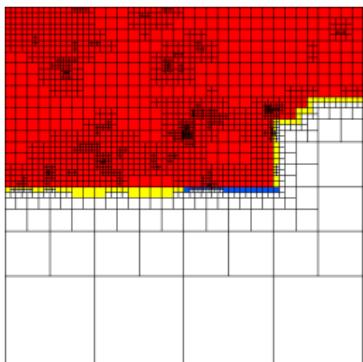


- Then: take *p4est* refinement information
- Recreate rich *deal.II* Triangulation only for local cells  
(stores coordinates, connectivity, faces, materials, . . . )
- How? recursive queries to *p4est*
- Also create ghost layer (one layer of cells around own ones)

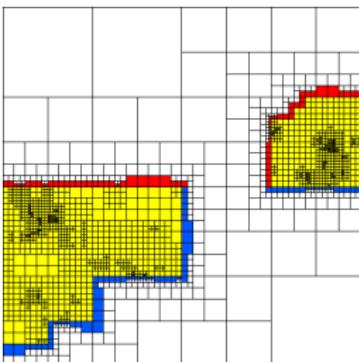
# Example: Distributed Mesh Storage



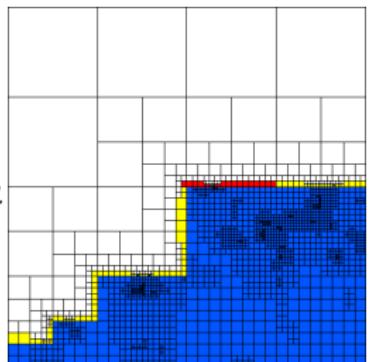
=



&amp;



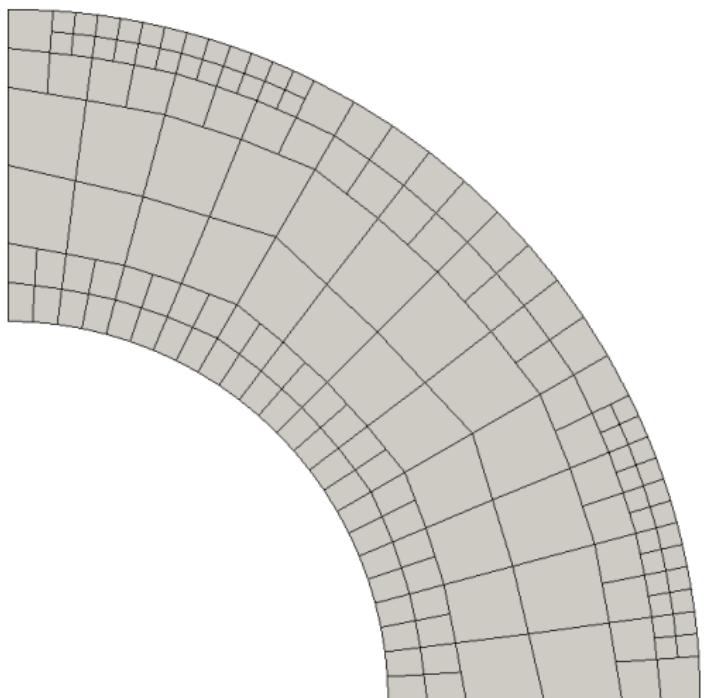
&amp;



Color: owned by CPU id

# Arbitrary Geometry and Limitations

- ❖ Curved domains/boundaries using higher order mappings and manifold descriptions
- ❖ Arbitrary geometry



## Limitations:

- ❖ Only regular refinement
- ❖ Limited to quads/hexas
- ❖ Coarse mesh duplicated on all nodes

# In Practice

## 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!



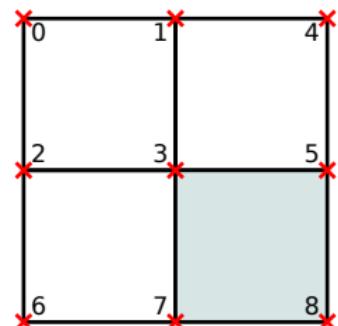
# Meshes in deal.II

	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	good?
hp?	yes	(planned)	yes?
geom. MG?	yes	in progress	?
Aniso. ref.?	yes	no	(offline only)
Periodicity	yes	yes	?
Scalability	100 cores	16k+ cores	?

parallel::shared::Triangulation will address some shortcomings of “serial mesh”: do not duplicate linear algebra, same API as parallel::distributed, ...

# Distributing the Degrees of Freedom (DoFs)

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



Sketch:

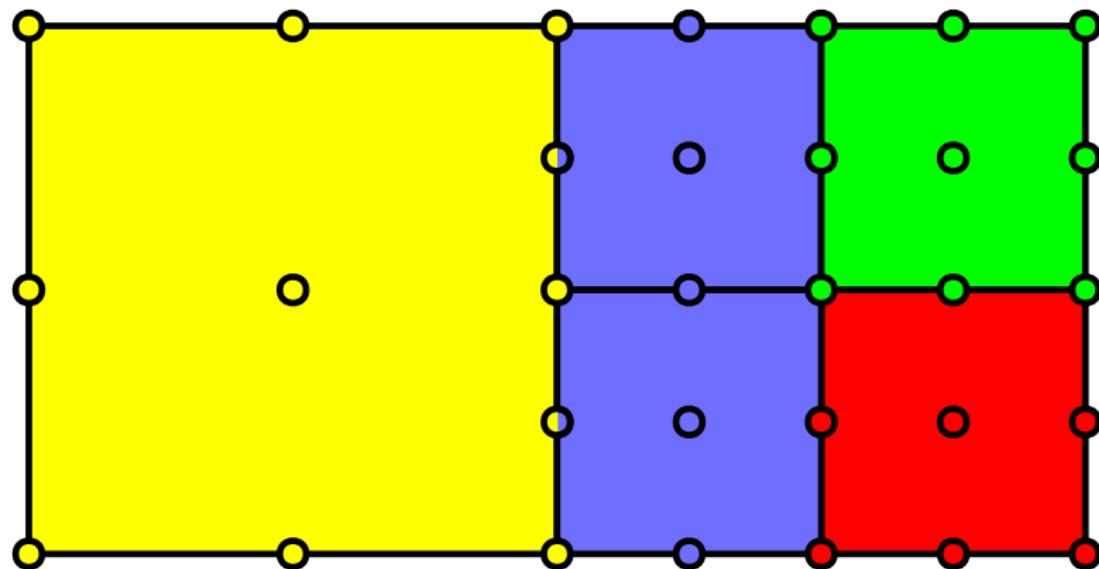
- Decide on ownership of DoFs on interface (no communication!)
- Enumerate locally (only own DoFs)
- Shift indices to make them globally unique (only communicate local quantities)
- Exchange indices to ghost neighbors

# Linear Algebra: Short Version

$$\begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix}$$

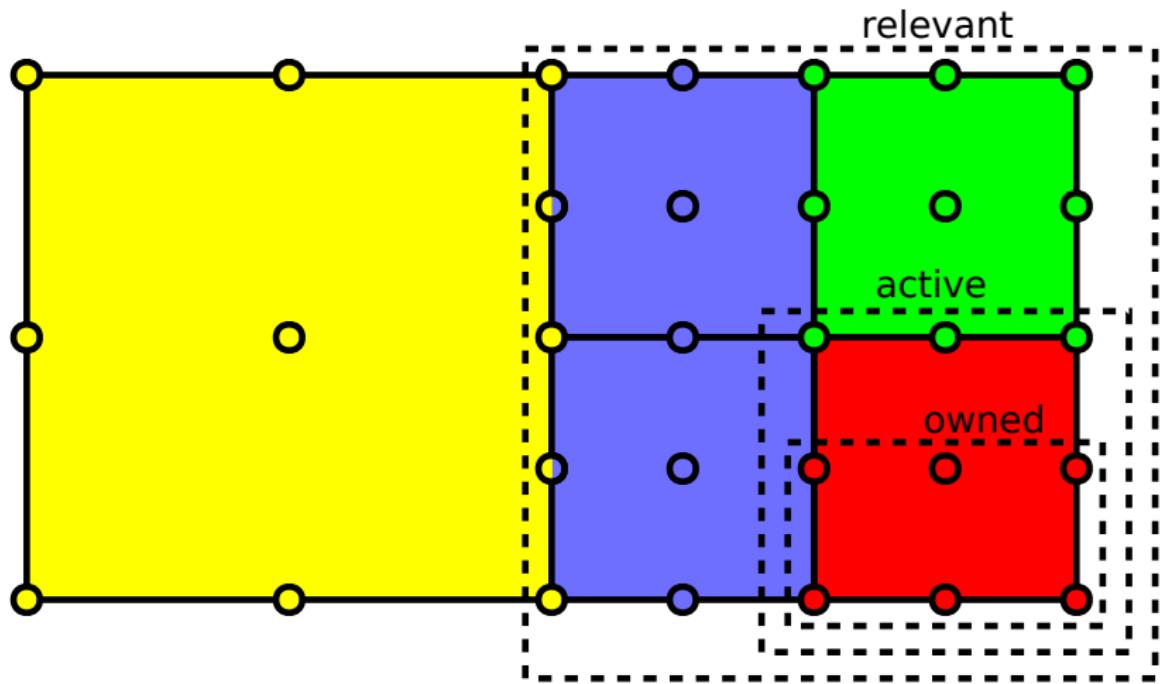
- ❖ Use distributed matrices and vectors
- ❖ Assemble local parts (some communication on interfaces)
- ❖ Iterative solvers (CG, GMRES, ...) are equivalent, only need:
  - ❖ Matrix-vector products
  - ❖ scalar products
- ❖ Preconditioners:
  - ❖ always problem dependent
  - ❖ similar to serial: block factorizations, Schur complement approximations
  - ❖ not enough: combine preconditioners on each node
  - ❖ good: algebraic multigrid
  - ❖ in progress: geometric multigrid

# Longer Version



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

# Longer Version: Interesting DoFs



(perspective of the **red** CPU)

# DoF Sets

- ❖ 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, ..., 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()`, ...

# Vectors/Matrices

- 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**

# Ghosted Vectors

- orange paw print icon read-only

- orange paw print icon create using

```
Vector(IndexSet owned, IndexSet ghost, MPI_COMM)  
where ghost is relevant or active
```

- orange paw print icon copy values into it by using operator=(Vector)

- orange paw print icon then just read entries you need



# Compressing Vectors/Matrices

## 🐾 Why?

- 🐾 After writing into foreign entries communication has to happen
- 🐾 All in one go for performance reasons

## 🐾 How?

- 🐾 `object.compress (VectorOperation::add);` if you added to entries
- 🐾 `object.compress (VectorOperation::insert);` if you set entries
- 🐾 This is a collective call

## 🐾 When?

- 🐾 After the assembly loop (with `::add`)
- 🐾 After you do `vec(j) = k;` or `vec(j) += k;` (and in add/insert groups)
- 🐾 In no other case (all functions inside deal.II compress if necessary)!



# Trilinos vs. PETSc



What should I use?

