The Distributed and Unified Numerics Environment (DUNE)

Oliver Sander, Freie Universität Berlin
DFG Research Center MATHEON

joint work with: Peter Bastian, Markus Blatt, Andreas Dedner,
Christian Engwer, Robert Klöfkorn, Mario Ohlberger

24.1.2008, TU Dresden

DFG research center **matheon**
mathematics for key technologies

Freie Universität Berlin

Dune
Distributed and Unified Numerics Environment

The Dilemma of Finite Element Software

There are many PDE software packages, each with a particular set of features:

- UG: unstructured, multi-element, red-green refinement, parallel
- Alberta: unstructured, simplicial, bisection refinement
- FEAST: block-structured, parallel
- Many more: DiffPack, DEAL, IPARS, libMesh++, ...

Using one package it may be

- either impossible to have a certain feature
- or very inefficient in certain applications

Extension of the feature set is usually very difficult

Reason: Algorithms are implemented on the basis of a particular grid data structure.

Design Concepts

The three DUNE design concepts:

Flexibility: Separate data structures and algorithms

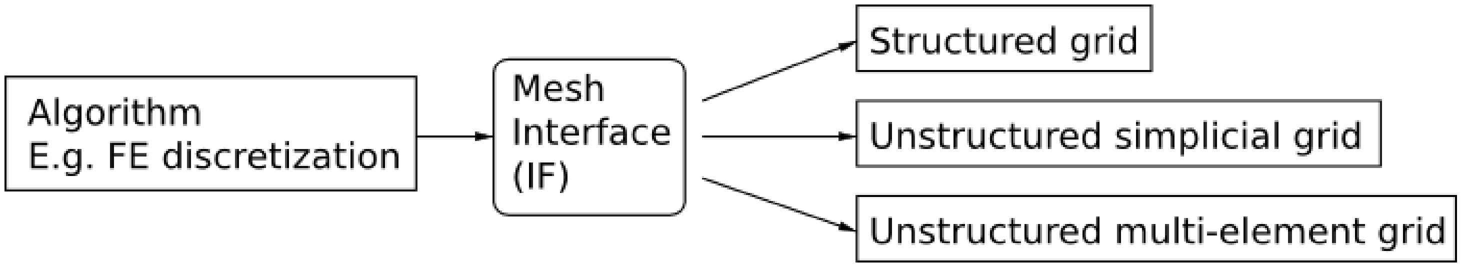
Modularity: Maintainability and software reuse

Efficiency: Low overhead

Concept I: Flexibility

Separate data structure and algorithms

- Determine what algorithms require from a data structure to operate efficiently (`abstract interface')
- Formulate algorithms based in this interface
- Provide different implementations of the interface



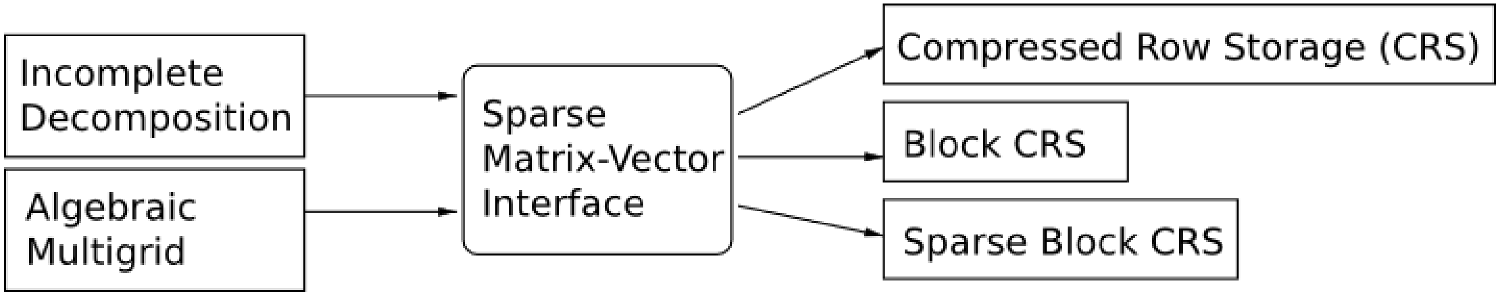
Algorithm
E.g. FE discretization

Mesh
Interface
(IF)

Structured grid

Unstructured simplicial grid

Unstructured multi-element grid



Incomplete
Decomposition

Algebraic
Multigrid

Sparse
Matrix-Vector
Interface

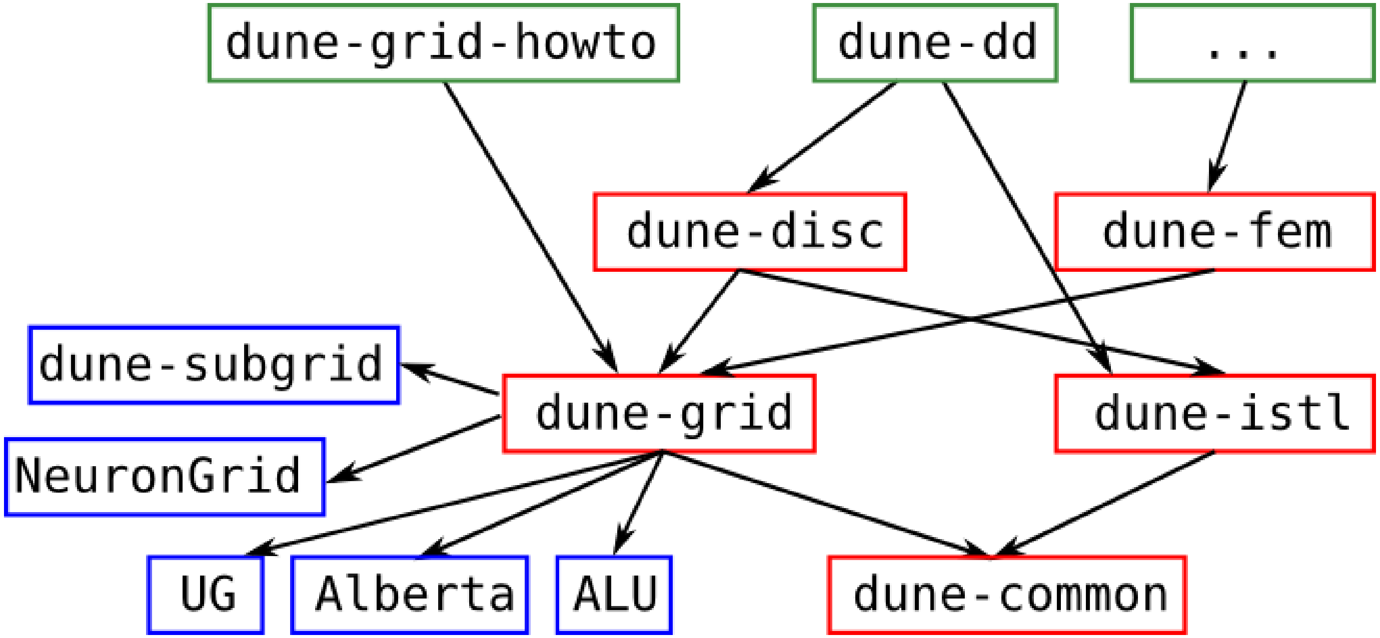
Compressed Row Storage (CRS)

