

Flexibility and High Performance in Numerical Software

P. Bastian

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Universität Heidelberg

Schloß Dagstuhl, 14.2.2006

Joint work with:

M. Blatt C. Engwer R. Klöfkorn
M. Ohlberger O. Sander

Peter.Bastian@iwr.uni-heidelberg.de
<http://hal.iwr.uni-heidelberg.de/dune/>

- Application developers point of view.
- Software life-cycle much longer than hardware life-cycle.
- Physics and algorithms getting always more complex.
- Software reuse is mandatory.
- Annotations — about performance, too low level?
- New languages — too high level (and need something NOW)?
- Solution: Application domain specific frameworks.
- Domain of interest here: PDE numerics.

- 1 DUNE - Distributed Unified Numerics Environment
- 2 Abstractions for Finite Element Grids
- 3 Abstractions for Linear Algebra and Solvers
- 4 Scalability of a Multigrid Implementation
- 5 Conclusions

1 DUNE - Distributed Unified Numerics Environment

2 Abstractions for Finite Element Grids

3 Abstractions for Linear Algebra and Solvers

4 Scalability of a Multigrid Implementation

5 Conclusions

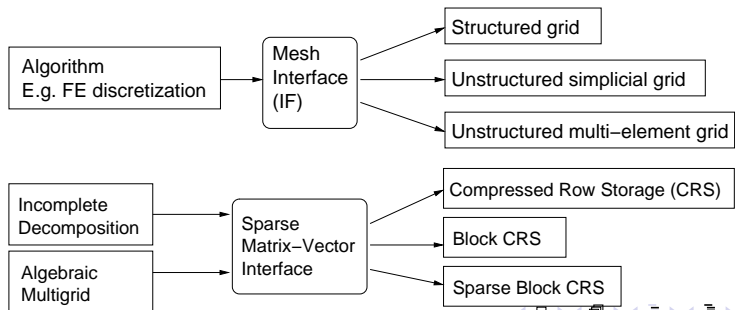
The Problem with Finite Element Software

- There are many PDE software packages, each with a particular set of features:
 - IPARS: block structured, parallel, multiphysics.
 - Alberta: simplicial, unstructured, bisection refinement.
 - UG: unstructured, multi-element, red-green refinement, parallel.
 - QuocMesh: Fast, on-the-fly structured grids.
 - Many more: DiffPack, DEAL, libMesh++, ...
- Using one framework, it
 - might be either impossible have a particular feature,
 - or very inefficient in certain applications.
- Extension of the feature set is usually hard

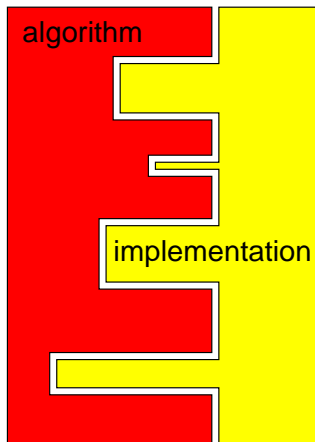
Reason: Algorithms are implemented on the basis of a particular data structure

Separate data structures and algorithms.

- Programming with concepts
 - Determine what algorithms require from a data structure to operate efficiently (“concepts”, “abstract interfaces”)
 - Formulate algorithms based on these interfaces
 - Provide different implementations of the interface

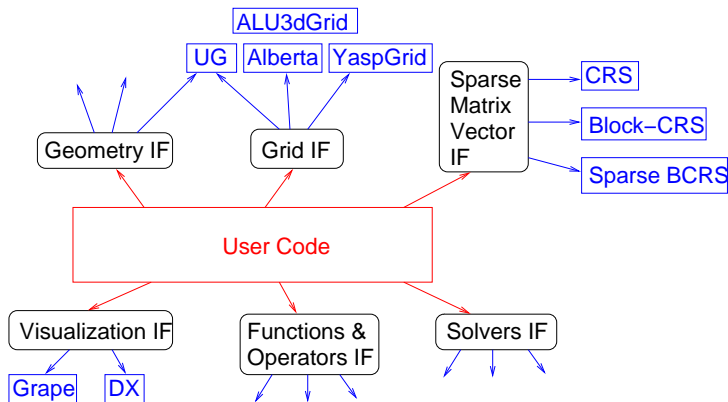


Implementation with generic programming techniques.



