DUNE —
Distributed and Unified Numerics Environment

Christian Engwer

Interdisciplinary Center for Scientific Computing, University of Heidelberg

February 23, 2006

Joint work with:

P. Bastian M. Blatt R. Klöfkorn S. Kuttanikkad T. Neubauer M. Ohlberger O. Sander

Christian.Engwer@iwr.uni-heidelberg.de http://dune.uni-hd.de/

The Problem with Finite Element Software

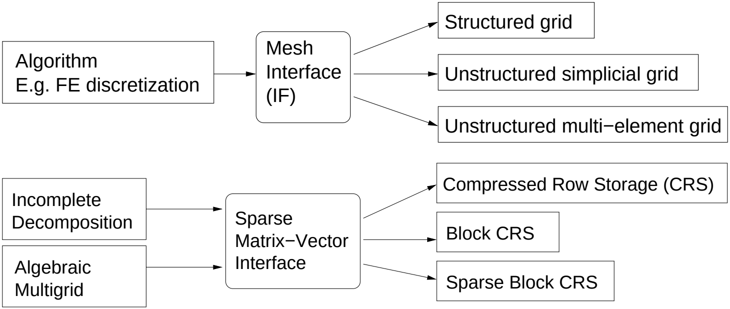
- There are many PDE software packages, each with a particular set of features:
    - IPARS: block structured, parallel, multiphysics.
    - Alberta: simplicial, unstructured, bisection refinement.
    - UG: unstructured, multi-element, red-green refinement, parallel.
    - QuocMesh: Fast, on-the-fly structured grids.
- 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

Outline

Concept I

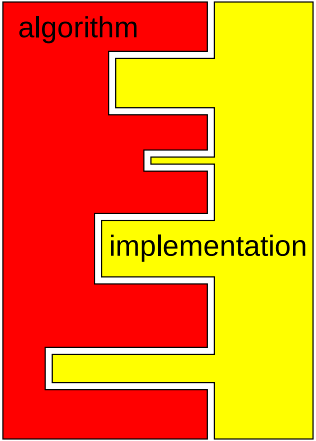
Seperate data structures and algorithms.

- 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



Concept II

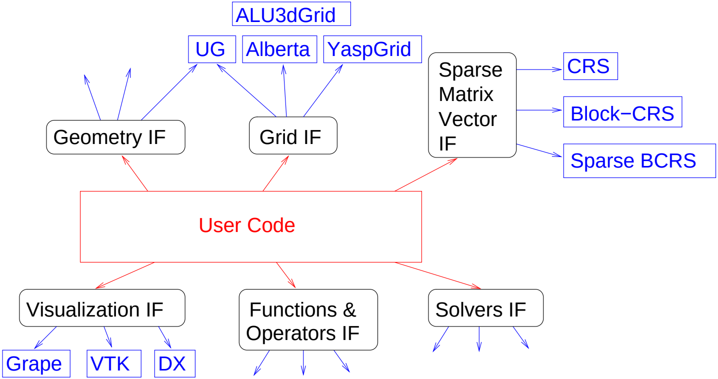
Implementation with generic programming techniques.



- 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 is known for some time:
  - Standard Template Library (1998): Containers. Blitz++, MTL/ITL, GTL, . . .
  - Thesis of Gundram Berti (2000): Concepts for grid based algorithms.

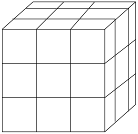
Concept III

Reuse existing finite element software.

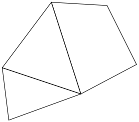


- Efficient integration of existing FE software.
- Developed by groups in Berlin, Freiburg and Heidelberg

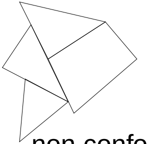
Finite Element Grids



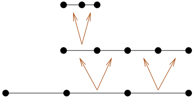
structured



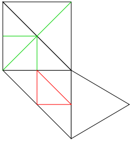
conforming



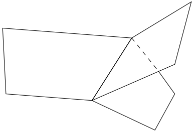
non conforming



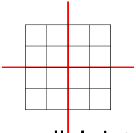
nested, 1D



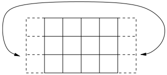
red-green, bisektion



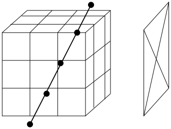
manifolds



parallel data decomposition

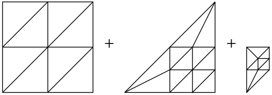


periodic

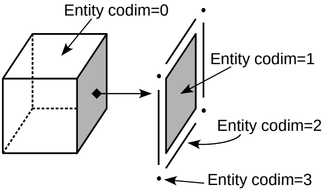


mixed dimensions

Grid

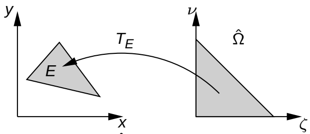


Hierarchic Grid

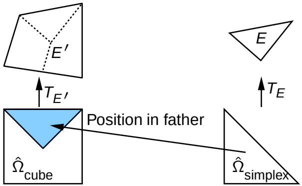


- A (hierarchic) grid has a dimension d, a world dimension w and maximum level J.
- A grid is a Container of entities (geometrical/topologocal objects) of different codimensions.
- *View Model*: Read-only access to grid entities, consequent use of **`const`**.
- Access to entities is only through iterators. Allows on-the-fly implementations.
- Several instances of a grid with different dimension and implementation can coexist in a single program.

Entities

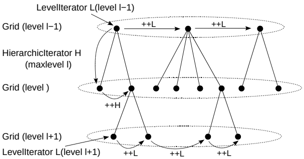


Mapping from $\hat{\Omega}$ into global coordinates.