Block CRS

Sparse Block CRS

Concept II: Modularity

Modularity and reuse of existing PDE software

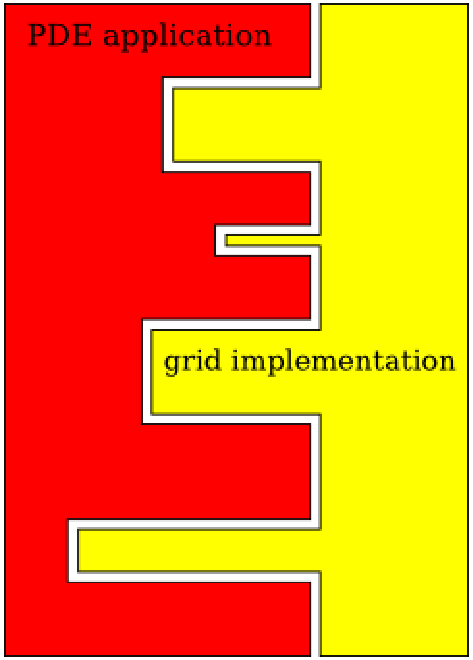


dune-grid-howto
dune-dd
...
dune-disc
dune-fem
dune-subgrid
dune-grid
dune-istl
NeuronGrid
UG
Alberta
ALU
dune-common

(Your contribution is welcome!)

Concept III: Efficiency

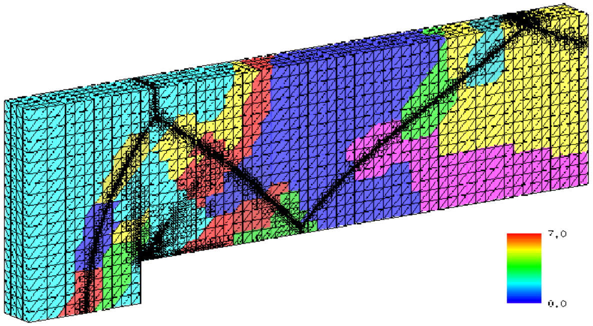
Implementation with generic programming techniques

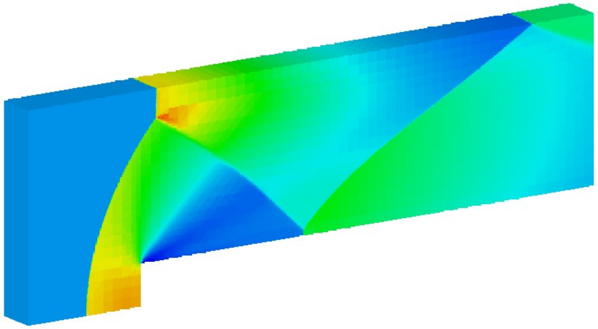


- Compile-time selection of data structures (static polymorphism)
- Compiler generates code for each algorithm / data structure combination
- All optimizations apply, in particular inlining
- Allows interfaces with fine granularity

Concept III: Efficiency

ALUGrid direct vs. ALUGrid through DUNE



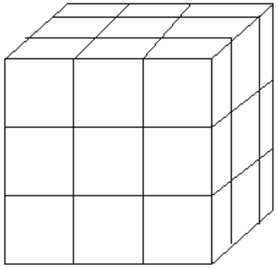


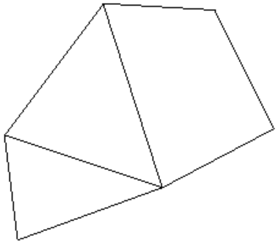
compressible Euler equations

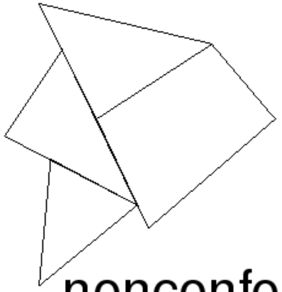
P	flux	evolve	adapt.	total
4	7.8	-5.0	9.3	12
8	7,5	-5.0	9.2	12
16	6.9	-5.0	9.2	11
32	4.9	-5.0	9.1	9

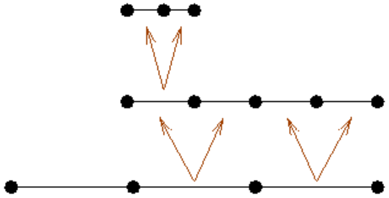
relative performance loss [%]

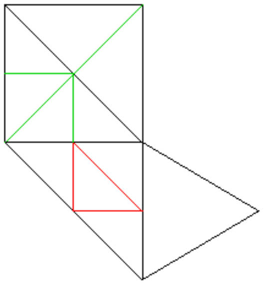
Scope of the Grid Interface

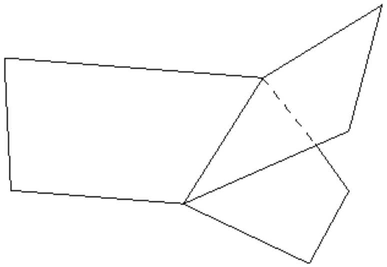

structured, 3D

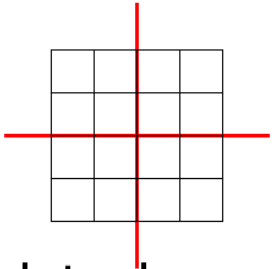

conforming, 2D

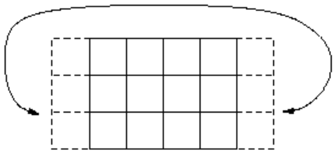

nonconforming

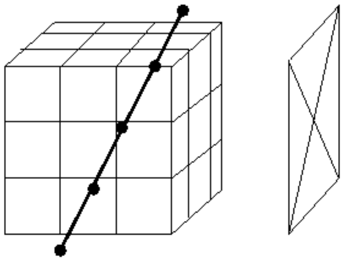

nested, 1D


red-green, bisection


topological spaces


data decomposition


periodic


mixed dimensions

Formal Definition of a Grid

Grids in the DUNE sense are hierarchical!