- Compile-time selection of data structures (static polymorphism).
- Compiler generates code for each algorithm-data structure combination.
- All optimizations apply, in particular function inlining.
- Allows use of interfaces with fine granularity.
- Concept has been around for some time:
 - Standard Template Library (1998): Containers, Blitz++, MTL/ITL, GTL, ...
 - Thesis of Gundram Berti (2000): Concepts for grid based algorithms.

Reuse existing finite element software.

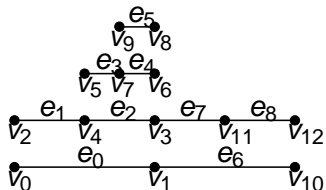
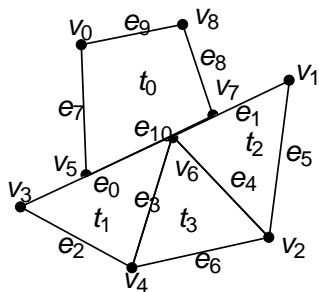


- Efficient integration of existing FE software.
- Developed by groups in Berlin, Freiburg and Heidelberg

- 1 DUNE - Distributed Unified Numerics Environment
- 2 Abstractions for Finite Element Grids**
- 3 Abstractions for Linear Algebra and Solvers
- 4 Scalability of a Multigrid Implementation
- 5 Conclusions

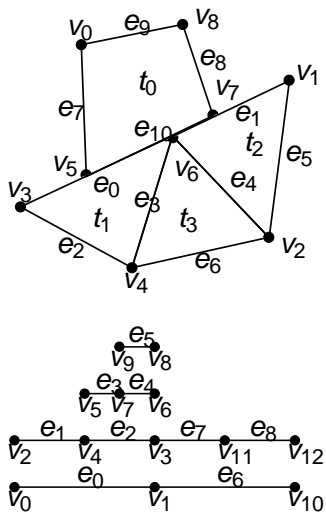
There is great variability in finite element grids:

- *Structured grid*: $O(1)$ memory, on-the-fly generation.
- *Unstructured grid*: different element types
- *Conforming/nonconforming grids*
- *Local mesh refinement*: nested r. vs. point insertion, conforming r. (red/green, bisection) vs. nonconforming r. (hanging nodes).
- *Grids on manifolds*: shells, fractures (2D in 3D), wells, neural networks (1D in 3D).
- *Dimension independence*: Uniform access to entities of all codimensions.
- *Parallel data decomposition*: Overlapping, nonoverlapping, dynamic load balancing.
- *Coupled grids*: Overlapping, nonoverlapping, mortars.
- *Other issues*: Sparse grids, periodicity.

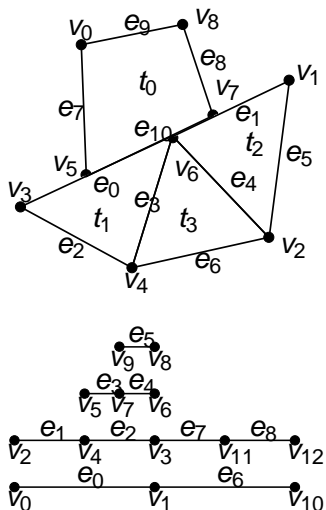


- Describe a single element:
 - Its hierarchic construction from higher codimensions.
 - Its transformation from a reference element.
 - Can have arbitrary shape.
- Position of elements relative to each other:
 - On one grid level.
 - With respect to different levels.
- A formal specification of grids is required to enable an accurate description of the grid interface.

