DUNE — The Distributed Unified Numerics Environment

P. Bastian

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Universität Heidelberg

Berlin, 1.7.2005

Joint work with:
M. Blatt       C. Engwer       R. Klöfkorn      S. Kuttanikkad
T. Neubauer    M. Ohlberger    O. Sander

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

1. The Concept
2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids
3. Interface Implementation
   - Classes
   - Example
4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance
5. Conclusions
1. The Concept

2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids

3. Interface Implementation
   - Classes
   - Example

4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance

5. Conclusions
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 to 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
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

Diagram:
- 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
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

Diagram:
- 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
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, ...
Reuse existing finite element software.

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

1. The Concept
2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids
3. Interface Implementation
   - Classes
   - Example
4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance
5. Conclusions
Finite Element Grids

There is great variability in finite element grids:

- **Structured grid**: $O(1)$ memory, transformation might be simple.
- **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.

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.
- 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.
- 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.
Convex polytope $H \subset \mathbb{R}^w$

- Is the convex hull of a finite set of points $X = \{x_0, \ldots, x_n\}$. $H$ is a closed set and $H = \overset{o}{H} \cup \partial H$.
- If $n = 0$, then $H$ is a single point $\{x_0\}$.
- If $n > 0$, then let $\{b_1, \ldots, b_d\}$ be a basis of $\{x_1 - x_0, \ldots, x_n - x_0\}$. $\dim(H) = d \leq \min(n, w)$ is the dimension of $H$.

Face of a convex polytope

Let $H$ be the polytope generated by the point set $X$. $F$ is a face of $H$ iff

(i) $F \subset \partial H$, and

(ii) $F$ is generated by $Y \subset X$.

A face $F$ has dimension $0 \leq \dim(F) \leq \dim(H)$. 
### Convex polytope $H \subset \mathbb{R}^w$

- Is the convex hull of a finite set of points $X = \{x_0, \ldots, x_n\}$. $H$ is a closed set and $H = \text{int}(H) \cup \partial H$.
- If $n = 0$, then $H$ is a single point $\{x_0\}$.
- If $n > 0$, then let $\{b_1, \ldots, b_d\}$ be a basis of $\{x_1 - x_0, \ldots, x_n - x_0\}$. $\dim(H) = d \leq \min(n, w)$ is the dimension of $H$.

### Face of a convex polytope

Let $H$ be the polytope generated by the point set $X$. $F$ is a face of $H$ iff

1. $F \subset \text{int}(H)$, and
2. $F$ is generated by $Y \subset X$.

A face $F$ has dimension $0 \leq \dim(F) \leq \dim(H)$. 

---

Bastian (IWR Heidelberg)
Codimension of a face

A face $F$ of a polytope $H$ has codimension $c$ iff $\dim(F) = \dim(H) - c$. $H$ itself has codimension 0. Some common names: Facet ($c = 1$), ridge ($c = 2$), edge ($c = \dim(H) - 1$), vertex ($c = \dim(H)$).

Transformation

Let $0 \leq d \leq w$ be integers. $(D, f)$ is a transformation iff

(i) $D \subset \mathbb{R}^d$ is a closed, bounded point set, and

(ii) $f \in (C^1(D))^w$ is one-to-one.

Generalized polytope

$E \subset \mathbb{R}^w$ is a generalized polytope if there is a convex polytope $H$ and a transformation $(H, f)$ such that $\text{Range}(f) = E$. $F$ is a face of $E$ if $G$ is a face of $H$ such that $\text{Range}(f|_G) = F$. 
Preliminaries II

Codimension of a face

A face $F$ of a polytope $H$ has codimension $c$ iff $\dim(F) = \dim(H) - c$. $H$ itself has codimension 0. Some common names: Facet ($c = 1$), ridge ($c = 2$), edge ($c = \dim(H) - 1$), vertex ($c = \dim(H)$).

Transformation

Let $0 \leq d \leq w$ be integers. $(D, f)$ is a transformation iff

(i) $D \subseteq \mathbb{R}^d$ is a closed, bounded point set, and

(ii) $f \in \left( C^1(D) \right)^w$ is one-to-one.

Generalized polytope

$E \subseteq \mathbb{R}^w$ is a generalized polytope if there is a convex polytope $H$ and a transformation $(H, f)$ such that $\text{Range}(f) = E$. $F$ is a face of $E$ if $G$ is a face of $H$ such that $\text{Range}(f|_G) = F$. 
Codimension of a face

A face $F$ of a polytope $H$ has codimension $c$ iff $\dim(F) = \dim(H) - c$. $H$ itself has codimension 0. Some common names: Facet ($c = 1$), ridge ($c = 2$), edge ($c = \dim(H) - 1$), vertex ($c = \dim(H)$).

Transformation

Let $0 \leq d \leq w$ be integers. $(D, f)$ is a transformation iff

(i) $D \subset \mathbb{R}^d$ is a closed, bounded point set, and
(ii) $f \in (C^1(D))^w$ is one-to-one.