A hierarchical grid consists of three things:

- A set of entity complexes

$$\mathcal{E} = \{E_0, \ldots, E_k\}$$

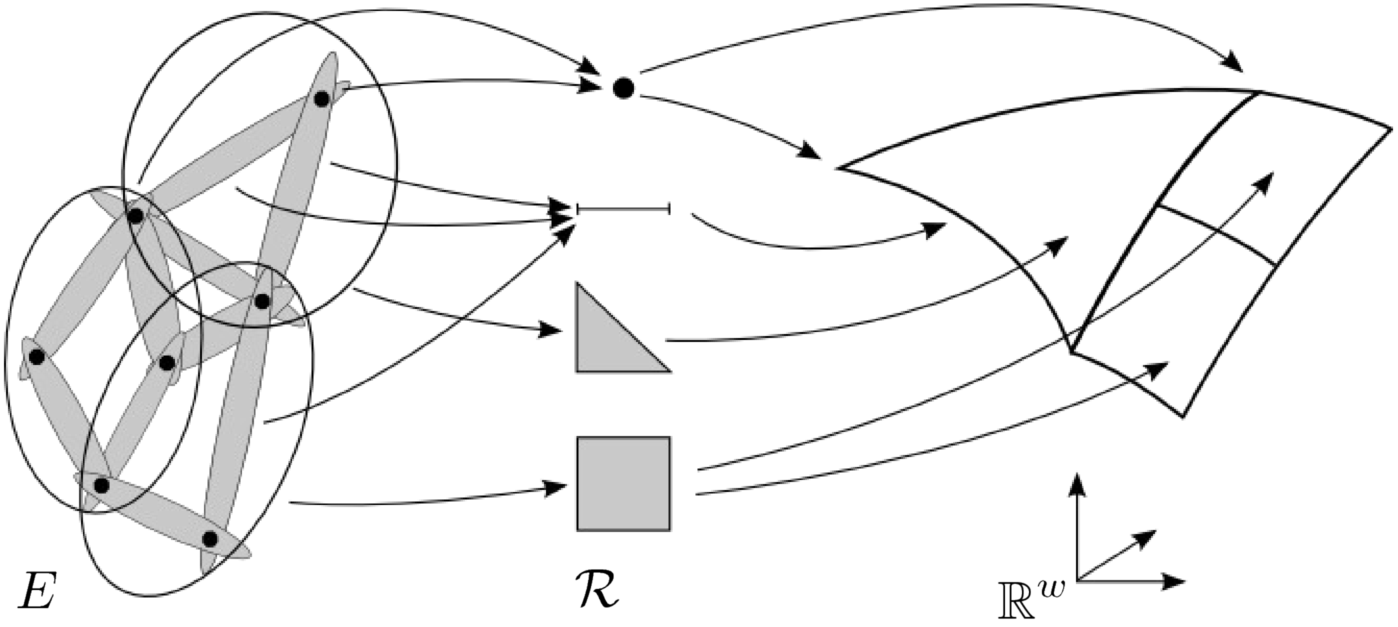
- A set of geometric realizations

$$\mathcal{M} = \{M_0, \ldots, M_k\}$$

- A set of father relations

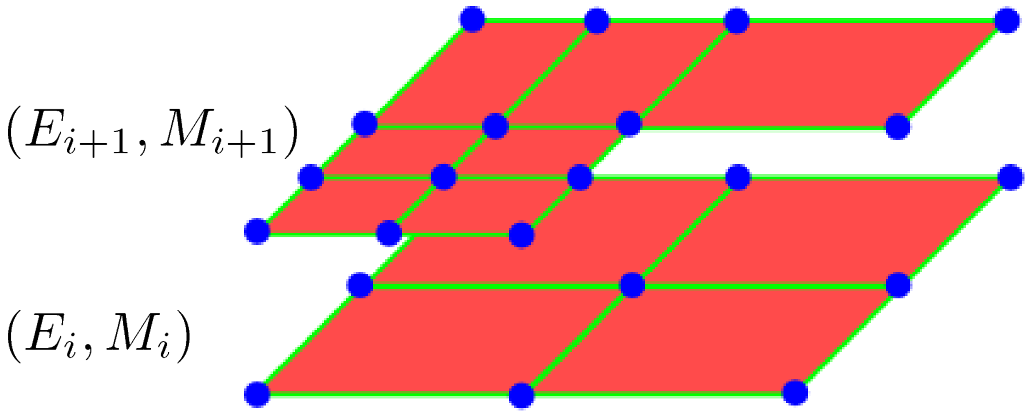
$$\mathcal{F} = \{F_0, \ldots, F_{k-1}\}$$

Entity Complexes and Geometric Realizations

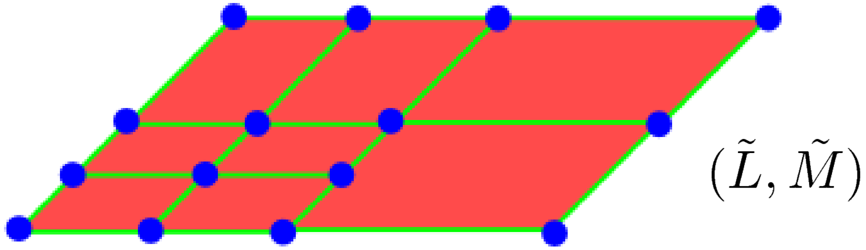


- Entity complex: set system of entities, topological information
- Reference elements: classify entities
- Geometric realization: map from the RE into Euclidean space

Father Relation



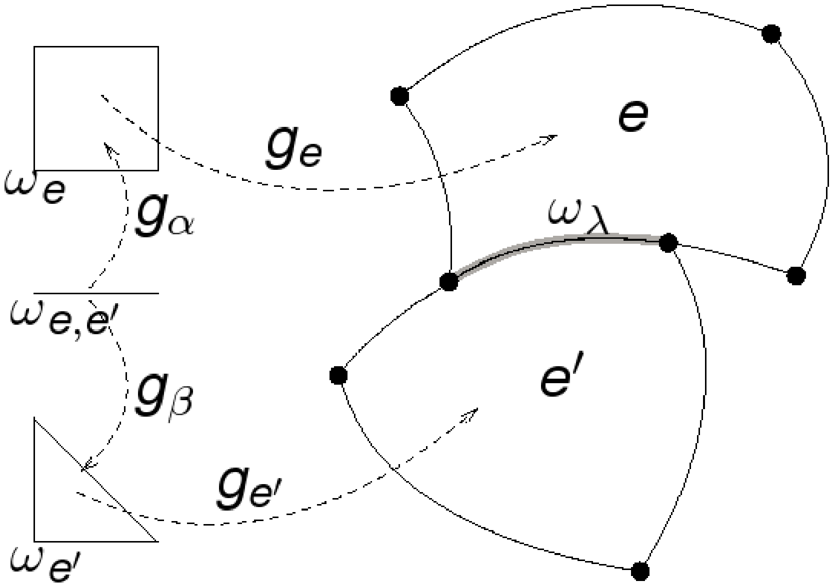
Hierarchical grid



Leaf grid

- Connect two level grids with a father relation
- Only element father relation appears in the interface
- Leaf entities constitute the leaf grid