General Idea



- Describe a single element:
 - Its hierarchic construction from higher codimensions.
 - Its transformation from a reference element.
 - Can have arbitrary shape.
- Position of elements relative to each other:
 - On one grid level.
 - With respect to different levels.
- A formal specification of grids is required to enable an accurate description of the grid interface.

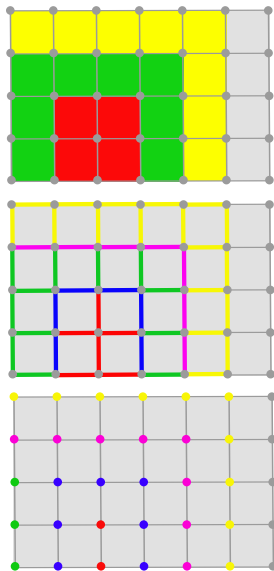


- Describe a single element:
 - Its hierarchic construction from higher codimensions.
 - Its transformation from a reference element.
 - Can have arbitrary shape.
- Position of elements relative to each other:
 - On one grid level.
 - With respect to different levels.
- A formal specification of grids is required to enable an accurate description of the grid interface.

- `Grid<d, w>` is a container of entities.
- Template parameters are dimension and world dimension.
- *View Model*: Read-only access to grid entities, consequent use of `const`.
- Nested refinement is only way to change a grid.
- Access to entities is only through iterators. Allows on-the-fly implementations.
- Traits classes: Grid exports the types of its constituents.
- Several instances of a grid with different dimension and implementation can coexist in a single program.
- Available implementations: `SGrid` (structured, n -dimensional), `YaspGrid` (structured, parallel, n -dimensional), `AlbertaGrid` (1D/2D/3D, unstructured, simplex, bisection), `UGGrid` (2D/3D, unstructured, parallel, multi-element), `Alu3DGrid` (3D, unstructured, tet/hex, parallel).
- In preparation: Networks (1D in n -D).

- $\text{Entity}\langle c, d \rangle$ is the entity of codimension c in d dimensions.
- It is a $d - c$ dimensional object.
- Consists of topological information and a Geometry object.
- Codimension 0 provides subentity and father relations as well as intersections.
- $\text{Geometry}\langle c, d, w \rangle$ is a transformation (Θ, f) from a $d - c$ dimensional reference element to \mathbb{R}^w .
- Geometry provides
 - local-to-global, global-to-local
 - Jacobian inverse of the map.
 - Determinant of Jacobian inverse.
 - Tangential vectors.

Parallel Data Decomposition



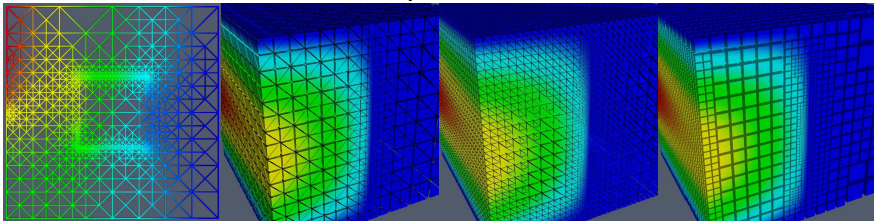
- Grid is mapped to $\mathcal{P} = \{0, \dots, P - 1\}$.
- $E = \bigcup_{p \in \mathcal{P}} E|_p$ possibly overlapping.
- $\pi_p : E|_p \rightarrow$ “partition type”.
- For codimension 0 there are three partition types:
 - *interior*: Nonoverlapping decomposition.
 - *overlap*: Arbitrary size.
 - *ghost*: 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.