- Similar features and performance
- Pro Trilinos: more development, some more features (automatic differentiation, . . . ), 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

```
1 #include <deal.II/lac/generic_linear_algebra.h>
2 #define USE_PETSC_LA // uncomment this to run with Trilinos
3
4 namespace LA
5 {
6 #ifdef USE_PETSC_LA
7     using namespace dealii::LinearAlgebraPETSc;
8 #else
9     using namespace dealii::LinearAlgebraTrilinos;
10 #endif
11 }
12
13 // ...
14 LA::MPI::SparseMatrix system_matrix;
15 LA::MPI::Vector solution;
16
17 // ...
18 LA::SolverCG solver(solver_control, mpi_communicator);
19 LA::MPI::PreconditionAMG preconditioner;
20
21 LA::MPI::PreconditionAMG::AdditionalData data;
22
23 #ifdef USE_PETSC_LA
24     data.symmetric_operator = true;
25 #else
26     //trilinos defaults are good
27 #endif
28     preconditioner.initialize(system_matrix, data);
29
30 // ...
```

# Postprocessing, . . .

Not covered today:

- Error estimation
- Decide over refinement and coarsening (communication!)
- Handling hanging nodes and other constraints
- Solution transfer (after refinement and repartitioning)
- Parallel I/O
- and probably more

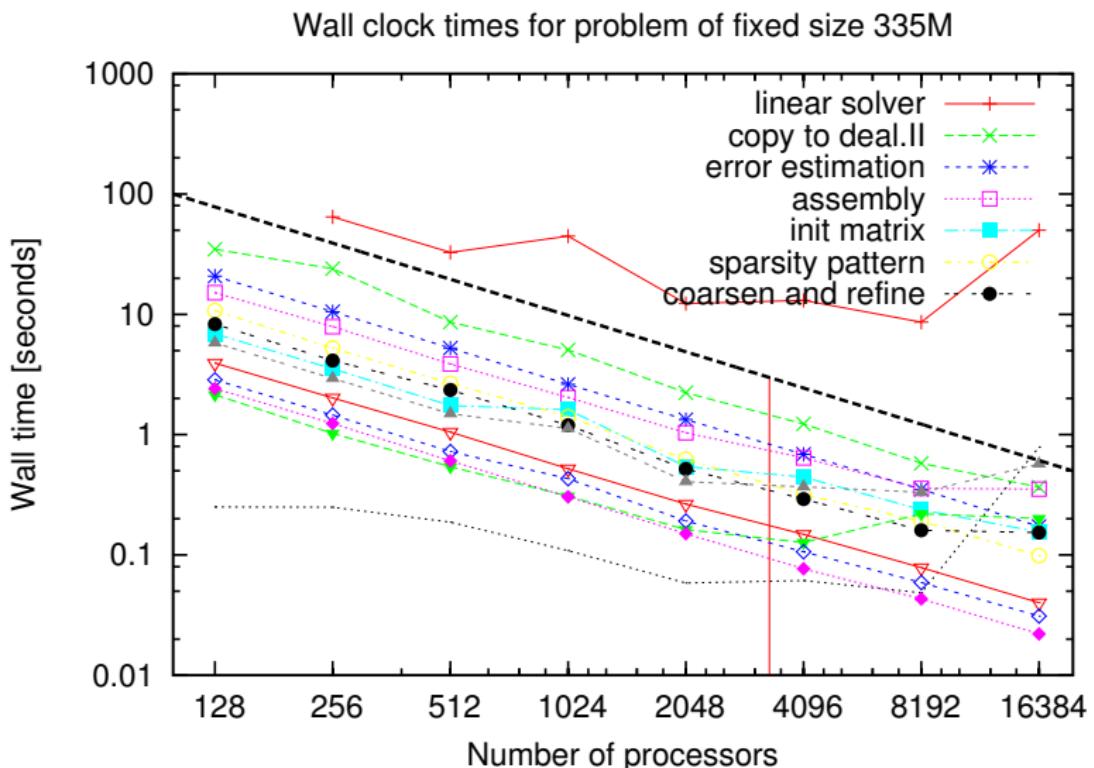
# My workflow

1. Write code in serial to test ideas, use UMFPACK
2. Switch to use MPI using namespace LA with direct solver (SparseDirect) – [can we remove this step?]
3. Linear solver: Schur complement based block preconditioner with AMG for each block
4. Profit!

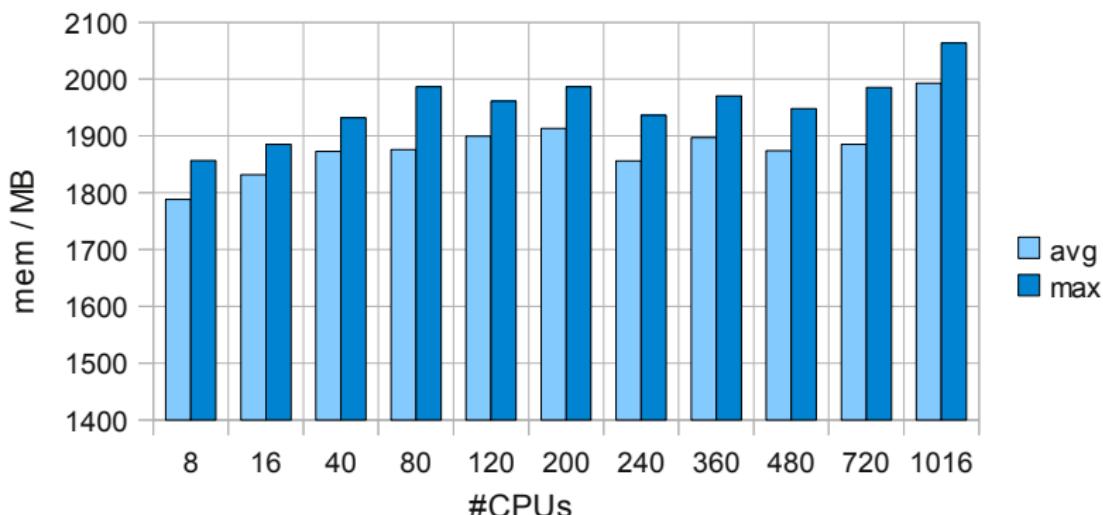
# Wishlist

- ❖ Unify and simplify linear algebra? Include serial matrices/vectors?
- ❖ Need to incorporate even more backends (Tpetra, new PETSc, ...)
- ❖ Make something like namespace LA the default? Use inheritance instead?
- ❖ Need: different API for writing into vectors because of multithreading
- ❖ Our block matrices don't mesh well with PETSc

# Strong Scaling: 2d Adaptive Poisson Problem



# Test: Memory Consumption



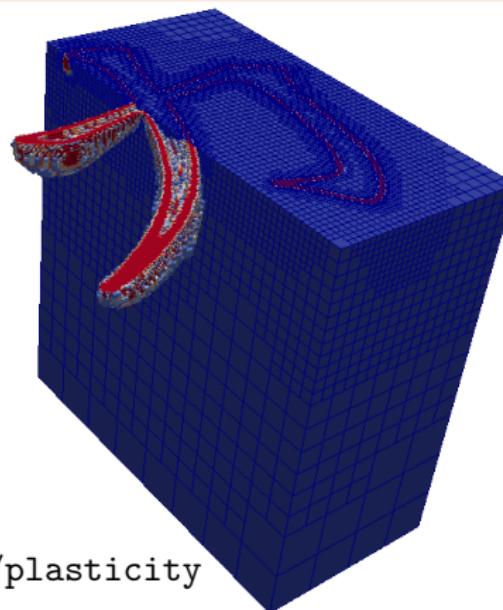
average and maximum memory consumption (VmPeak)

3D, weak scalability from 8 to 1000 processors with about 500.000  
DoFs per processor (4 million up to 500 million total)

~~ Constant memory usage with increasing  
# CPUs & problem size

# Plasticity

- 3d contact problem
- elasto-plastic material
- isotropic hardening
- semi-smooth Newton  
+ active set strategy



code: step-42 or <https://github.com/tjhei/plasticity>

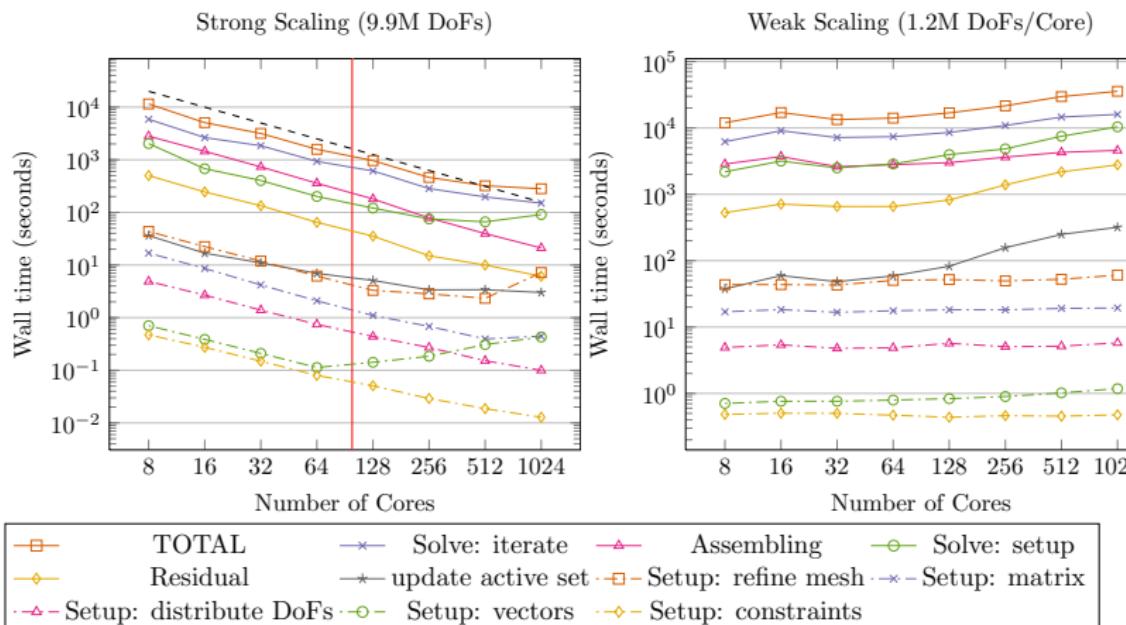


Frohne, Heister, Bangerth.

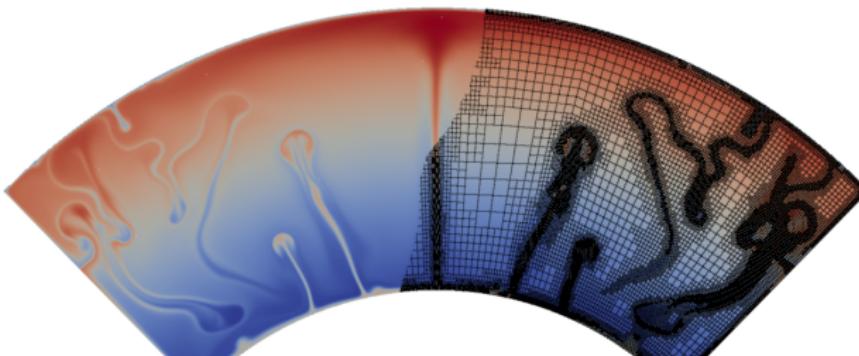
Efficient numerical methods for the large-scale,  
parallel solution of elastoplastic contact problems.

Accepted for publication in IJNME.

# Plasticity: Scalability



# ASPECT



(temperature snapshot, 700,000 degrees of freedom, 2d simulation)

- ❖ ASPECT: <http://aspect.dealii.org/>
  - ❖ Global mantle convection in the Earth's mantle
  - ❖ 3d computations, adaptive meshes, 100 million+ DoFs
  - ❖ Need: fast refinement, partitioning
  - ❖ <https://www.youtube.com/watch?v=iwm68TC5YxM>, file:aspect.mp4
-  Kronbichler, Heister, and Bangerth.  
High Accuracy Mantle Convection Simulation through Modern Numerical Methods.  
*Geophysical Journal International*, 2012, 191, 12-29.

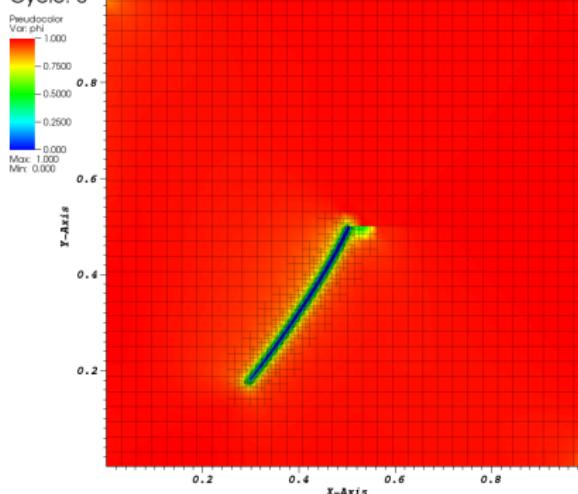
# Crack propagation



## Crack propagation:

DB: solution\_00212.0000.vtu

Cycle: 0



code: <https://github.com/tjhei/cracks>



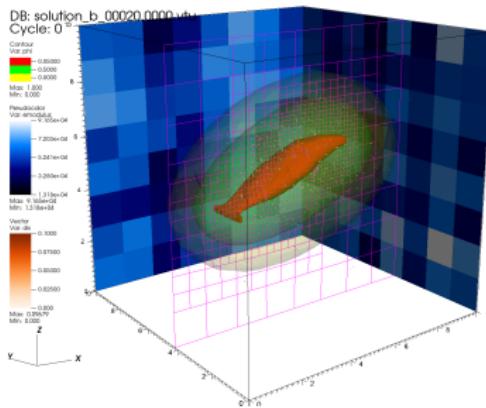
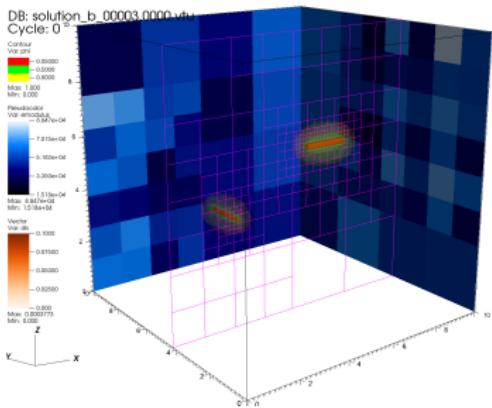
Heister, Wheeler, Wick.