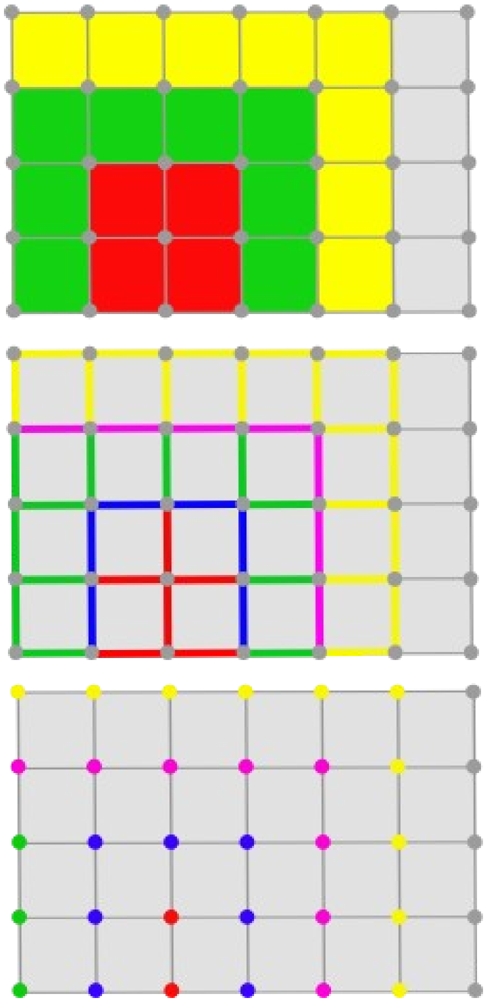
Intersections



- An d-1 dimensional point set shared by two elements.
- Described by transformations
- from a reference element
- Arbitrary nonconforming
- intersections can be handled.
- Leaf- and level-wise intersections

- Intersections with the domain
- boundary and the processor boundary

Parallel Data Decomposition



- Grid is mapped to $\mathcal{P} = \{0, \ldots, P-1\}$.
- $E = \bigcup_{p \in \mathcal{P}} E|_p$ possibly overlapping.
- $\pi_p : E|_p \to$ "partition type".
- For codimension 0 there are three partition types:
  - *interior*: Nonoverlapping decomposition.
  - *overlap*: Arbitrary size.
  - *ghost*: Rest.
- For codimension > 0 there are two additional types:
  - *border*: Boundary of interior.
  - *front*: Boundary of interior+overlap.
- Allows implementation of overlapping and nonoverlapping DD methods.

Index Sets

- Grid and data are totally decoupled
- Grid entities only provide indices

- Level index: consecutive, starting from zero for all entities of a given dimension on a given level

  → index arrays

- Leaf index: consecutive, starting from zero for all entities of a given dimension on the leaf grid

  → index arrays

- Persistent index: nonconsecutive, does not change during grid modifications (refinement / load balancing)

  → index associative arrays

Implementation

- Mathematical definition translates directly into C++ classes
- Implementations using wrapper and engine classes
- Access to entities by STL-style iterators:
  `LevelIterator`, `LeafIterator`, `HierarchicIterator`, `IntersectionIterator`
- Arbitrary sets of grids can coexist in the same application
- Currently available implementations:
  `AlbertaGrid`, `ALUGrid`, `OneDGrid`, `SGrid`, `UGGrid`, `YaspGrid`
- GNU AutoTools build system
- Runs on most flavours of Unix
- Licence: LGPL + linking exception
- Surprisingly easy to use!

Code Example: Grid Creation

Create a structured grid

```
const int dim =3;
typedef Dune :: SGrid < dim , dim > GridType;
Dune :: FieldVector < int , dim > N (3);
Dune :: FieldVector < GridType :: ctype , dim > L (-1.0);
Dune :: FieldVector < GridType :: ctype , dim > H ( 1.0);
GridType grid (N, L, H);
```

Create a UGGrid from an AmiraMesh file

```
const int dim =3;
typedef Dune :: UGGrid < dim > GridType;
GridType grid;
Dune :: AmiraMeshReader<GridType>::read(grid, “filename”);
```

Under discussion: interface for unstructured grid creation

Code Example: Grid Traversal

Iterate over all elements on the leaf grid

```
typedef GridType :: Codim <0>:: LeafIterator ElementLeafIterator;

for ( ElementLeafIterator it = grid . template leafbegin <0>();
       it != grid . template leafend <0>(); ++it )
  {
     std :: cout << " visiting element which is a " << it -> type ()
                       << std :: endl ;
  }
```

Iterate over all vertices on the leaf grid

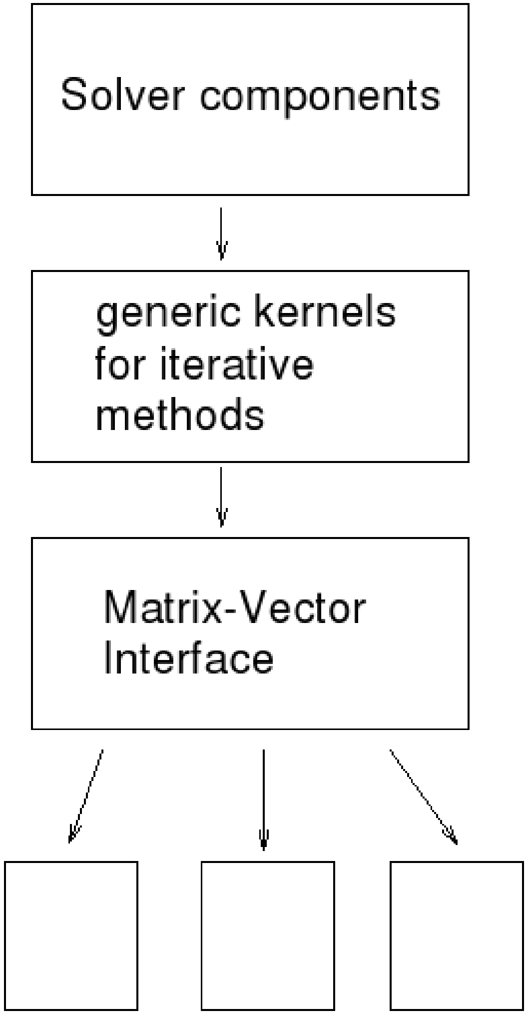
```
typedef GridType :: Codim <dim> :: LeafIterator VertexLeafIterator;

for ( VertexLeafIterator it = grid . template leafbegin <dim>();
       it != grid . template leafend <dim>(); ++it )
  {
     std :: cout << " visiting vertex at " << it -> geometry ()[0]
                       << std :: endl;
  }
```