- Access to entities is only through iterators.
- `LeafIterator<c,d>` iterates over codimension c leaf entities in a process. Begin is on the grid.
- `LevelIterator<c,d>` iterates over codimension c entities on a given level in a process. Begin is on the grid.
- Specializations for different partition types exist.
- `IntersectionIterator<d>`: iterate over intersections of a single codimension 0 entity. Begin is on the codimension 0 entity.
- `HierarchicalIterator<d>`: iterate over all childs of a codimension 0 entity. Begin is on the codimension 0 entity.

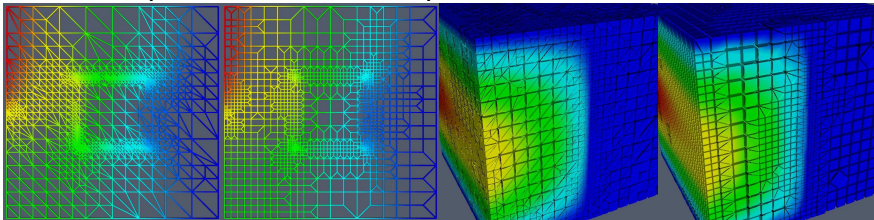
- In FE computations data is associated with subsets of entities.
- Subsets could be “vertices of level l ”, “faces of leaf elements”, ...
- Data should be stored in arrays for efficiency.
- Associate several numbers with each entity.
- *Leaf index*: zero-starting, consecutive, non-persistent numbering on “leaf” entities. Used to store solution and stiffness matrix.
- *Level index*: zero-starting, consecutive, non-persistent numbering of entities on one level. Used for geometric multigrid.
- *Globally unique id*: persistent id assigned to each entity. Used to transfer solution from one grid to another during mesh modification.
- Mapper classes use indices/ids to access data associated with a grid.

Dune Example

Alberta 2d, 3d, ALU3dGrid, simplices, cubes



UG 2d, simplices, cubes, 3d, simplices, cubes



(Viz.: ParaView/VTK)

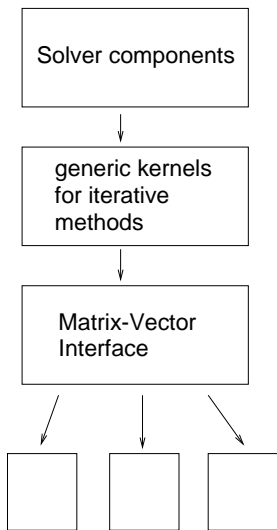
1 DUNE - Distributed Unified Numerics Environment

2 Abstractions for Finite Element Grids

3 Abstractions for Linear Algebra and Solvers

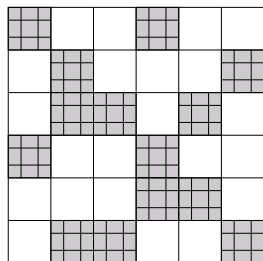
4 Scalability of a Multigrid Implementation

5 Conclusions

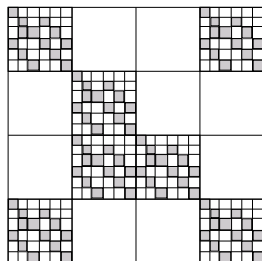


- There are already template libraries for linear algebra: MTL/ITL.
- Existing libraries exploit structure of FE-matrices at compile-time.
- Solver components: Based on operator concept, Krylov methods, (A)MG preconditioners.
- Generic kernels: Triangular solves, Gauß-Seidel step, ILU decomposition.
- Matrix-Vector Interface: Support recursively block structured matrices.
- Various implementations of the interface are available.