A primal-dual active set method and predictor-corrector mesh adaptivity for computing fracture propagation using a phase-field approach.

CMAME, Volume 290, 15 June 2015

# 3D Computations

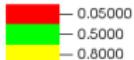
- 3d adaptive test problem
- pressurized crack in heterogeneous medium
- novel adaptive refinement strategy



# DB: solution\_b\_00003.0000.vtu

Cycle: 0

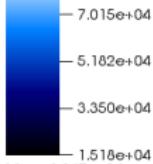
Contour  
Var: phi



Max: 1.000  
Min: 0.000

Pseudocolor

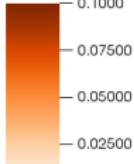
Var: emodulus  
— 8.847e+04



Max: 8.847e+04  
Min: 1.518e+04

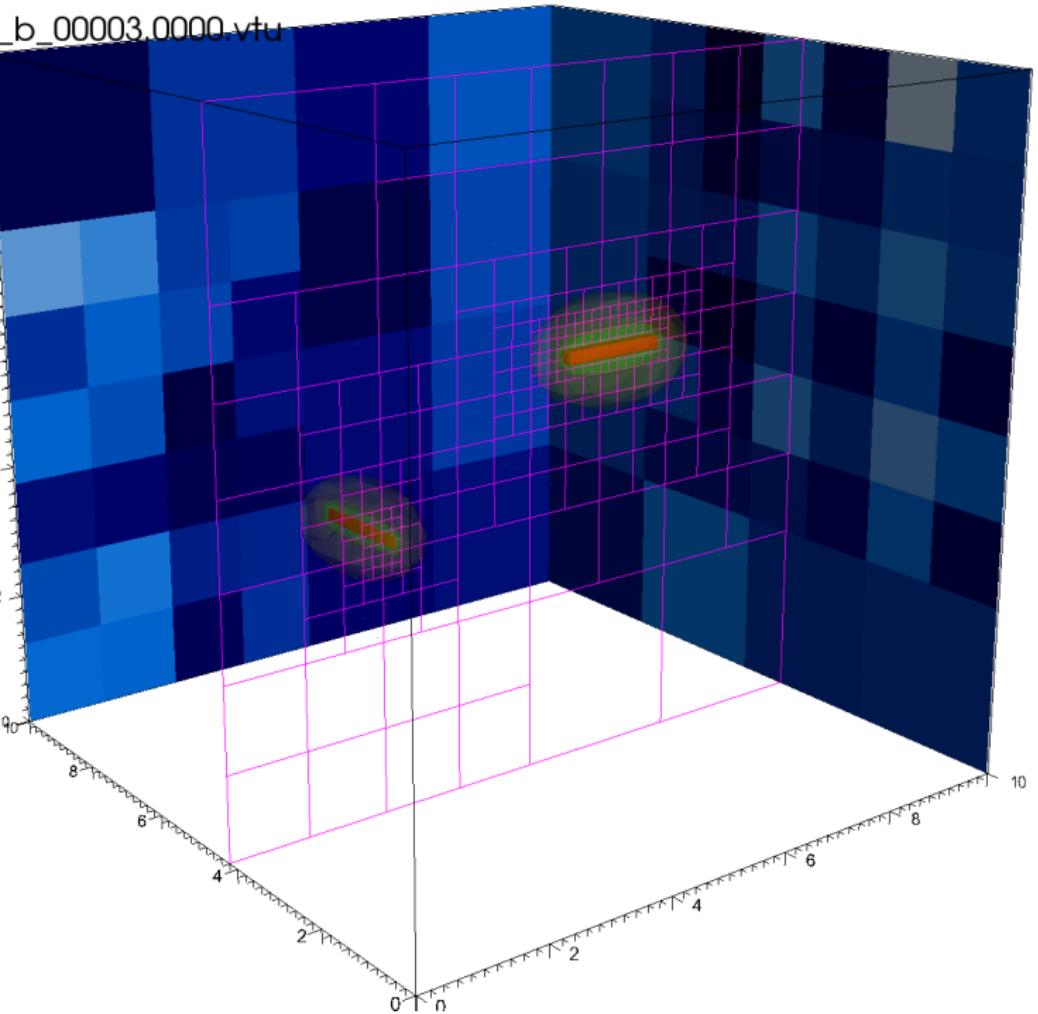
Vector

Var: dis



Max: 0.0003773  
Min: 0.000

Z  
X  
Y



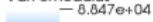
DB: solution\_b\_00013.0000.vtu

Cycle: 0

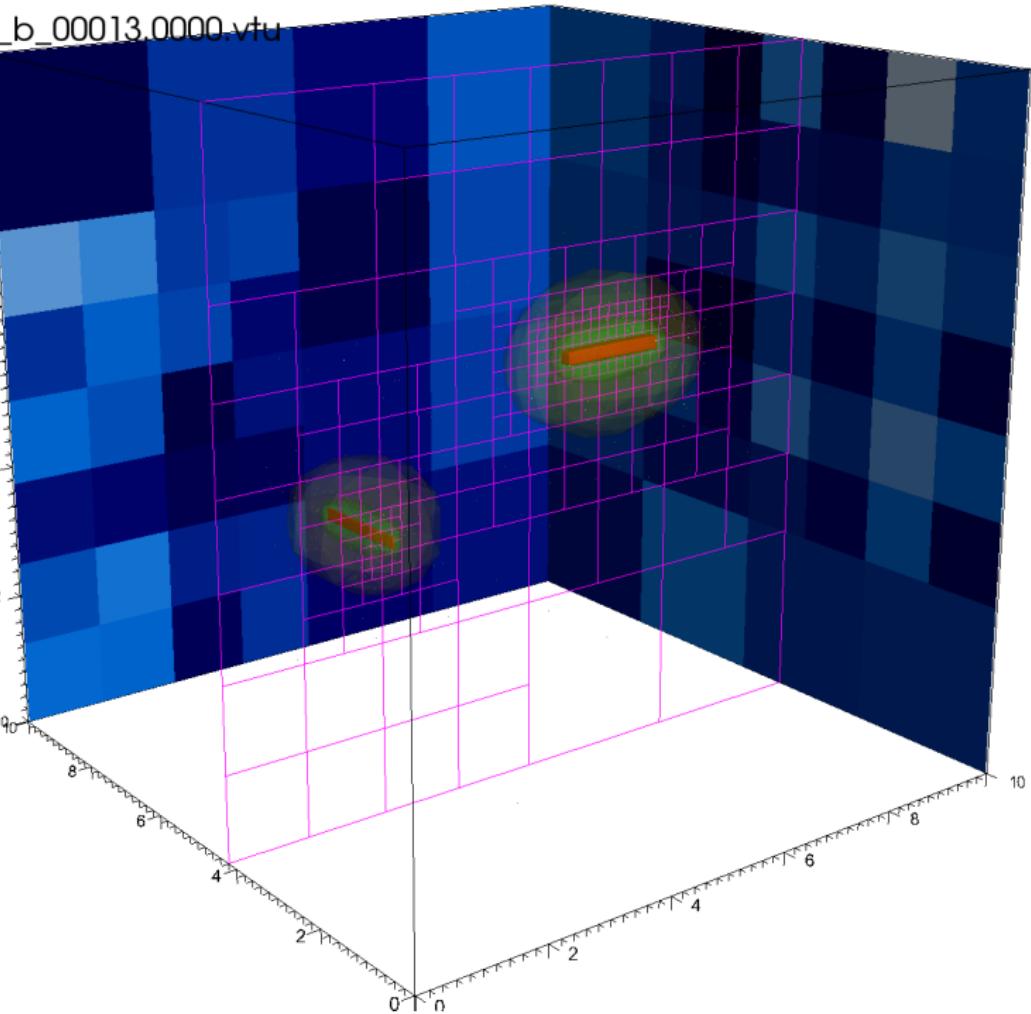
Contour  
Var: phi



Pseudocolor  
Var: emodulus



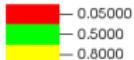
Vector  
Var: dis



# DB: solution\_b\_00014.0000.vtu

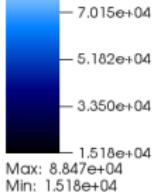
Cycle: 0

Contour  
Var: phi

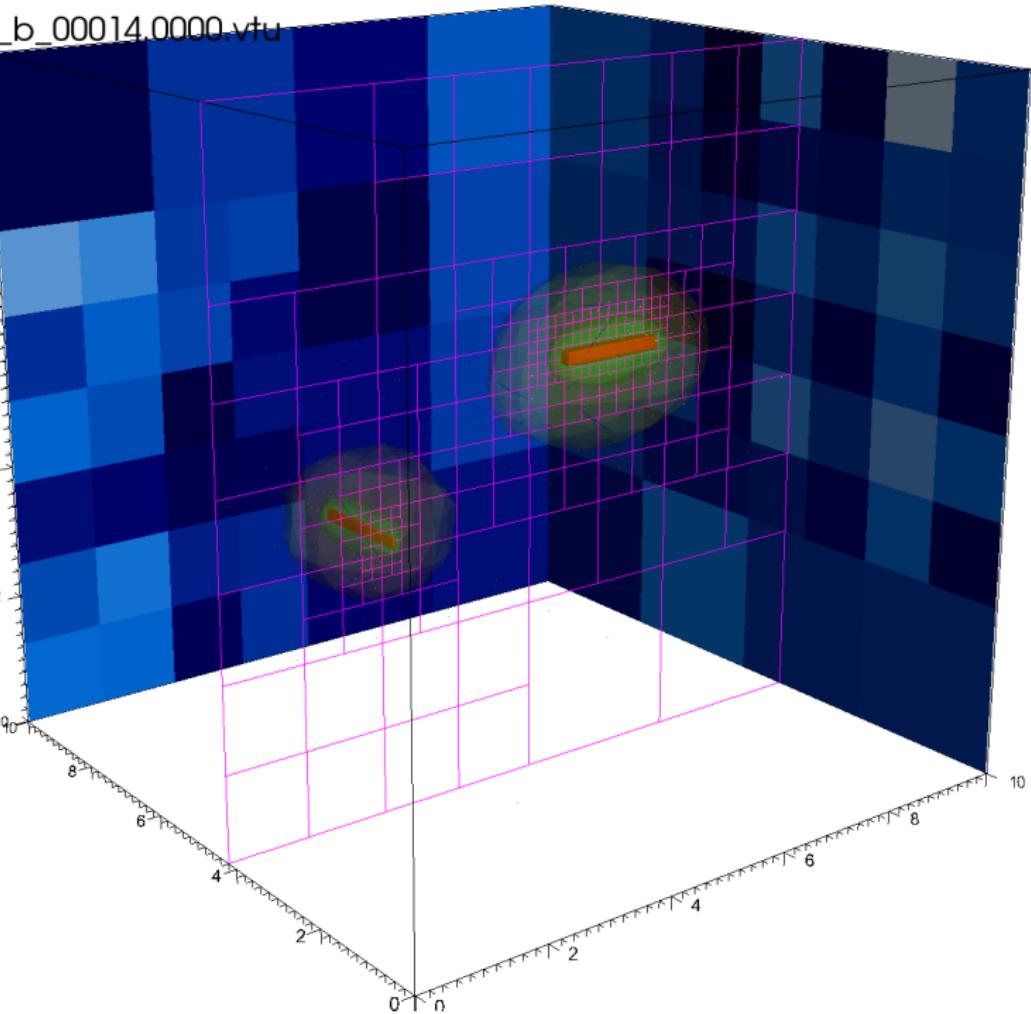
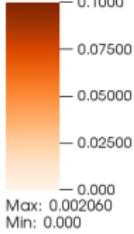