Code Example: Quadrature

Integrate a function *f* over an element *it

```
Dune :: GeometryType gt = it - > type ();

const Dune :: QuadratureRule < double , dim >&
     rule = Dune :: QuadratureRules < double , dim >:: rule ( gt , p );

double result =0;

for ( int i = 0; i < rule.size(); i++)
  {
     FieldVector<double,dim>
         globalPosition = it -> geometry (). global (rule[i] . position ())
     double fval   = f (globalPosition);
     double weight = rule[i] . weight ();
     double detjac = it->geometry(). integrationElement (rule[i].position());
     result += fval * weight * detjac ;
  }
```

Linear Algebra: dune-istl


Solver components

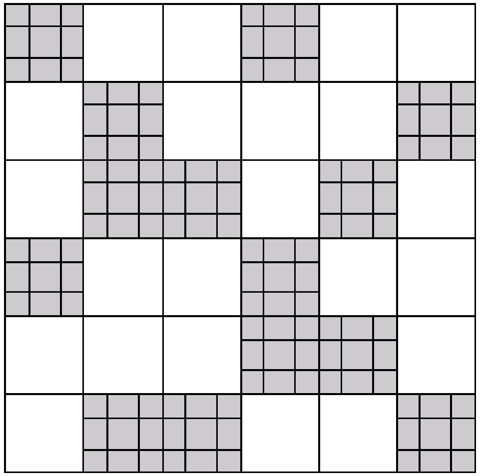
generic kernels for iterative methods

Matrix-Vector Interface

- There are already template libraries for linear algebra: MTL/ITL
- Existing libraries cannot efficiently use (small) structure of FE-Matrices
- Solver components: Based on operator concept, Krylov methods, (A)MG preconditioners
- Generic kernels: Triangular solves, Gauß-Seidel step, ILU decomposition
- Matrix-Vector Interface: Support recursively block structured matrices
- Various implementations of the interface are available

dune-istl is completely independent of dune-grid!

Block Structure in FE Matrices

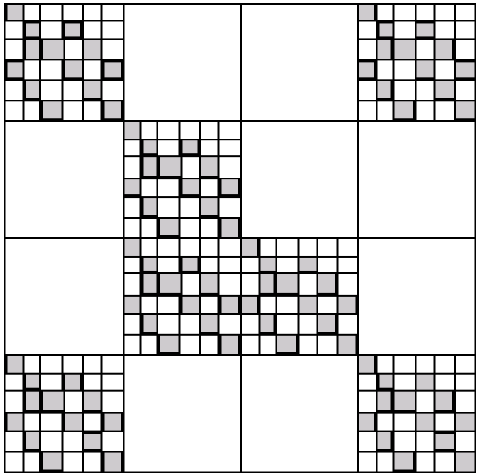


sparse block matrix

blocks are dense

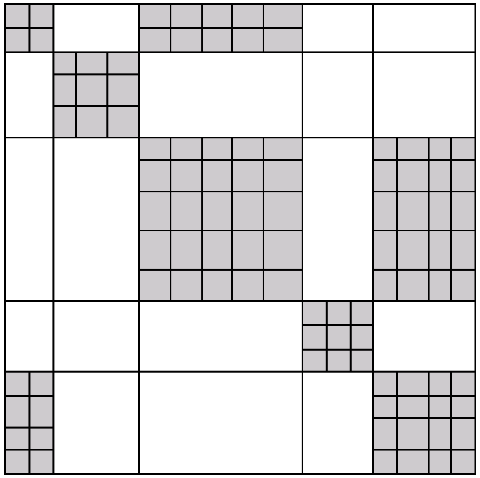
blocks have fixed size

DG fixed p



blocks are sparse

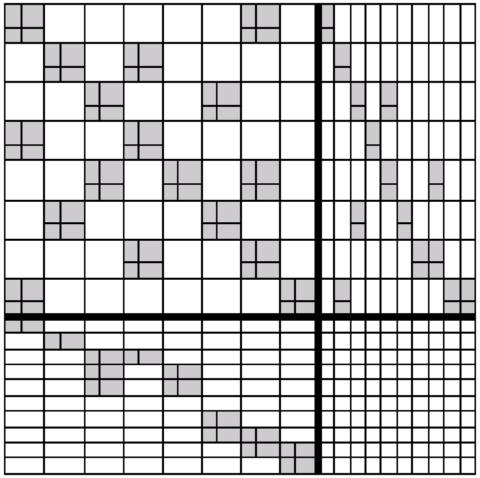
diffusion-reaction systems



blocks are dense

blocks have variable size

DG hp version



2x2 block matrix

each block is sparse

Taylor-Hood elements

Dune

Distributed and Unified Numerics Environment

Example Definitions

- A vector containing 20 blocks where each block contains two complex numbers using **`double`** for each component:

```
typedef FieldVector<complex<double>,2> MyBlock;
BlockVector<MyBlock> x(20);
x[3][1] = complex<double>(1,-1);
```

- A sparse matrix consisting of sparse matrices having scalar entries:

```
typedef FieldMatrix<double,1,1> DenseBlock;
typedef BCRSMatrix<DenseBlock> SparseBlock;
typedef BCRSMatrix<SparseBlock> Matrix;
Matrix A(10,10,40,Matrix::row_wise);
... // fill matrix
A[1][1][3][4][0][0] = 3.14;
```

Vector and Matrix Interface

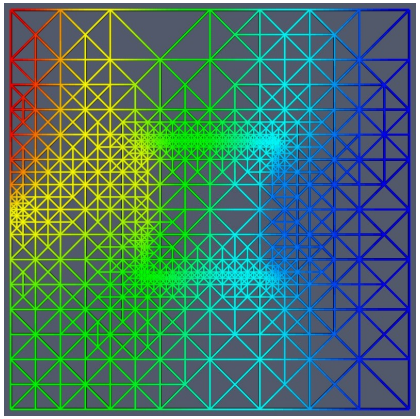
Mainly taken from sparse BLAS

- Vector
  - Is a one-dimensional container
  - Sequential access
  - Random access
  - Vector space operations: Addition, scaling
  - Scalar product
  - Various norms
  - Sizes
- Matrix
  - Is a two-dimensional container
  - Sequential access using iterators
  - Random access
  - Organization is row-wise
  - Mappings $y = y + Ax; y = y + A^T x; y = y + A^H x;$
  - Solve, inverse, left multiplication
  - Various norms
  - Sizes