Generalized polytope

$E \subset \mathbb{R}^w$ is a generalized polytope if there is a convex polytope $H$ and a transformation $(H, f)$ such that $\text{Range}(f) = E$. $F$ is a face of $E$ if $G$ is a face of $H$ such that $\text{Range}(f|_G) = F$. 
Reference Elements

- Reference elements are standard convex polytopes.
- $\Theta_{d,t}$ is the $d$-dimensional reference element of type $t$.
- The polytope and all its faces are entities: $R_{d,t} = \{t_0, f_0, \ldots, v_3\}$.
- $\tau : R_{d,t} \rightarrow \text{“types”}$, $c : R_{d,t} \rightarrow \{0, \ldots, d\}$.
- $\mathcal{H}_{d,t} \subset R_{d,t} \times R_{d,t}$: $(r, r') \in \mathcal{H}_{d,t}$ iff $r'$ subentity (part of) of $r$.
- Local numbering of subentities w.r.t. containing entity.
- Recursive construction over dimension via isomorphic edge-weighted graphs.
- Positions: $\pi : R_{d,t} \rightarrow \mathbb{R}^d$. 

Part of $\Theta_{\text{simplex},3}$
A grid consists of generalized polytopes.

A (hierarchic) grid has a dimension $d$, a world dimension $w$ and maximum level $J$.

Entity set: $E = \bigcup_{j \in J} \bigcup_{c \in C} E_j^c$, where $J = \{0, \ldots, J\}$, $C = \{0, \ldots, d\}$.

Every $e \in E$ is a generalized polytope with associated polytope $\Theta_{d-c(e),\tau(e)}$.

$S \subset E \times E$: $(e, e') \in S$ iff $e'$ subentity of $e$. Then $c(e') > c(e)$ and $j(e') = j(e)$.

Exact subentity relation can be deduced from reference element.

For $e \in E$, $(\Omega(\Theta_{d-c(e),\tau(e)}), f_e)$ maps reference element to $e$. 
Grid refinement is always logically nested.

\[ \mathcal{F} \subseteq E \times E: (e, e') \in \mathcal{F} \iff e \text{ is obtained from refinement of } e'. \]

\[ \mathcal{F} \text{ includes all codims}. \]

\[ e \in E, c(e) = 0, (e, e') \in \mathcal{F}: g_{e,e'} : \Omega(\Theta_{d,\tau(e)}) \rightarrow \Omega(\Theta_{d,\tau(e')}) \]

Allows evaluation of coarse grid function on the fine mesh.

Does not imply \( \Omega(e) \subseteq \Omega(e') \).

Leaf entities: \( L = \{ e' \in E \mid \nexists e \in E : (e, e') \in \mathcal{F} \} \).

Copy relation: \( \mathcal{Y} \subseteq E \times E: (e, e') \in \mathcal{Y} \iff e \text{ is a copy of } e' \).

\( \mathcal{Y} \) is transitive.

Copies may only be copied.
Intersections

- Intersection $\lambda = (e, e', \epsilon, \epsilon', \theta, m_g, m_l, m_l')$:
  - $e, e' \in E^0$, $\epsilon, \epsilon' \in E^1$,
  - $\theta$ : reference element,
  - $m_g : \Omega(\theta) \rightarrow \mathbb{R}^w$,
  - $m_l : \Omega(\theta) \rightarrow \Omega(\Theta_{d,\tau}(e))$,
  - $m_l' : \Omega(\theta) \rightarrow \Omega(\Theta_{d,\tau}(e'))$.

- For $e_3 : (e_3, e_1, \ldots), (e_3, e_4, \ldots)$, for $e_2$:
  - $(e_2, e_1, \ldots), (e_2, e_7, \ldots)$.

- Handles nonconforming meshes and nonconforming refinement.

- 3D : There might be several intersections per face.

- Internal and external boundaries handled similarly.
Grid is mapped to $\mathcal{P} = \{0, \ldots, P - 1\}$.

$E = \bigcup_{\rho \in \mathcal{P}} E|_{\rho}$ possibly overlapping.

$\pi_\rho : E|_{\rho} \to \text{“partition type”}$.

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.
In FE computations data is associated with subsets of entities $E' \subseteq E$.

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**: $\text{leaf}_p^c : E|_p \cap L \cap E^c \rightarrow \{0, \ldots, N_p^c - 1\}$, zero-starting, consecutive, non-persistent, accessible on copies. Used to store solution and stiffness matrix.

**Level index**: $\text{level}_{j,p}^c : E|_p \cap E_j^c \rightarrow \{0, \ldots, M_{j,p}^c - 1\}$, zero-starting, consecutive, non-persistent. Used for geometric multigrid.

**Globally unique id**: $\text{id} : E \rightarrow \mathbb{N}_0$, persistent across grid modifications. Used to transfer solution from one grid to another.