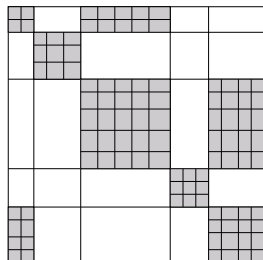
Block Structure in FE Matrices



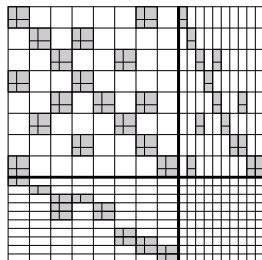
sparse block matrix
blocks are dense
blocks have fixed size
DG fixed p



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

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

Performance I

- Pentium 4 Mobile 2.4 GHz: Stream for $x = y + \alpha z$ is 1084 MB/s
- Compiler: GNU C++ compiler version 4.0
- Scalar product of two vectors (block size 1)

N	500	5000	50000	500000	5000000
MFLOPS	896	775	167	160	164

- daxpy operation $y = y + \alpha x$, 1200 MB/s transfer rate for large N

N	500	5000	50000	500000	5000000
MFLOPS	936	910	108	103	107

- Matrix-vector product, BCRSMatrix, 5-point stencil, b : block size

N, b	100,1	10000,1	1000000,1	1000000,2	1000000,3
MFLOPS	388	140	136	230	260

- Damped Gauß-Seidel scheme.
- 5-point stencil on 1000 by 1000 grid.
- Comparison of generic implementation in ISTL with specialized C implementation in AMGLIB.

	AMGLIB	ISTL
Time per iteration [s]	0.17	0.18

- Corresponds to about 150 MFLOPS (on PIV Mobile 2.4GHz).

- 1 DUNE - Distributed Unified Numerics Environment
- 2 Abstractions for Finite Element Grids
- 3 Abstractions for Linear Algebra and Solvers
- 4 Scalability of a Multigrid Implementation**
- 5 Conclusions

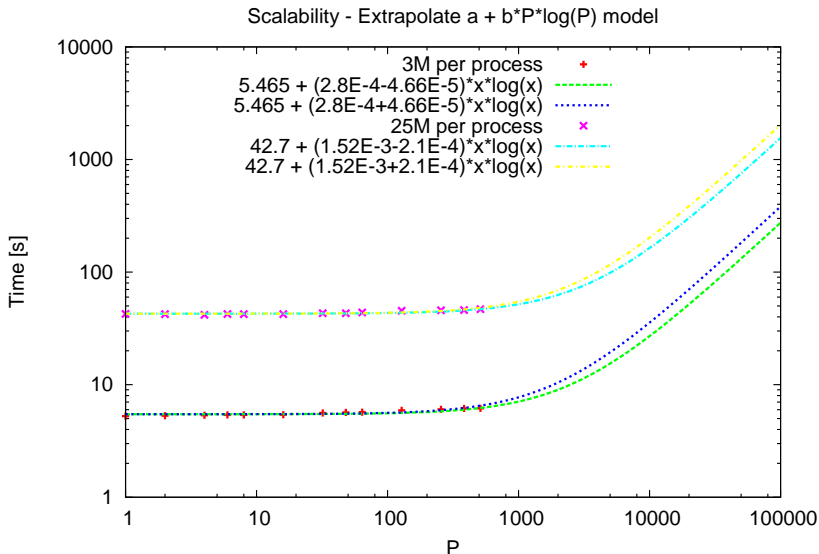
- Do we need new algorithms (for solving elliptic PDEs)?
- Solves $-\nabla \cdot \{K(x)\nabla u\} = f$ in $(0, 1)^d$ with Dirichlet, Neumann and periodic boundary conditions.
- Cell-centered finite volumes with harmonic averaging of permeabilities (“7-point stencil”).
- Cell-centered multigrid with constant restriction and linear interpolation.
- Structured grid with uniform refinement.
- Based on `Dune::YaspGrid` but does not use the Dune grid interface.
- On-the-fly implementation, 28 Bytes per cell, $25 \cdot 10^6$ cells in 1GByte, largest problem: $1.3 \cdot 10^{10}$ cells.
- Dimension independent implementation, results for $d = 3$.

- 256 nodes dual Athlon 1.3 GHz, installed in 2002.
- 2 GB RAM, 30 GB disk per node.
- Myrinet interconnect, full bisection bandwidth.
- Debian Woody.
- MPICH/gm.
- Intel Compiler 9.0.