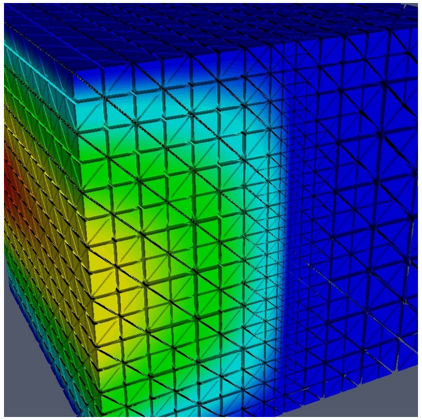
Code Example: Block Gauß-Seidel

```
for (int i=0; i<x->size(); i++) {

    VectorBlock r, v;

    typedef MatrixType::row_type RowType;
    const RowType& row = matrix[i];

    typedef typename RowType::ConstIterator ColumnIterator;

    r = rhs[i];

    for (ColumnIterator cIt=row.begin(); cIt!=row.end(); ++cIt)
        // r_i -= A_ij x_j
        cIt->mmv(x[cIt.index()], r);

    // Compute v = A_{i,i}^{-1} r[i]
    mat[i][i].solve(v, r);

    // Add correction
    x[i] += v;

}
```

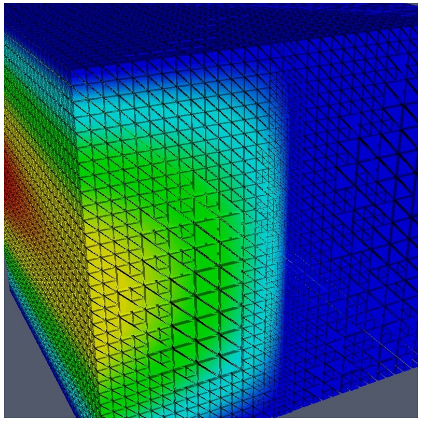
Example: Poisson Problem



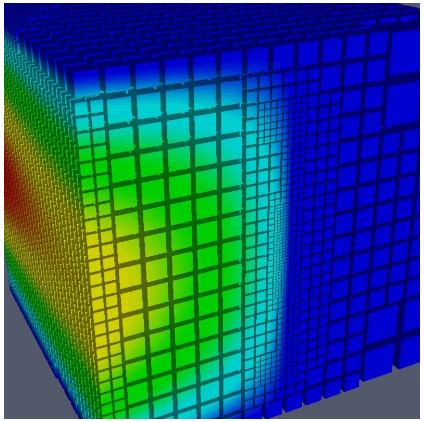
AlbertaGrid, 2d



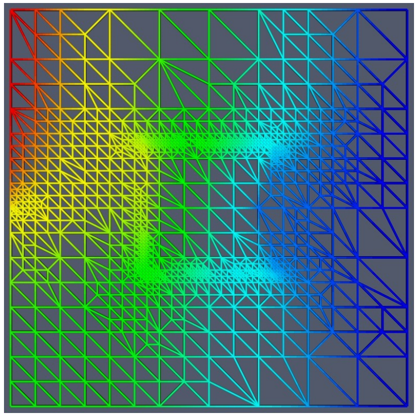