Pseudocolor

Var: emodulus  
— 8.847e+04



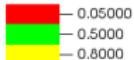
Vector  
Var: dis



DB: solution\_b\_00015.0000.vtu

Cycle: 0

Contour  
Var: phi



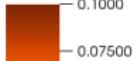
Max: 1.000  
Min: 0.000

Pseudocolor  
Var: emodulus

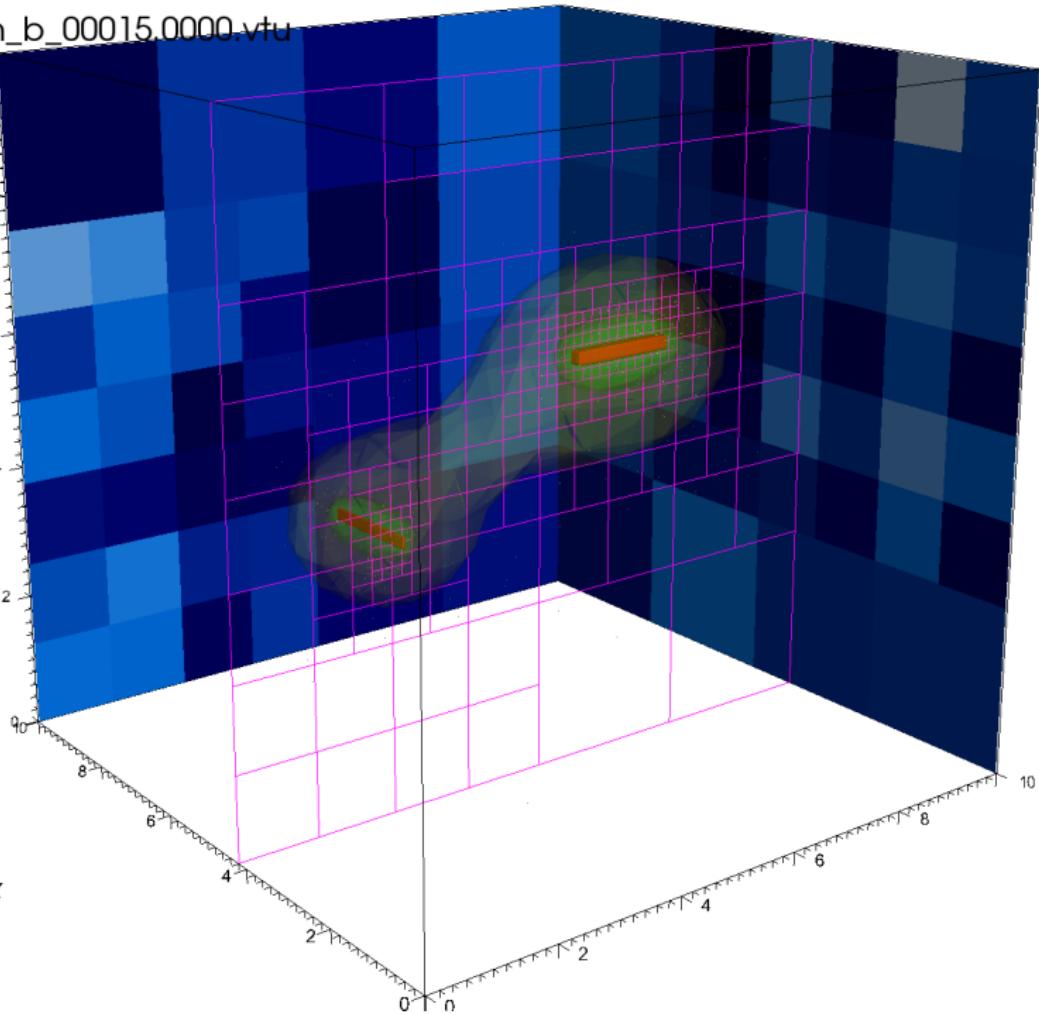


Max: 8.847e+04  
Min: 1.518e+04

Vector  
Var: dis



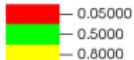
Max: 0.002332  
Min: 0.0000



DB: solution\_b\_00016.0000.vtu

Cycle: 0

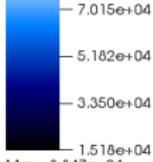
Contour  
Var: phi



Max: 1.000  
Min: 0.000

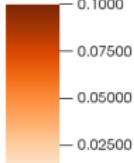
Pseudocolor

Var: emodulus  
- 8.847e+04

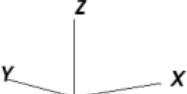


Max: 8.847e+04  
Min: 1.518e+04

Vector  
Var: dis



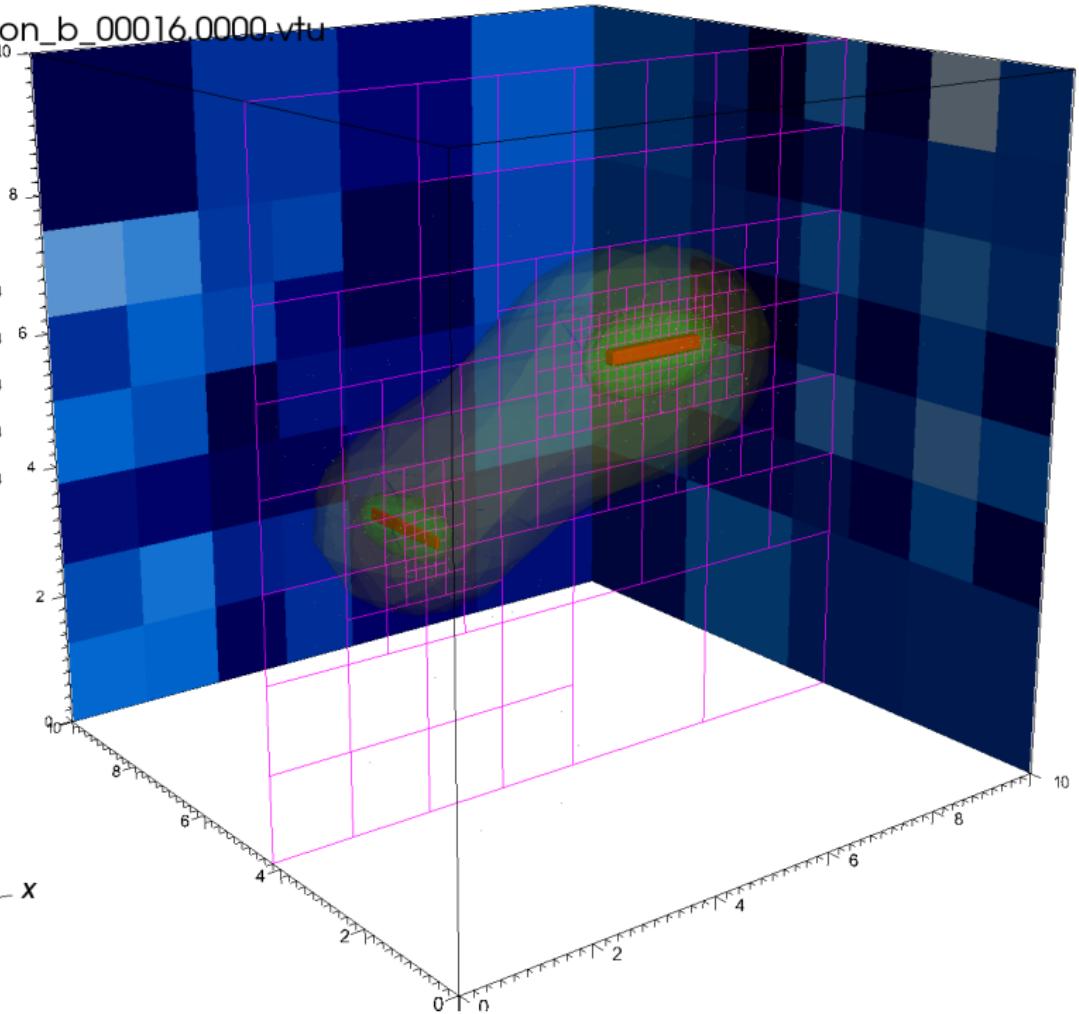
Max: 0.002690  
Min: 0.000



Z

X

Y



DB: solution\_b\_00017.0000.vtu

Cycle: 0

Contour  
Var: phi

red	-0.05000
green	-0.5000
yellow	-0.8000

Max: 1.000  
Min: 0.000

Pseudocolor

Var: emodulus  
- 8.847e+04

blue	- 7.015e+04
dark blue	- 5.182e+04
medium blue	- 3.350e+04
light blue	- 1.518e+04
white	Max: 8.847e+04
black	Min: 1.518e+04

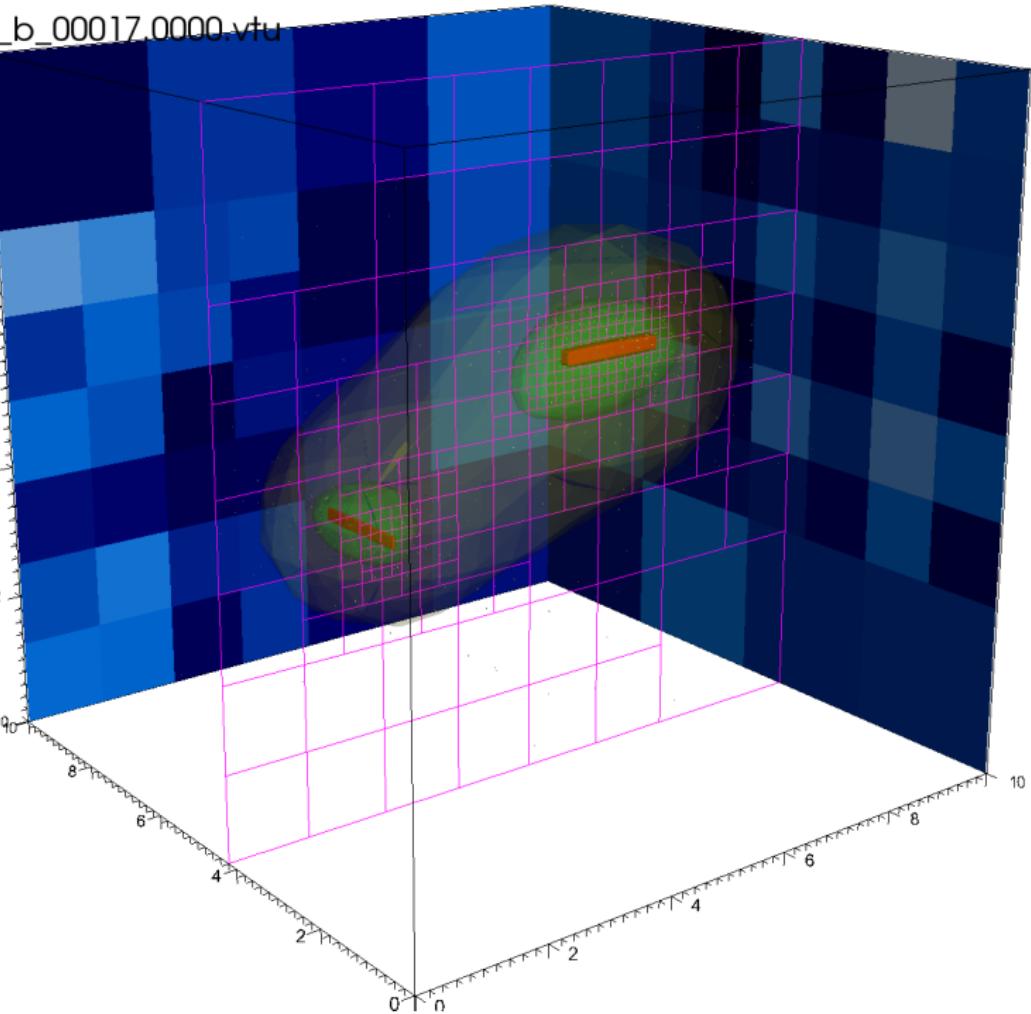
Max: 8.847e+04  
Min: 1.518e+04

Vector  
Var: dis

orange	-0.1000
yellow	-0.07500
light yellow	-0.05000
light orange	-0.02500
white	Max: 0.003212
black	Min: 0.000