Mappers use indices/ids to access data associated with a grid.
Outline

1. The Concept
2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids
3. Interface Implementation
   - Classes
   - Example
4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance
5. Conclusions
Grid

- `Grid<d, w>` is a container of entities.
- Template parameters are dimension and world dimension (if supported by underlying implementation).
- **View Model**: Read-only access to grid entities, consequent use of `const`.
- 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).
**Entity/Geometry**

- Entity\(<c,d>\) is the entity of codimension \(c\) in \(d\) dimensions.
- Contains topological information about entity, geometry is in separate class.
- Specializations for codimension 0 and \(d\).
- Codimension 0 provides subentity and father relations as well as intersections.
- Geometry\(<c,d,w>\) is a transformation \((\Theta, f)\) from a reference element to the entity.
- It provides Jacobian, its inverse and tangential vectors.
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.
- Specializations for different partition types exist.
Example: \( L_2 \) interpolation error for conforming FE

template<class G, class Functor>

double L2Error (G& grid, Functor f, int k, int p) { // polynomial order k, quadrature order p
    const int dim = G::dimension;
    const int dimworld = G::dimensionworld;
    typedef typename G::ctype ct;
    typedef typename G::Traits::LeafIterator LeafIterator;

double sum = 0.0;
LeafIterator eendit = grid.leafend(grid.maxlevel());
for (LeafIterator it = grid.leafbegin(grid.maxlevel()); it!=eendit; ++it) {
    Dune::GeometryType gt = it->geometry().type();
    double coefficients[Dune::LagrangeShapeFunctionSetContainer<ct,double,dim>::maxsize];
    for (int j=0; j<Dune::LagrangeShapeFunctions<ct,double,dim>::general(gt,k).size(); j++)
        coefficients[j] = f(it->geometry().global(Dune::LagrangeShapeFunctions<ct,double,dim>::general(gt,k)[j].position()));
    for (int i=0; i<Dune::QuadratureRules<ct,dim>::rule(gt,p).size(); ++i) {
        const Dune::FieldVector<ct,dim>& ippos = Dune::QuadratureRules<ct,dim>::rule(gt,p)[i].position();
        double exact = f(it->geometry().global(ippos));
        double approx = 0;
        for (int j=0; j<Dune::LagrangeShapeFunctions<ct,double,dim>::general(gt,k).size(); j++)
            approx += coefficients[j]*Dune::LagrangeShapeFunctions<ct,double,dim>::general(gt,k)[j].evaluateFunction(0,ippos);
        double weight = Dune::QuadratureRules<ct,dim>::rule(gt,p)[i].weight();
        double refvolume = Dune::ReferenceElements<ct,dim>::general(gt).volume();
        double detjac = it->geometry().integrationElement(ippos);
        sum += (exact-approx)*(exact-approx)*weight*refvolume*detjac;
    }
    return sqrt(sum);
}
Performance Evaluation

- 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>

- First results thanks to S. Kuttanikkad and O. Sander!
- **YaspGrid** is on-the-fly compared to **UGGrid**.
- Basis functions are not cached.
1. The Concept

2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids

3. Interface Implementation
   - Classes
   - Example

4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance

5. Conclusions
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
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:

  ```cpp
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:

  ```cpp
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-Matrix Interface

**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
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)

<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>

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

<table>
<thead>
<tr>
<th>$N, b$</th>
<th>100,1</th>
<th>10000,1</th>
<th>1000000,1</th>
<th>1000000,2</th>
<th>1000000,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLOPS</td>
<td>388</td>
<td>140</td>
<td>136</td>
<td>230</td>
<td>260</td>
</tr>
</tbody>
</table>
Example: Generic Gauß-Seidel

```cpp
template<class M, class X, class Y, class K>
static void dbgs (const M& A, X& x, const Y& b, const K& w) {
    typedef typename M::ConstRowIterator rowiterator;
    typedef typename M::ConstColIterator coliterator;
    typedef typename Y::block_type bblock;
    typedef typename X::block_type xblock;
    bblock rhs; X xold(x); rowiterator endi=A.end();
    for (rowiterator i=A.begin(); i!=endi; ++i) {  // loop over rows
        rhs = b[i.index()];  // initialize rhs
        coliterator endj=(*i).end();  // end of row i
        coliterator j=(*i).begin();  // start of row i
        for (; j.index()<i.index(); ++j)  // lower triangle
            (*j).mmv(x[j.index()],rhs);
        coliterator diag=j;
        for (; j!=endj; ++j)  // upper triangle
            (*j).mmv(x[j.index()],rhs);
        algmeta_itsteps<I-1>::dbgs(*diag,x[i.index()],rhs,w); // 'solve'
    }
    x *= w; x.axpy(1-w,xold);  // update with damping
}
```
Damped Gauß-Seidel solver

5-point stencil on 1000 by 1000 grid

Comparison of generic implementation in ISTL with specialized C implementation in AMGLIB

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

Corresponds to about 150 MFLOPS
1. The Concept

2. Abstract Description of Grids
   - Preliminaries
   - Reference Elements
   - Grids

3. Interface Implementation
   - Classes
   - Example

4. Application to Linear Algebra and Solvers
   - Expressing Structure in FE Matrices
   - Performance

5. Conclusions
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.