AlbertaGrid, 3d



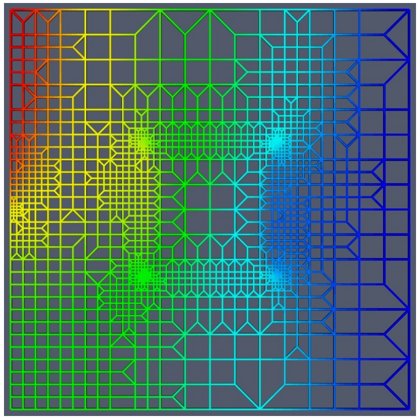
AluSimplexGrid, 3d



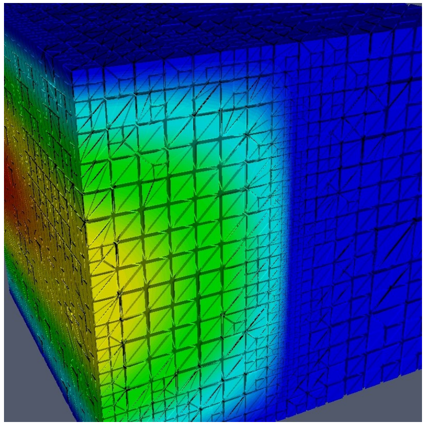
AluCubeGrid, 3d



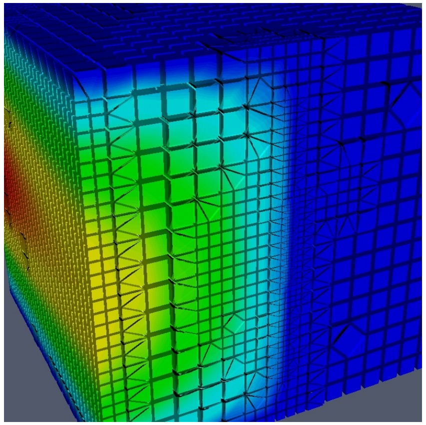
UGGrid, 2d, simplices



UGGrid, 2d, cubes

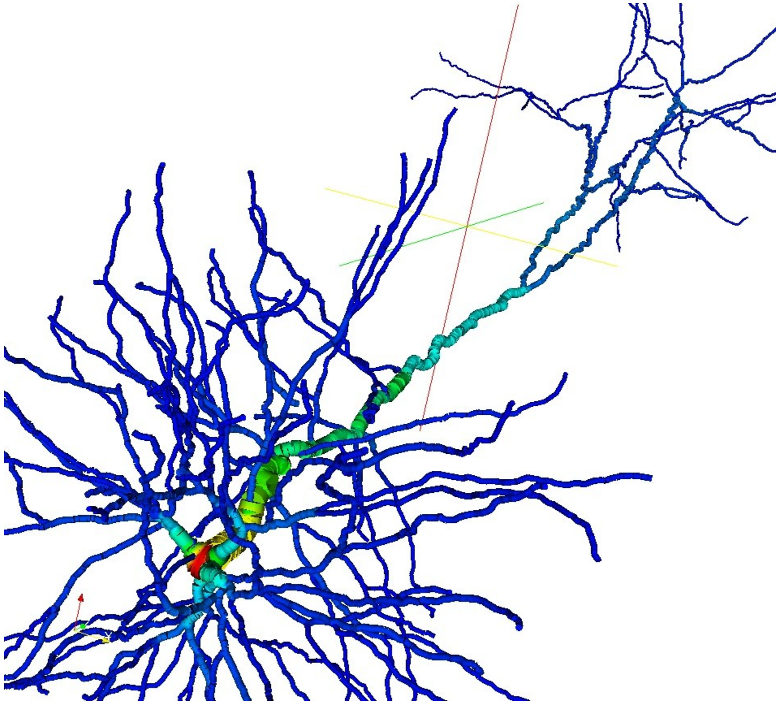


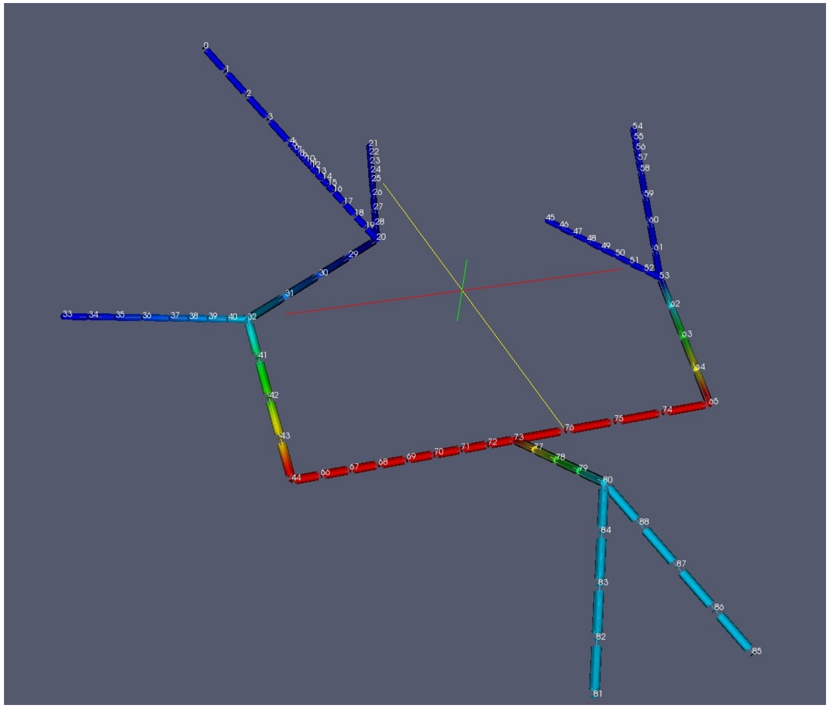
UGGrid, 3d, simplices



UGGrid, 3d, cubes

Example: Neuron Grid

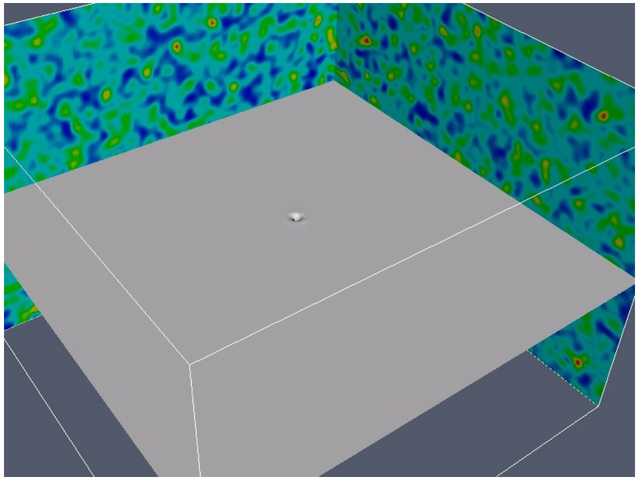


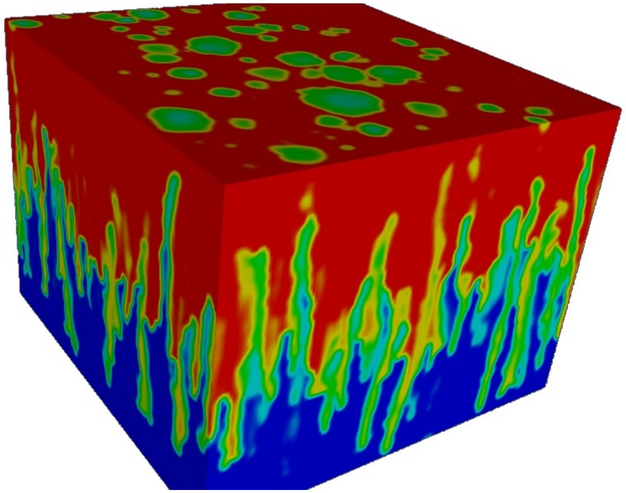


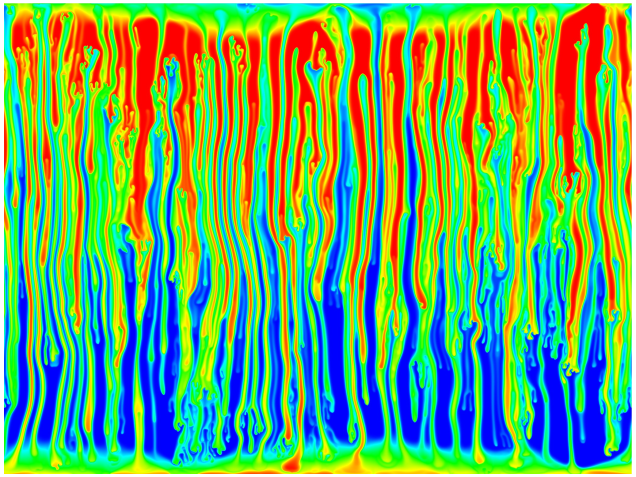
- Dendritic tree of L5 B pyramidal neuron (reconstruction by Christiaan de Kock, MPIMF, Heidelberg)
- NeuronGrid simulator (Stefan Lang, Olaf Ippisch)

Example: Parallel Computing

Density-driven flow (P. Bastian)