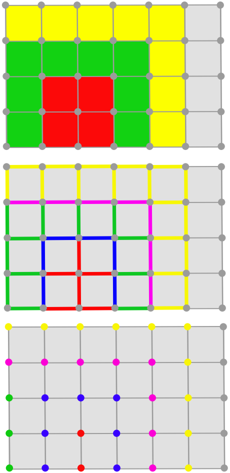
- Entity E is defined by...
  - Reference Element $\hat{\Omega}$
    - Describes all topological information.
    - Can be recursively constructed over dimension.
  - Transformation T_E
    - Maps from the reference element into global coordinates.
    - Provides Jacobian, its inverse and tangential vectors.
- Entity of Codimension 0 provides...
  - subentity and father relations.
  - intersections with neighbours and boundary.

Iterators



- `LeafIterator<d>` iterates over codimension 0 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.
- `IntersectionIterator<d>`: iterate over intersections of a single codimension 0 entity. Begin is on the codimension 0 entity.
- `HierarchicIterator<d>`: iterate over all childs of a codimension 0 entity. Begin is on the codimension 0 entity.

Parallel Data Decomposition



- Grid is mapped to $\mathcal{P} = \{0, \dots, P-1\}$.
- Each Entities is present on one or more processors.
- Each Entities is associated to **one** “partition type”.
- partition types:

interior	Nonoverlapping decomposition.	
overlap	Arbitrary size.	
ghost	Rest.	
border	Boundary of interior.	*(not for cd=0)*
front	Boundary of interior+overlap.	*(not for cd=0)*

- Allows implementation of overlapping and nonoverlapping Domain Decomposition methods.

Indices and Ids

- Allow association of FE computations data with subsets of entities.
- Subsets could be "vertices of level l", "faces of leaf elements", ...
- Data should be stored in arrays for efficiency.
- Associate index/id with each entity.

Leaf index
: zero-starting, consecutive, non-persistent, accessible on copies.
Used to store solution and stiffness matrix.

Level index
: zero-starting, consecutive, non-persistent.
Used for geometric multigrid.

Globally unique id
: persistent across grid modifications.
Used to transfer solution from one grid to another.

Available Implementations

- `SGrid` (structured, *n*-dimensional)
- `YaspGrid` (structured, parallel, *n*-dimensional)
- `AlbertaGrid` (1D/2D/3D, unstructured, simplex, bisection)
- `OneGrid` (adaptive, 1D)
- `UGGrid` (2D/3D, unstructured, parallel, multi-element)
- `ALU3DGrid` (3D, unstructured, tet/hex, parallel)
- In preparation: Networks (1D in *n*-D)

Performance of Grid Interface

- Consider Run-time for computing FE interpolation error for polynomial degree 1 and quadrature order 2.
- Same algorithm runs on `YaspGrid` and `UGGrid`

Grid	*d*	Type	Elements	Time [*s*]
`UGGrid`	2	simplex	131072	0.49
`UGGrid`	2	cube	65536	0.19
`YaspGrid`	2	cube	65536	0.09
`UGGrid`	3	cube	32768	0.19
`YaspGrid`	3	cube	32768	0.12

- `YaspGrid` is on-the-fly compared to `UGGrid`.
- Basis functions are not cached.

Performance Linear Algebra

Concepts I-III applied to Linear Algebra Interface

- Matrix-Vector performance
  - Pentium 4 Mobile 2.4 GHz, Compiler: GNU C++ 4.0
  - Stream benchmark for $x = y + \alpha z$ is 1084 MB/s
  - Scalar product of two vectors

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

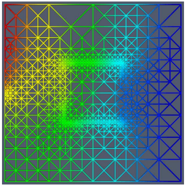
- Damped Gauß-Seidel solver
  - 5-point stencil on 1000×1000 grid
  - Comparison generic implementation in ISTL with specialized C implementation in AMGLIB

	AMGLIB	ISTL
Time per iteration [s]	0.17	0.18

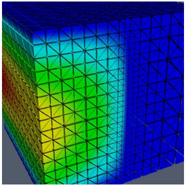
  - Corresponds to about 150 MFLOPS

Example: Generic Finite Element Discretization

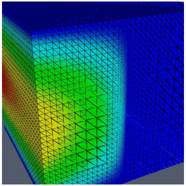
Generic P1 discretization of the Laplace equation. One code runs on all grids, with arbitrary element type and in arbitrary dimension.



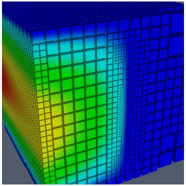
Alberta 2d



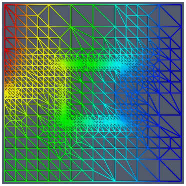
Alberta 3d



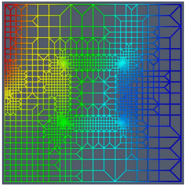
ALU3dGrid simplices



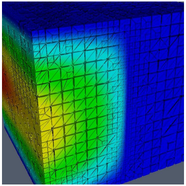
ALU3dGrid cubes



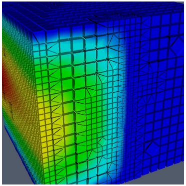
UGGrid 2d simplices



UGGrid 2d cubes



UGGrid 3d simplices



UGGrid 3d cubes

Conclusions

- DUNE is based on the following principles:
  - Seperation of data structures and algorithms.
  - Implementation through generic programming techniques.
  - Reuse of existing codes.
  - Free software.
- This approach allows for flexibility while not imposing any performance penalty.
- Current plans:
  - Finish grid interface, index/ids, reference elements.
  - Finish version 1.0 including documentation and tutorial.

`Distributed and Unified Numerics Environment`

`http://dune.uni-hd.de/`