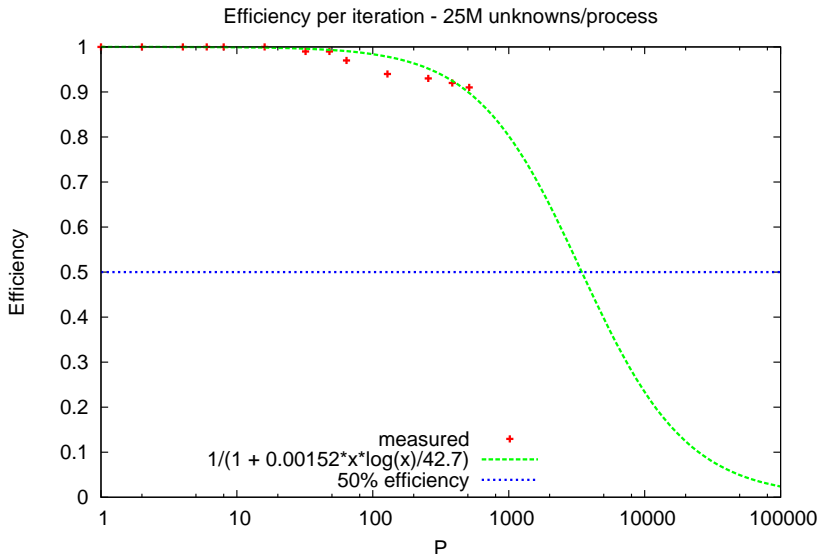
Parallel Implementation

- Levels m through J are partitioned to P processes.
- Optimally load balanced.
- Levels 0 through m are on one process.
- Requires $N_m = N_0(2^d)^m \geq P$.
- On level m , distributed residual is sent to all processes with all-to-all communication.
- All processors do a multigrid cycle down to level 0 with N_0 cells.
- $$T_P(N, P) = \underbrace{c_1 \frac{N}{P}}_{\text{comp.}} + \underbrace{c_2 \left(\frac{N}{P}\right)^\alpha}_{\text{local comm.}} + \underbrace{c_3 P \log P}_{\text{all-to-all}} + \underbrace{c_4 P}_{\text{coarse solve}} + \underbrace{c_5 \log P}_{\text{norm}}$$
- Scaled efficiency for $K = N/P$ large: $E_s(P) \approx \frac{1}{1 + CP \log P}$.
- Fit $T_P(N, P) = a + bP \log P$ to real data and extrapolate.

