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
1 The Concept

2 Abstract Grid Interface
   Grid
   Entities
   Iterators
   Parallel Data Decomposition
   Indices and Ids
   Available Implementations

3 Performance Evaluation

4 Conclusions
## 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mesh Interface (IF)</th>
<th>Sparse Matrix–Vector Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.g. FE discretization</td>
<td>Structured grid</td>
<td>Compressed Row Storage (CRS)</td>
</tr>
<tr>
<td>Incomplete Decomposition</td>
<td>Unstructured simplicial grid</td>
<td>Block CRS</td>
</tr>
<tr>
<td>Algebraic Multigrid</td>
<td>Unstructured multi–element grid</td>
<td>Sparse Block CRS</td>
</tr>
</tbody>
</table>

- **Examples:**
  - **Finite Element (FE) discretization**
  - **Mesh Interface (IF)**
    - Structured grid
    - Unstructured simplicial grid
    - Unstructured multi–element grid
  - **Sparse Matrix–Vector Interface**
    - Compressed Row Storage (CRS)
    - Block CRS
    - Sparse Block CRS
The Concept

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:
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
- Periodic
- Parallel data decomposition
- Mixed dimensions
A (hierarchic) grid has a dimension $d$, a world dimension $w$ and maximum level $J$.

A grid is a Container of entities (geometrical/topological 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.
Abstract Grid Interface

Entities

- **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 $\mathcal{P} = \{0, \ldots, P - 1\}$.
- Each Entity is present on one or more processors.
- Each Entity 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

<table>
<thead>
<tr>
<th>Grid</th>
<th>$d$</th>
<th>Type</th>
<th>Elements</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>UGGrid</td>
<td>2</td>
<td>simplex</td>
<td>131072</td>
<td>0.49</td>
</tr>
<tr>
<td>UGGrid</td>
<td>2</td>
<td>cube</td>
<td>65536</td>
<td>0.19</td>
</tr>
<tr>
<td>YaspGrid</td>
<td>2</td>
<td>cube</td>
<td>65536</td>
<td>0.09</td>
</tr>
<tr>
<td>UGGrid</td>
<td>3</td>
<td>cube</td>
<td>32768</td>
<td>0.19</td>
</tr>
<tr>
<td>YaspGrid</td>
<td>3</td>
<td>cube</td>
<td>32768</td>
<td>0.12</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>$N$</th>
<th>500</th>
<th>5000</th>
<th>50000</th>
<th>500000</th>
<th>5000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLOPS</td>
<td>896</td>
<td>775</td>
<td>167</td>
<td>160</td>
<td>164</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$N$</th>
<th>500</th>
<th>5000</th>
<th>50000</th>
<th>500000</th>
<th>5000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLOPS</td>
<td>936</td>
<td>910</td>
<td>108</td>
<td>103</td>
<td>107</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>AMGLIB</th>
<th>ISTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per iteration [s]</td>
<td>0.17</td>
<td>0.18</td>
</tr>
</tbody>
</table>

- 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.
Conclusions

• DUNE is based on the following principles:
  • Separation 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/