Max: 0.003212  
Min: 0.000

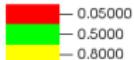
Z  
X  
Y



# DB: solution\_b\_00018.0000.vtu

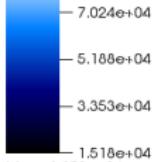
Cycle: 0

Contour  
Var: phi

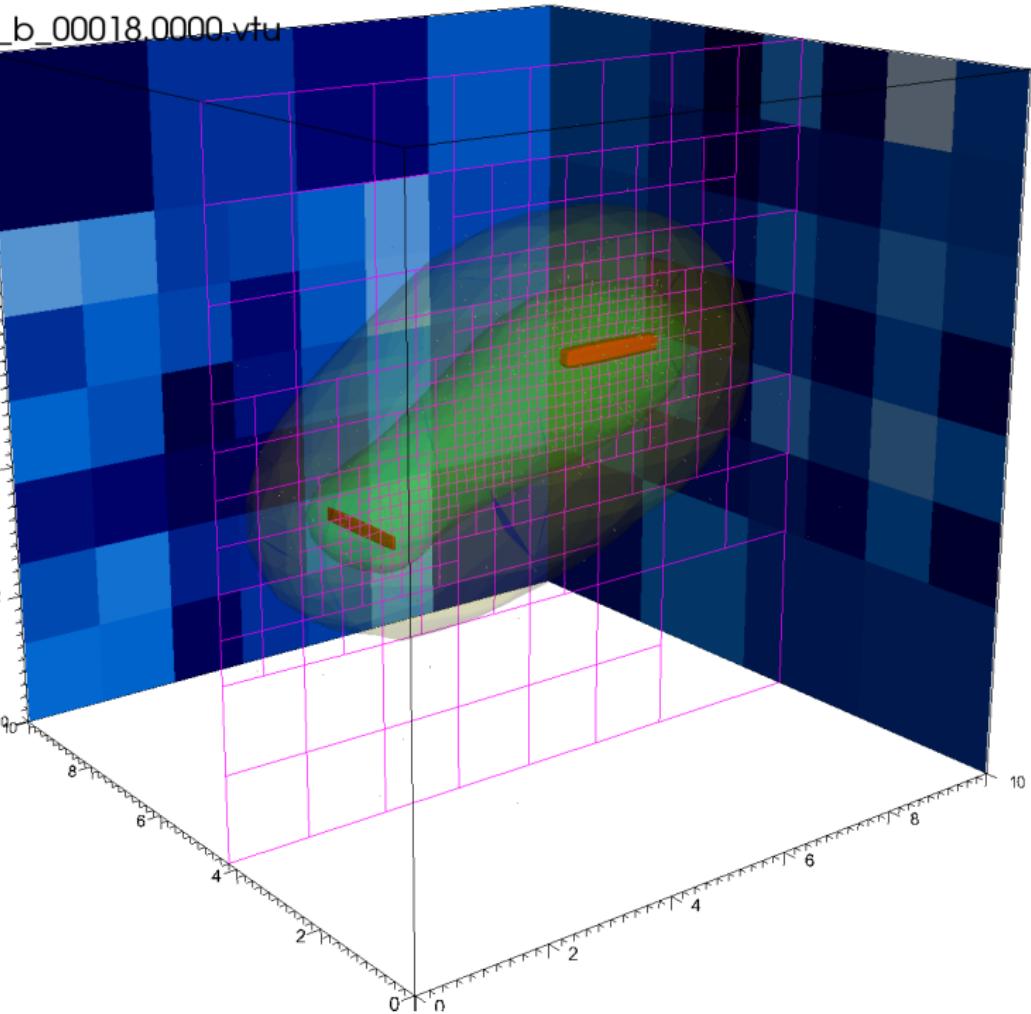
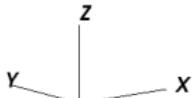
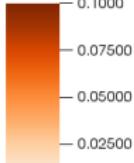


Pseudocolor

Var: emodulus  
— 8.859e+04



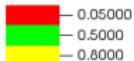
Vector  
Var: dis



# DB: solution\_b\_00019.0000.vtu

Cycle: 0

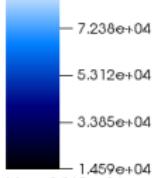
Contour  
Var: phi



Max: 1.000  
Min: 0.000

Pseudocolor

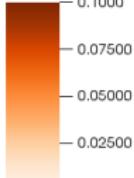
Var: emodulus  
— 9.165e+04



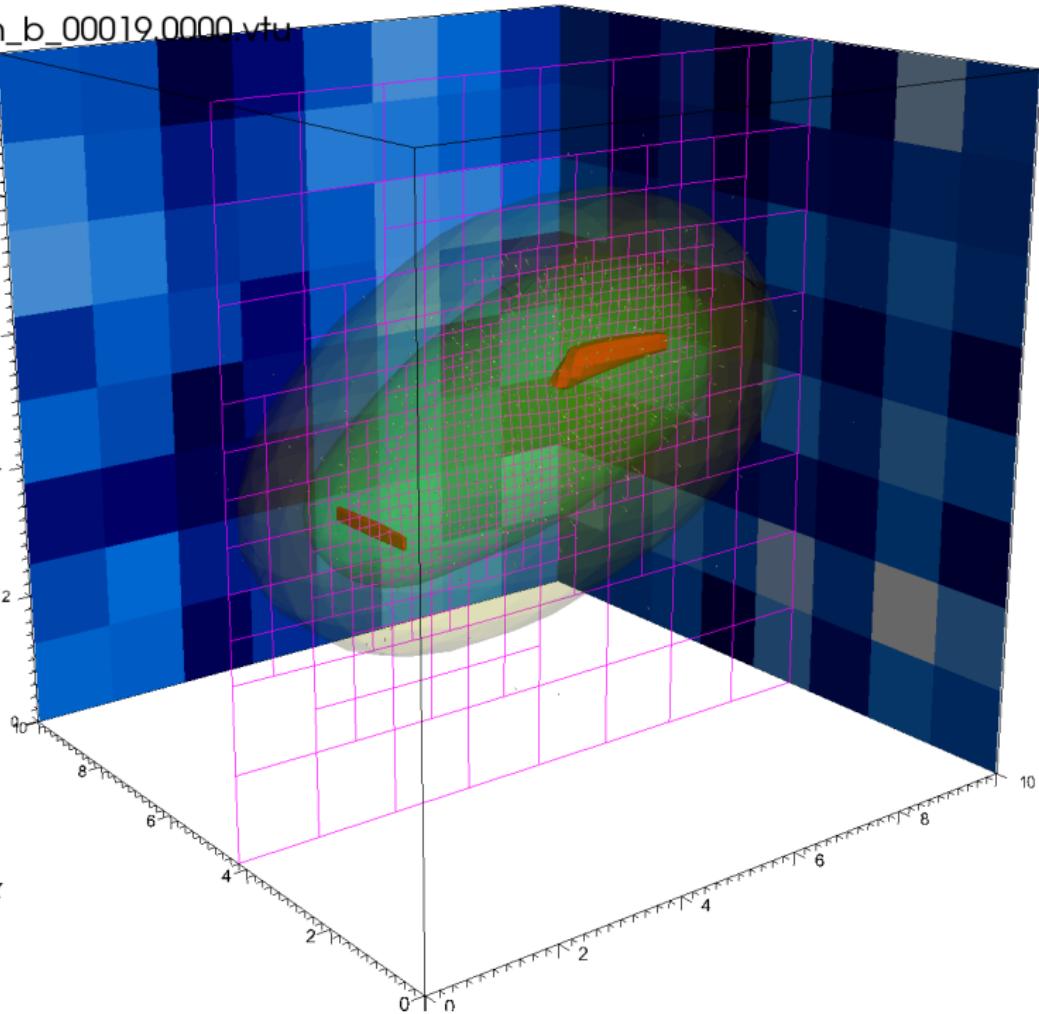
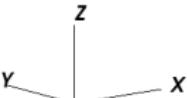
Max: 9.165e+04  
Min: 1.459e+04

Vector

Var: dis



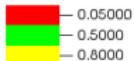
Max: 0.006843  
Min: 0.000



# DB: solution\_b\_00020,0000.vtu

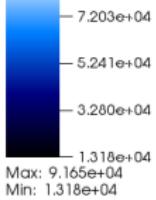
Cycle: 0

Contour  
Var: phi

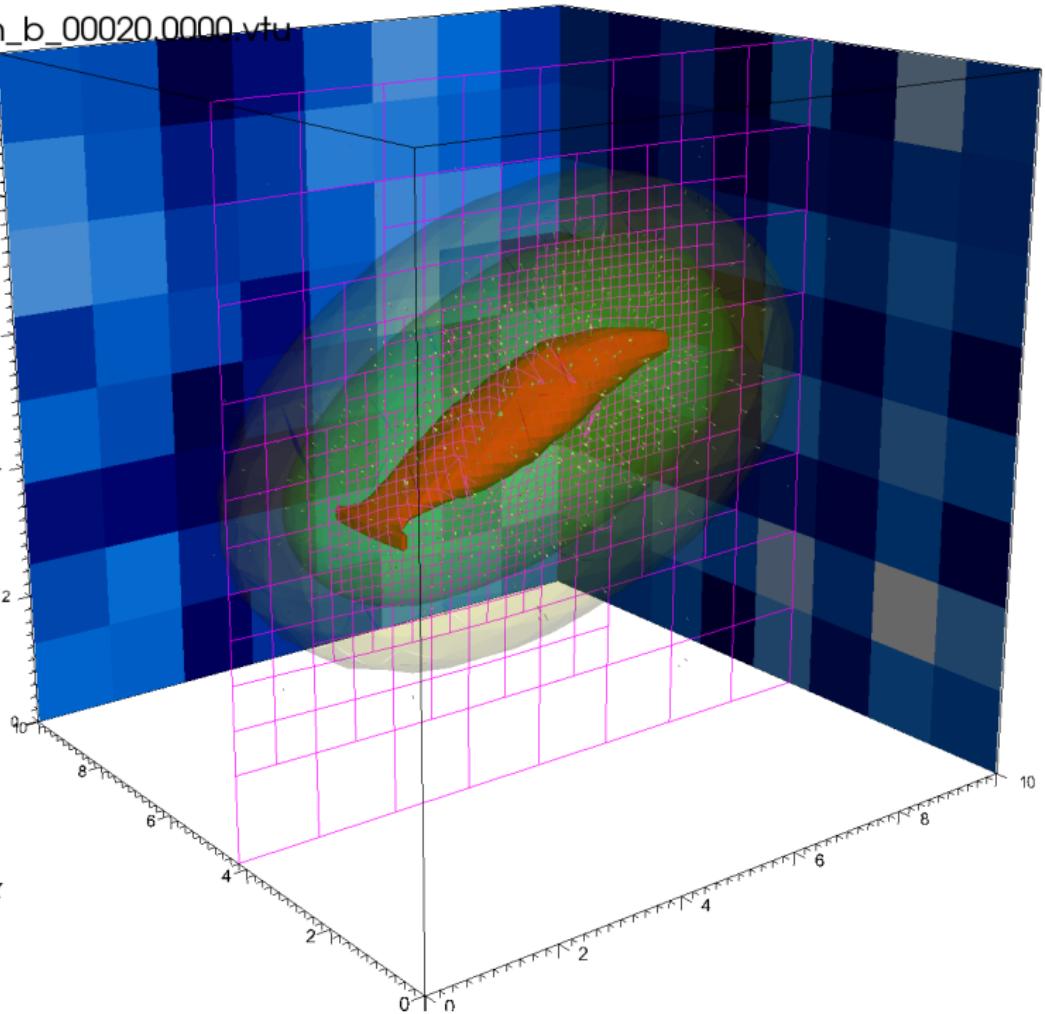
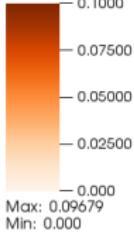