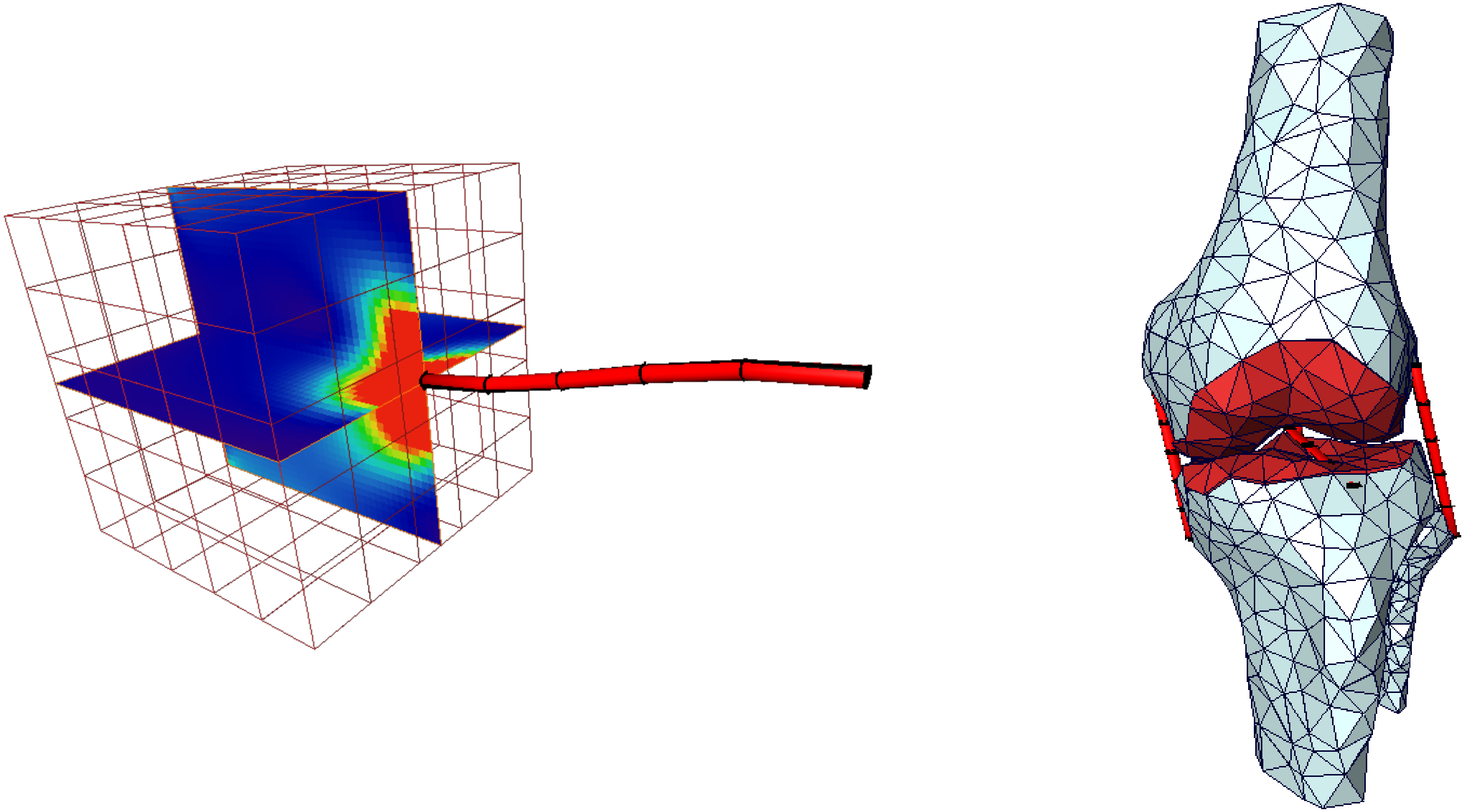






- cell-centered finite volume scheme
- matrix-free implementation
- YaspGrid, 8e8 cells, 384 processors
- 9000 timesteps, 3 days running time

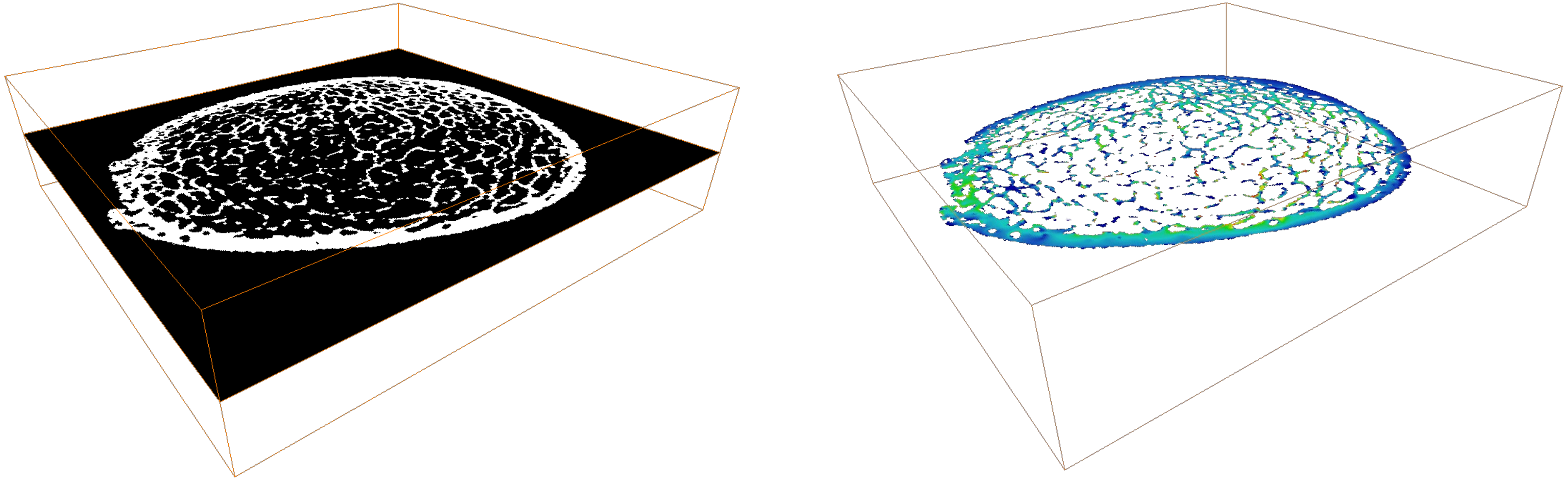
Example: Multidimensional Coupling



- Couple 3d linear elasticity with Cosserat rods
- Left: 1 UGGrid, 1 OneDGrid
- Right: 5 UGGrids, 4 OneDGrids

Example: dune-subgrid

(C. Gräser, S. Prohaska, Z. Ritter, O. Sander.)



- Axial compression of 9mm section of human distal radius
- Subgrid of uniform grid (YaspGrid)
- Uniform grid: 449x422x110, Subgrid: ca. 4.5e6 elements (22%)
- Geometric multigrid with CFE coarse grid spaces

Further Information

- P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander, `*A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework*', Matheon Preprint 403, submitted to `Computing'

- P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander, `*A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Implementation and Tests in DUNE*', Matheon Preprint 404, submitted to `Computing'

http://www.dune-project.org