Fit and Extrapolate



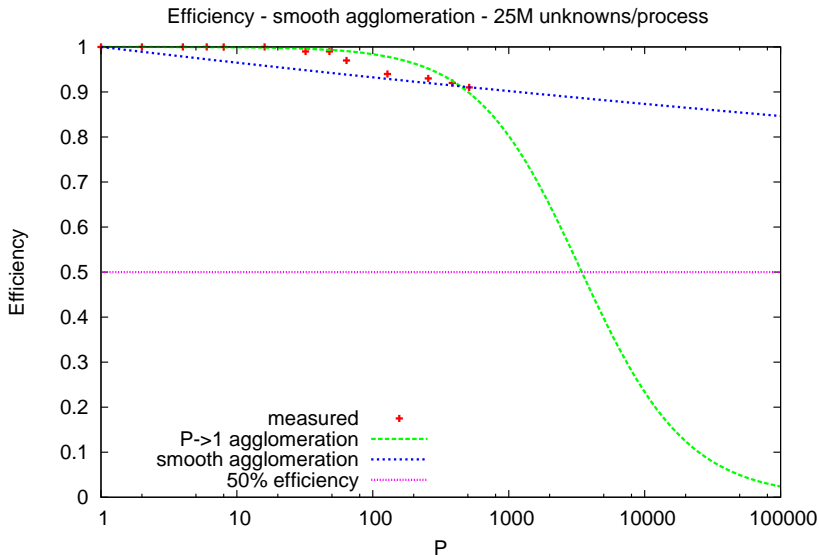
Parallel Efficiency



Smooth Agglomeration

- Agglomerate the coarse grid to fewer processes in several steps:
- Choose M .
- First M levels are on one process.
- Second M levels are on $(2^d)^M$ processes.
- Until level $L \cdot M$ is partitioned to $P = (2^d)^{LM}$ processes.
- Now $T_P(N, P) = c_1 \frac{N}{P} + c_2 \left(\frac{N}{P}\right)^\alpha + c_3 \log P$,
- and $E(N, P) = \frac{1}{1 + C_1 \left(\frac{P}{N}\right)^{1/d} + C_2 \left(\frac{P}{N}\right) \log P}$.
- Scaled efficiency for $K = N/P$ large enough: $E_s(P) \approx \frac{1}{1 + C \log P}$
- Now fit this model to the data on $P = 512$ to get an idea.

... And We Get



Application: Density Driven Flow

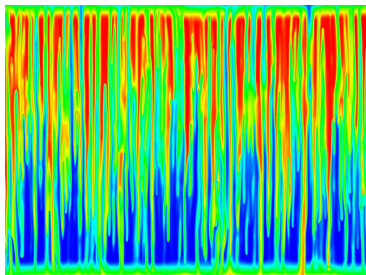
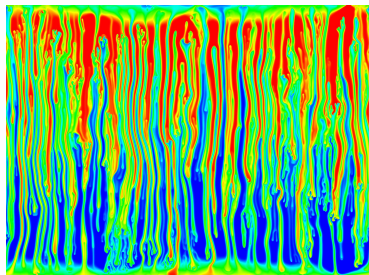
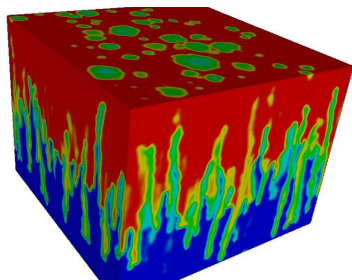
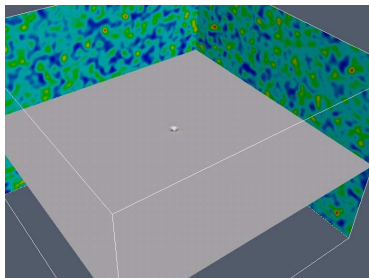
- This code can also do something useful ...
- Density driven flow:

$$\nabla \cdot u = f, \quad u = -\frac{K}{\mu}(\nabla p - \varrho(C)g), \quad \varrho(C) = C\varrho_b + (1 - C)\varrho_0 \quad \text{in } \Omega$$

$$\frac{\partial(\Phi C)}{\partial t} + \nabla \cdot j + q = 0, \quad j = Cu - D\nabla C$$

- Operator split.
- Transport equation solved with explicit cell-centered Finite-Volume scheme using Minmod slope limiter.

Simulation results



- 1 DUNE - Distributed Unified Numerics Environment
- 2 Abstractions for Finite Element Grids
- 3 Abstractions for Linear Algebra and Solvers
- 4 Scalability of a Multigrid Implementation
- 5 Conclusions**

- Dune Framework
 - High flexibility without significant performance penalty.
 - Provides unified access to different existing finite element packages.
 - Specialized grid implementations can be used through common IF.
- Scalability
 - Current multigrid code scales well up to 1000 processors.
 - Stepwise agglomeration should scale well up to 100000 processors.
- Challenges
 - Fast, transparent, reliable IO: Final example produces several TBytes of data on only 384 processors.
 - Parallel visualization: OK, there is ParaView/VTK but it is a pain.
 - Fault tolerance, MTBF in COTS clusters does not scale.
- Acknowledgement
 - Stefan Friedel - Helics system administrator.