Pseudocolor

Var: emodulus  
— 9.165e+04



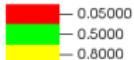
Vector  
Var: dis



# DB: solution\_b\_00021.0000.vtu

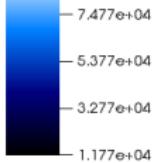
Cycle: 0

Contour  
Var: phi

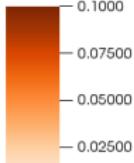


Pseudocolor

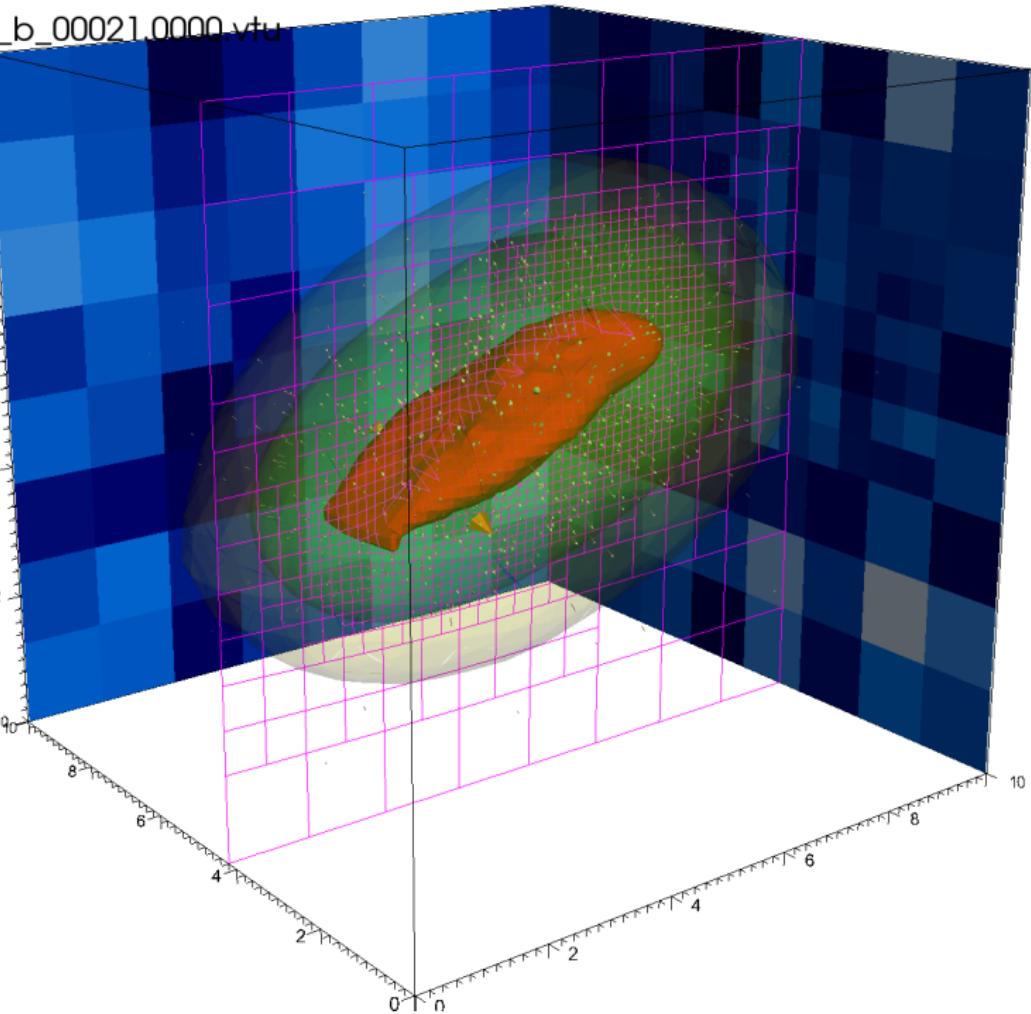
Var: emodulus  
— 9.577e+04



Vector  
Var: dis



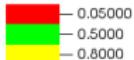
Max: 0.1674  
Min: 0.0000



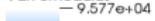
# DB: solution\_b\_00022,0000.vtu

Cycle: 0

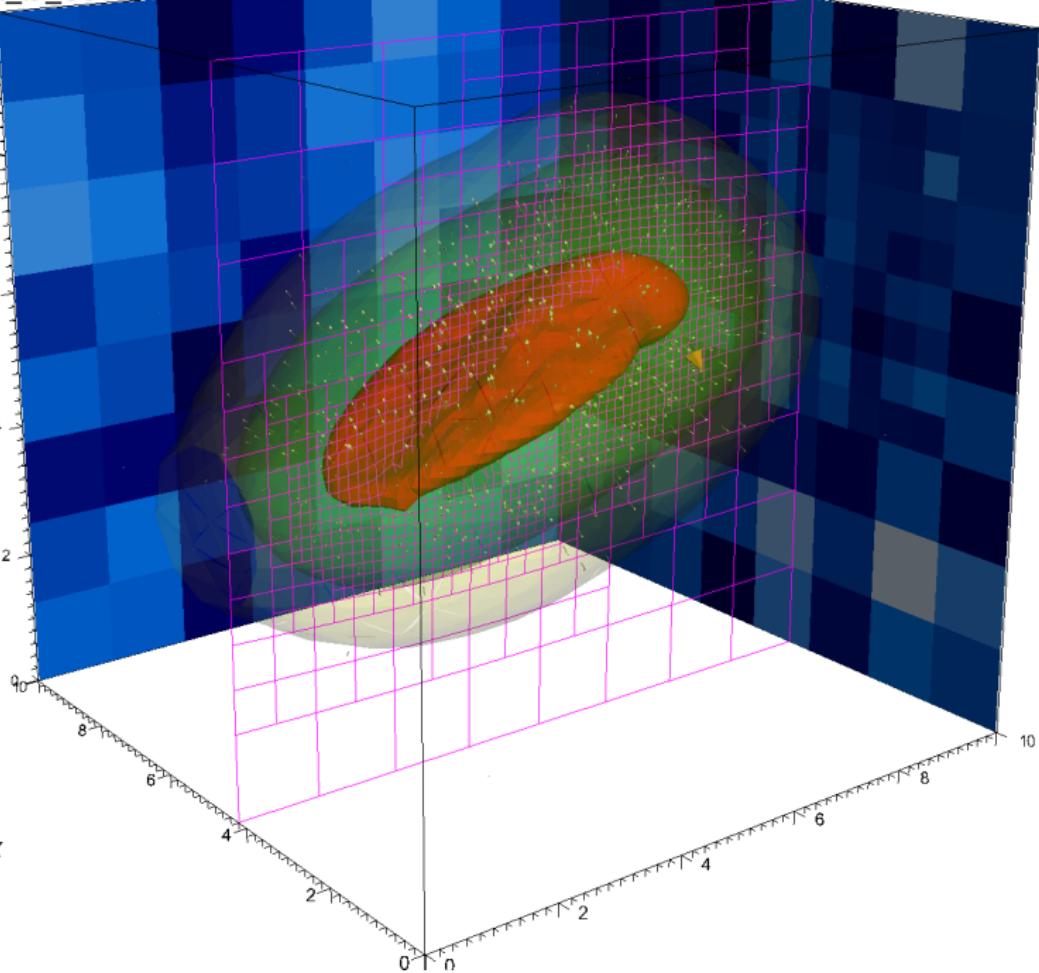
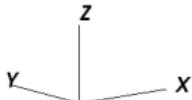
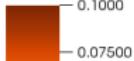
Contour  
Var: phi



Pseudocolor  
Var: emodulus



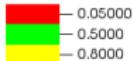
Vector  
Var: dis



# DB: solution\_b\_00023.0000.vtu

Cycle: 0

Contour  
Var: phi



Pseudocolor

Var: emodulus

- 9.659e+04

- 7.538e+04

- 5.418e+04

- 3.297e+04

- 1.177e+04

Max: 9.659e+04

Min: 1.177e+04

Vector  
Var: dis

- 0.1000

- 0.07500

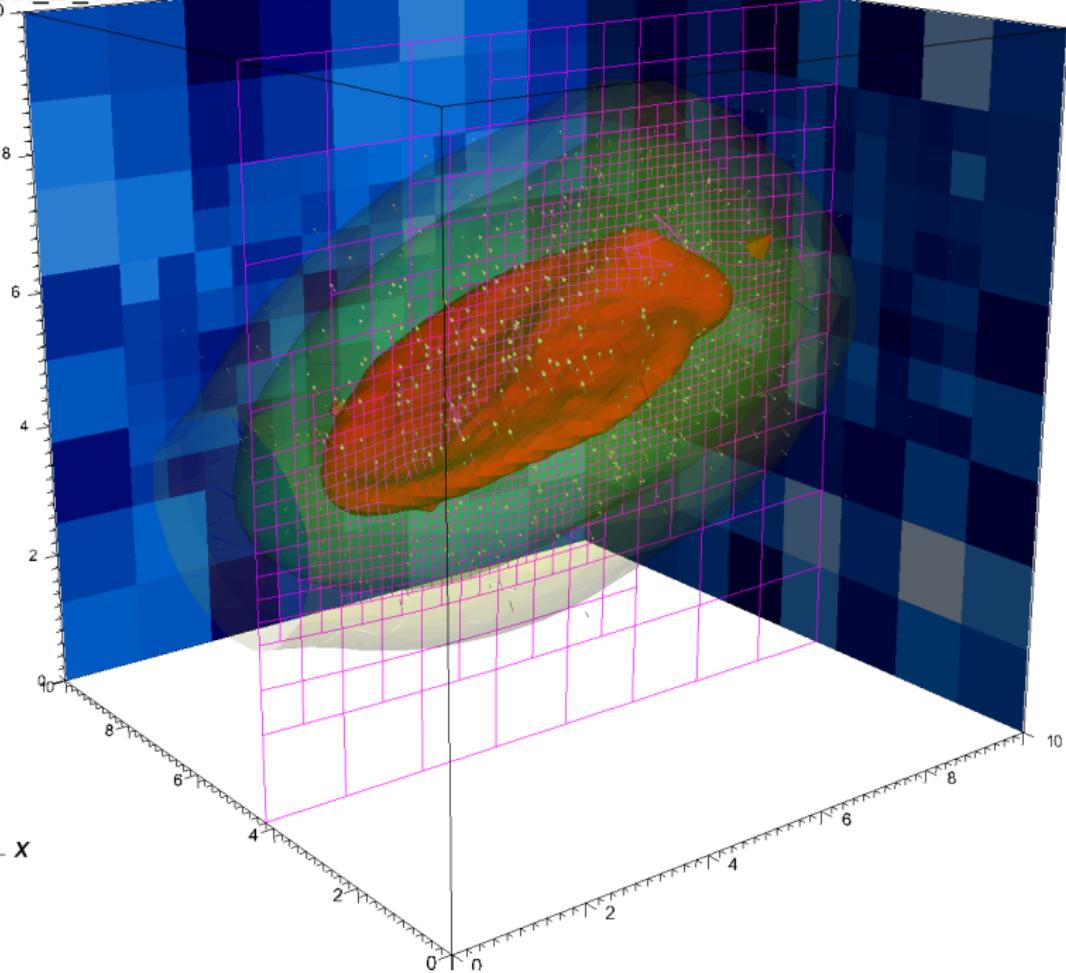
- 0.05000

- 0.02500

- 0.0000

Max: 0.1539

Min: 0.0000



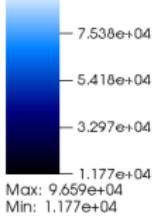
# DB: solution\_b\_00024.0000.vtu

Cycle: 0

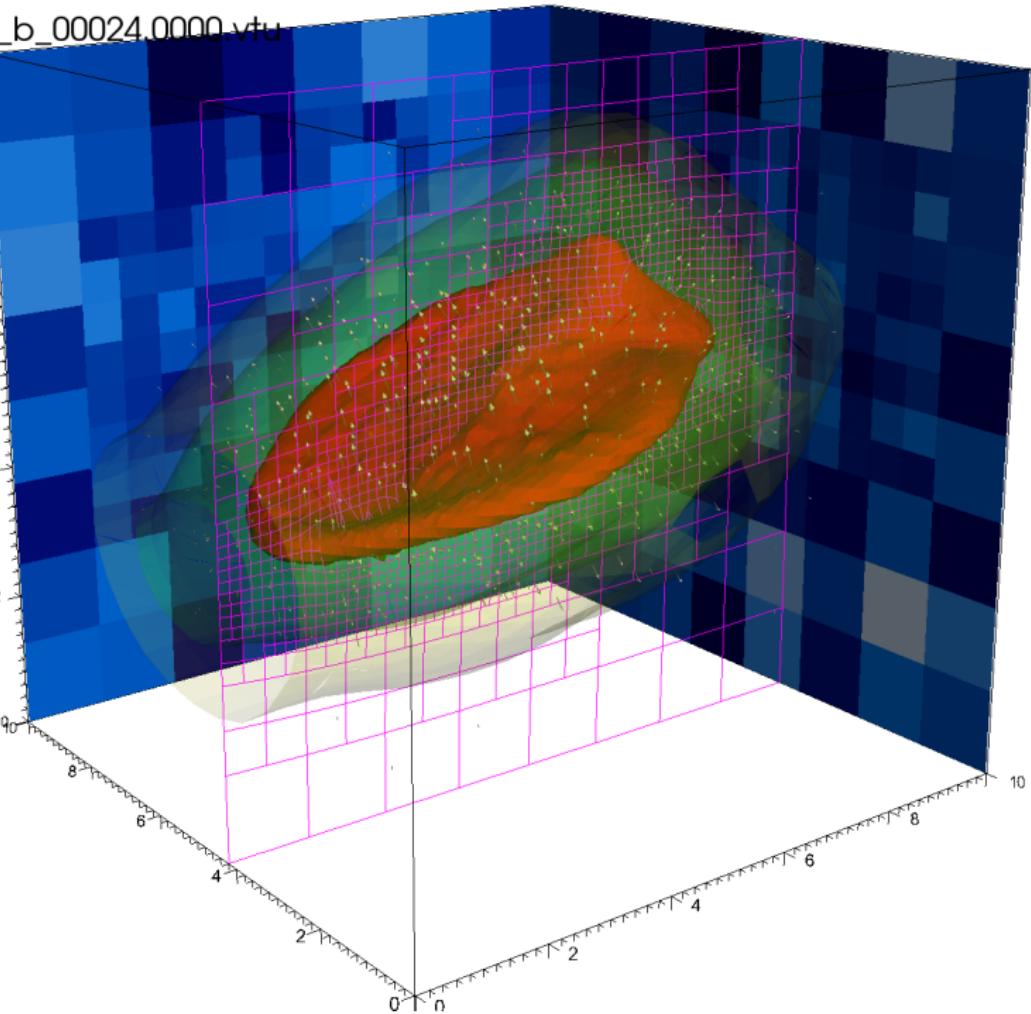
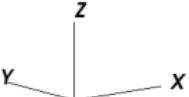
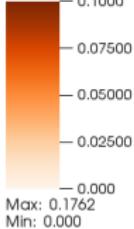
Contour  
Var: phi



Pseudocolor  
Var: emodulus  
- 9.659e+04



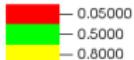
Vector  
Var: dis



# DB: solution\_b\_00025.0000.vtu

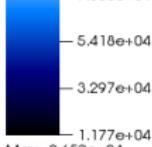
Cycle: 0

Contour  
Var: phi

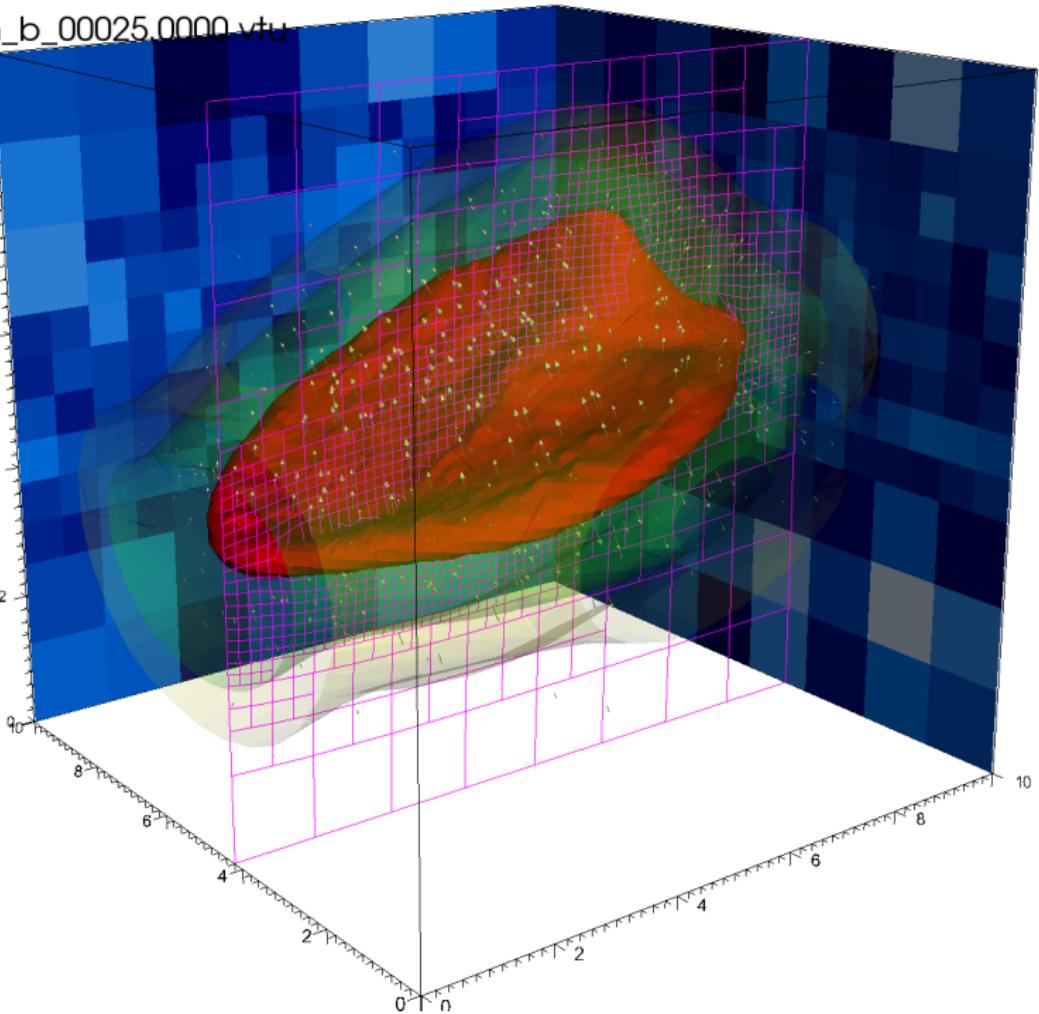
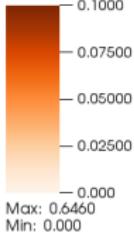


Pseudocolor

Var: emodulus  
— 9.659e+04

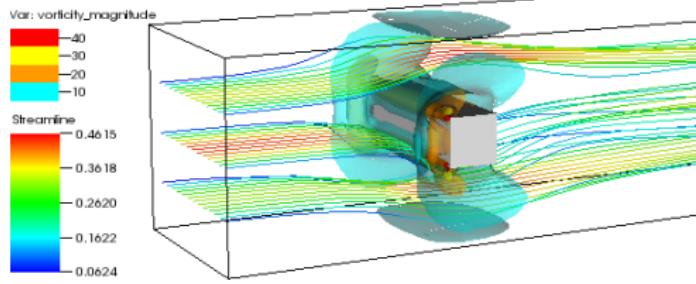
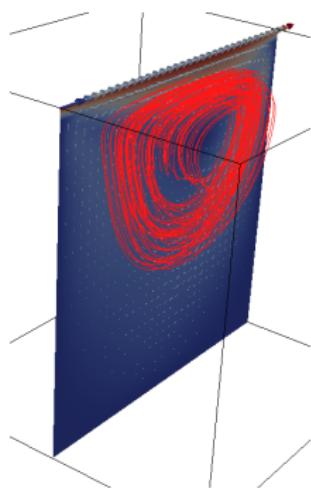


Vector  
Var: dis

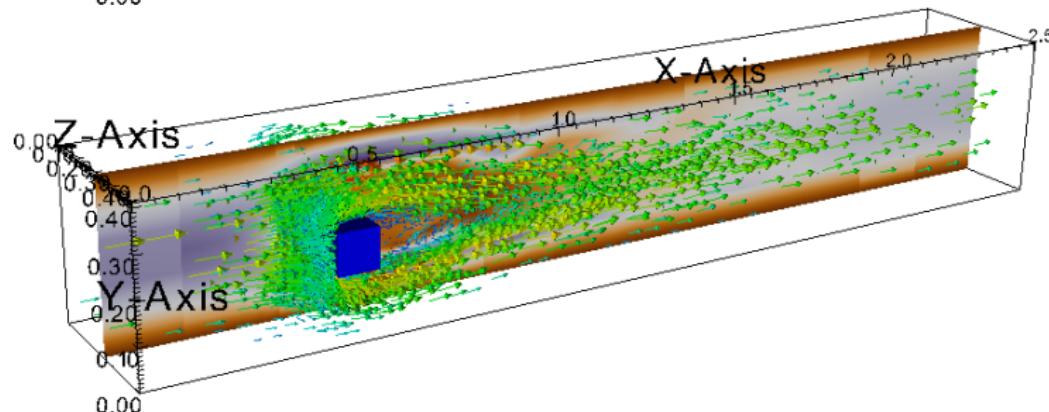
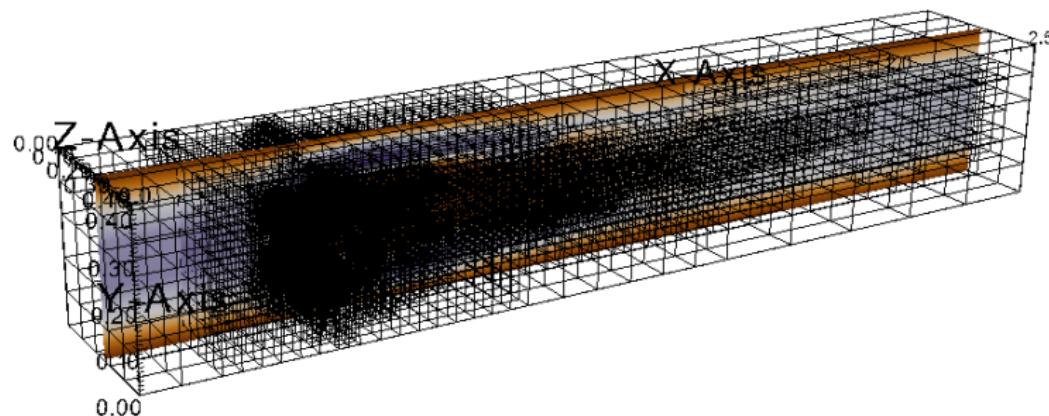


# Incompressible Flow (WIP)

- massively parallel, adaptive Incompressible Navier-Stokes solver
- 100+ million DoFs
- efficient linear solvers for coupled system
- even for stationary problems
- Testbed for different discretizations, e.g. velocity-vorticity
- Code: open sourced soon?

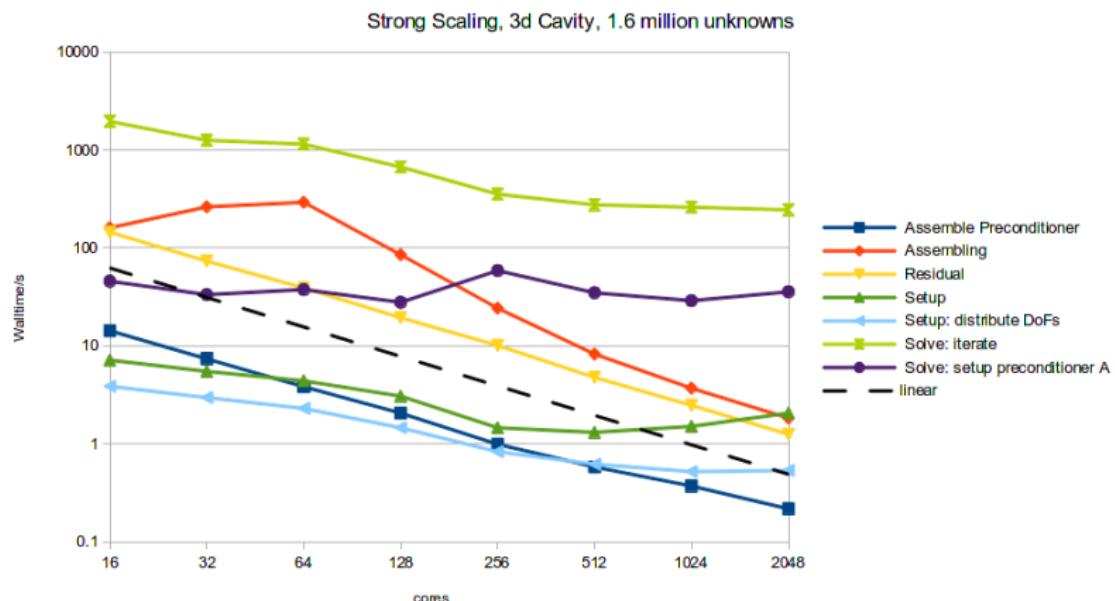


# Flow around Cylinder



# Scaling

strong scaling even for tiny test problems:



# Geometric Multigrid: Goals

linear solver that

- ❖ is generic (different PDEs)
- ❖ is flexible (CG, DG, arbitrary order, ...)
- ❖ supports adaptive mesh refinement
- ❖ is efficient
- ❖ is scalable
- ❖ works on future architectures

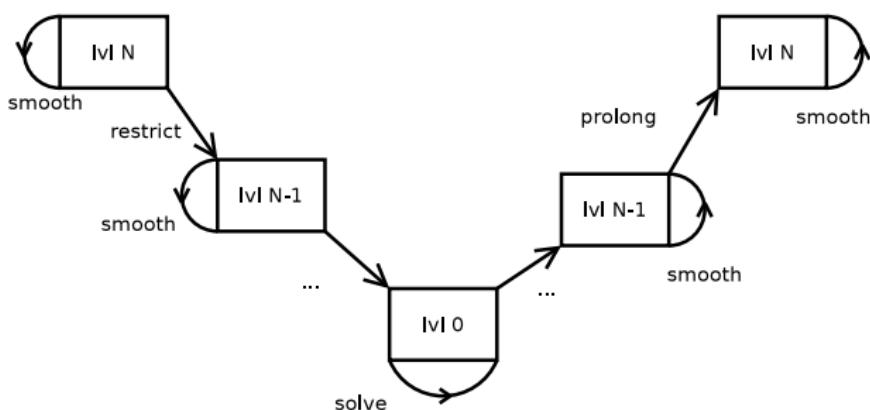
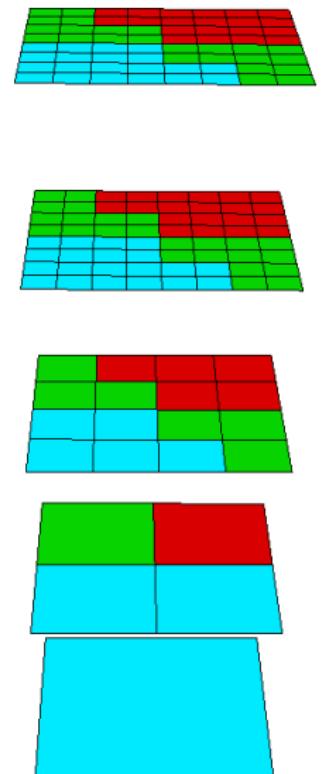
Note: I am ignoring GPUs and I think that is okay.

# Future Architectures

- ❖ likely:
  - ❖ many more cores per node
  - ❖ more flops per memory bandwidth
  - ❖ less memory per core
- ❖ consequences:
  - ❖ higher order elements
  - ❖ avoid building sparse matrices
  - ❖ hybrid parallelization (MPI+multithreading+vectorization)

# Multigrid Intro

- Only method known that is  $O(N)$
- Based on hierarchy of levels
- Operations:  
smooth, restrict, coarse solve, prolong



# Why not algebraic multigrid?

- ❖ always based on sparse matrices
- ❖ construct coarser levels by analyzing matrix structure/entries  
~~> communication overhead!
- ❖ does not take advantage of geometry structure
- ❖ mostly good for poisson type operators
- ❖ can not exploit PDE specifics
- ❖ but: easy to implement and use (Trilinos ML or Hypre)

# Past: geometric Multigrid

- based on local smoothing
- flexible (CG, DG, ...)
- sparse matrices for levels/restriction/prolongation
- serial implementation only



B. Janssen, G. Kanschat. Adaptive Multilevel Methods with Local Smoothing for  $H^1$ - and  $H^{\text{curl}}$ -Conforming High Order Finite Element Methods. *SIAM J. Sci. Comput.*, vol. 33/4, pp. 2095-2114, 2011.

# Past: massively parallel FE

- orange paw prints icon fully distributed, adaptively refined meshes
- orange paw prints icon MPI only
- orange paw prints icon based on sparse matrices, PETSc/Trilinos, AMG
- orange paw prints icon scalable: 10k+ cores, 4+ billion unknowns



W. Bangerth, C. Burstedde, T. Heister, M. Kronbichler. Algorithms and Data Structures for Massively Parallel Generic Finite Element Codes. *ACM Trans. Math. Softw.*, Volume 38(2), 2011.

# Past: matrix free computations

- take advantage of tensor-structure of FE spaces
- using the geometric multigrid framework
- multithreading (Intel TBB)
- (in part explicit) vectorization based on processing n cells at the same time



M. Kronbichler, K. Kormann. A generic interface for parallel cell-based finite element operator application *Computers and Fluids*, vol. 63, pp. 135-147, 2012

# Past, Present, and Future

- ❖ Past, incompatible:
  - ❖ serial GMG
  - ❖ distributed meshes
  - ❖ matrix free computations
- ❖ Now:
  - ❖ Parallel geometric multigrid
  - ❖ Distributed meshes
  - ❖ MPI, based on sparse matrices
- ❖ In the future:
  - ❖ matrix free
  - ❖ hybrid parallel
  - ❖ combine with AMG on coarser level

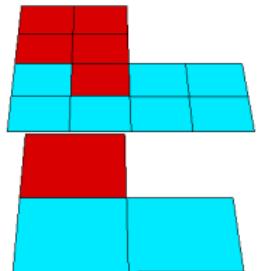
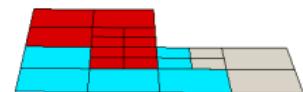
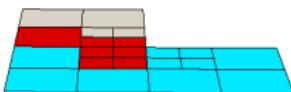
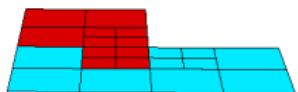
# Idea

- ❖ equivalent to serial MG
- ❖ create distributed transfer and level matrices
- ❖ Need: ownership of cells/dofs on each level

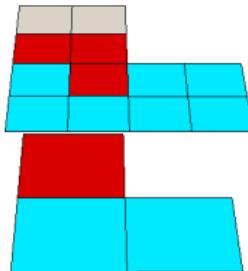
Level cell ownership:

- ❖ simplest idea: owner of first child
- ❖ need to construct ghost layer

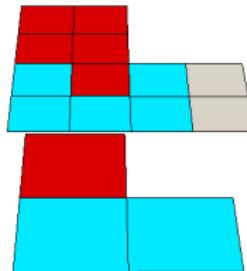
# Distribute Level Cells



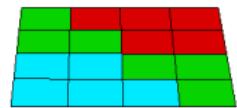
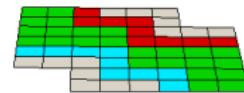
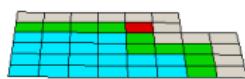
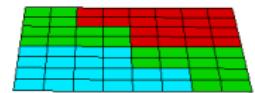
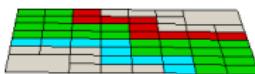
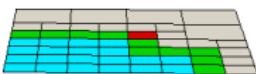
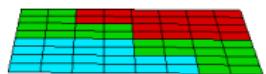
=



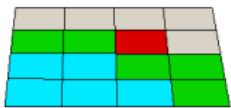
&



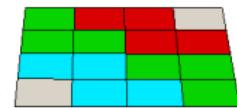
# Distribute Level Cells



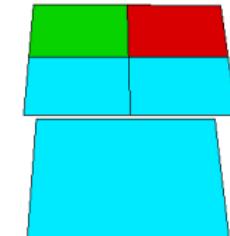
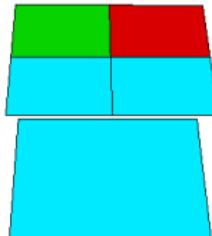
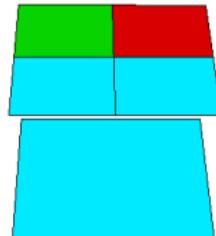
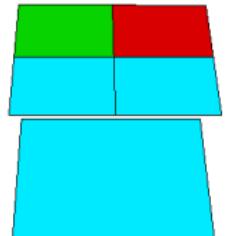
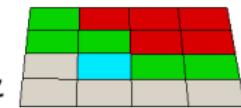
=



&amp;



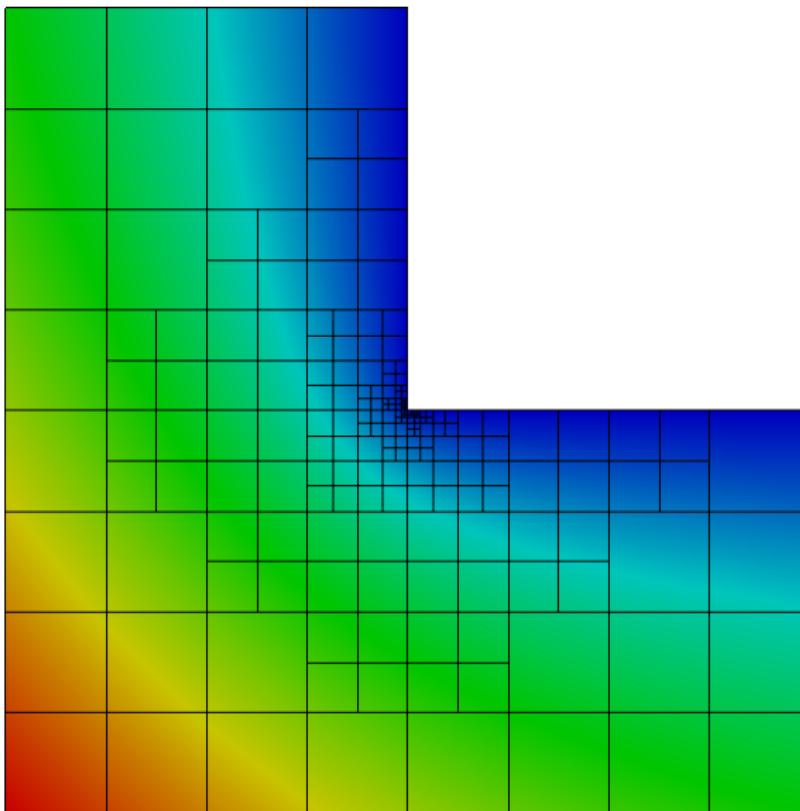
&amp;



# Distribute Level DoFs

- ❖ distribute DoFs locally on each CPU and level
- ❖ communicate with ghost neighbors
- ❖ difficult: consistent constraints (level boundary DoFs, hanging nodes, . . . )

## 🐾 L shape, 2d, laplace problem,DGQ2



ref	cells	lvs	dofs	iterations				
				1 CPU	2 CPUs	4 CPUs	8 CPU	
1	12	2	108	8	8	8	8	
5	45	5	405	9	9	9	9	
10	531	10	4779	9	9	9	9	
11	897	11	8073	9	9	9	9	
12	1521	12	13689	9	9	9	9	
13	2553	13	22977	9	9	9	9	
14	4413	14	39717	9	9	9	9	
15	7533	15	67797	9	9	9	9	
16	13005	16	117045	9	9	9	9	
17	22749	17	204741	9	9	9	9	
18	39063	18	351567	9	9	9	9	

(rel. res: 1e-8, CG, 1 V-cycle, jacobi smoother)

# Balanced?

Locally refined mesh. Number of cells per level:

lvl	proc0	proc1	proc2	proc3	proc4	proc5	proc6
0	1	1	0	1	0	2	0
1	3	2	3	2	2	6	2
2	11	9	9	11	6	25	9
3	41	38	35	46	23	85	36
4	162	145	132	174	87	145	131
5	647	485	484	652	304	474	466
6	152	232	224	160	311	225	232
7	232	304	324	212	456	284	332
8	40	68	76	32	96	40	76

Good enough?

# Performance?

❖ Compare with Trilinos ML:

	GMG	AMG
Setup	2s	2s
Assemble	5s	5s
Setup MG	4s	1.5s
Assemble MG	6s	-
Solve	17s	11s
total Solver	27s	12.5s

Note:

- ❖ Setup MG does some stupid things
- ❖ We go all the way down to 3 cells
- ❖ Naive smoother: Jacobi
- ❖ Better on large scale?

( Triangulation 113304 cells, 20 levels, 1019736 dofs, 4 cores)

# My TODO list

- Test scalability
- fix some parts that do not scale at the moment (involving all-to-all or vector<bool> for DoFs)
- More interesting test problems
- Switch to matrix-free (transfer, smoothing) with Martin's help
- Hybrid parallel experiments
- New assembling technique in deal.II?
- Run on Xeon Phi's?

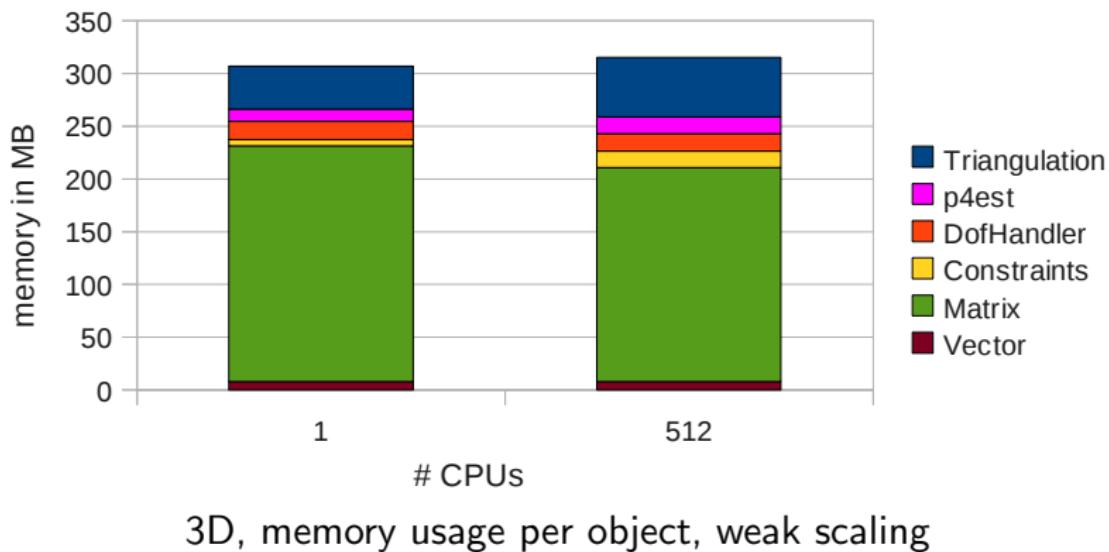
**Thanks for your attention!**



# Hybrid

- hybrid = MPI between nodes, multithreading inside node
- Advantage: save memory (important in the future, see Guido's talk)
- Bottleneck in codes: Preconditioners
- Inside PETSc/Trilinos: preconditioners are not multithreaded  
~~> not worth it today
- But we are ready: multithreading in assembly, etc.

# Test: memory consumption



# What is ASPECT?

ASPECT = Advanced Solver for Problems in Earth's ConvecTion

- ❖ Modern numerical methods
- ❖ Open source, C++: <http://www.dealii.org/aspect/>
- ❖ Based on the finite element library deal.II
- ❖ Supported by CIG
- ❖ Main author



- Bangerth and Heister.  
ASPECT: Advanced Solver for Problems in Earth's ConvecTion, 2012.  
<http://www.dealii.org/aspect/>.
- Kronbichler, Heister and Bangerth.  
High Accuracy Mantle Convection Simulation through Modern Numerical Methods.  
Geophysical Journal International, 2012, 191, 12-29.