The Distributed and Unified Numerics Environment (DUNE)

Peter Bastian

Institut für Parallele und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart
email: Peter.Bastian@ipvs.uni-stuttgart.de

8. September 2008
Contents

DUNE Concept

DUNE Grid Interface

Iterative Solver Template Library

Some Numerical Examples
About DUNE

Software framework for mesh based computations, mainly numerical solution of partial differential equations.

DUNE developer groups:

- **Berlin**: Ralf Kornhuber, Oliver Sander, ...
- **Freiburg**: Dietmar Kröner, Andreas Dedner, Robert Klöfkorn, ...
- **Münster**: Mario Ohlberger, ...
- **Stuttgart**: Markus Blatt, Christian Engwer, Peter Bastian, ...

DUNE 1.0 released December 20, 2007

Available from [www.dune-project.org](http://www.dune-project.org)
Inhalt

DUNE Concept

DUNE Grid Interface

Iterative Solver Template Library

Some Numerical Examples
Requirements for PDE Software

- **Flexibility in the mesh**
  - Structured meshes for image processing (filtering, segmentation, reconstruction).
  - Arbitrary dimensional (unstructured) grids, grids on manifolds.
  - Nonconforming grids, adaptive grids, moving grids, periodic grids, ... 

- **Flexibility in the methods**
  - Arbitrary placement and number of degrees of freedom.
  - Dimension-independent formulation.
  - Flexible (separate) linear algebra.

- **Multiphysics/Multiscale capability**
  - Coupling of models on same/different scales.

- **Efficiency**
  - Single processor (core) efficiency (locality, pipelining).
  - Parallelism, from multicore to supercomputers.

All in one software framework?
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 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
Concept 1

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

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
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 has been around for some time:
  - Standard Template Library (1998): Containers. Blitz++, MTL/ITL, GTL, ...
Modularity and reuse of existing finite element software.

- Efficient integration of existing FE software.
- Concentrate on dune-grid and dune-istl.
Inhalt

DUNE Concept

DUNE Grid Interface

Iterative Solver Template Library

Some Numerical Examples
Finite Element Grids

structured, 3D

conforming, 2D

nonconforming

nested, 1D

red-green, bisection

topological spaces

periodic

mixed dimensions

data decomposition
The DUNE Grid Interface

The DUNE grid interface supports meshes with the following properties:

- Elements of various shapes and dimensions (in principle arbitrary).
- Grids embedded in higher-dimensional spaces (e.g., grids on manifolds).
- Logically nested local grid refinement.
- Nonconformity in and between levels of refinement.
- Overlapping and nonoverlapping decompositions for parallel processing.
- Separation of grid and data associated with grid entities.

A single implementation does not support all these features.
How to Describe such a General FE Grid?

- Describe a single element:
  - Its hierarchic construction from higher codimensions (topology).
  - Its transformation from a reference element (geometry).

- Position of elements relative to each other:
  - On one grid level (intersection).
  - With respect to different levels (hierarchic relation).

- A formal specification of grids has been given to enable an accurate description of the grid interface\(^1\).

---

Aim: dimension-independent formulation.

A grid is a container of entities.

Grid dimension is $d$.

**Codimension**: Entity of codimension $c$ is a $d - c$ dimensional object.

Each entity has a $d - c$-dimensional reference element.

Reference Element is mapped to $\mathbb{R}^w$.

Reference elements are convex polytopes.
Index Maps

- All user data is stored external to a grid.
- In FE computations data is associated with subsets of entities $E' \subseteq E$.
- An injective map $i : E' \to \mathbb{N}$ is called an index map.
- A consecutive index map has $\text{Range}(i) = \{0, \ldots, |E'|-1\}$.
- Consecutive index maps are used to store user data in arrays.
- A persistent index map ensures that an entity is mapped to the same index as long as it exists.
- An index map can in general not be persistent and consecutive.
- Persistent index maps are used to transfer user data between grids after refinement and load balancing.
Additional Concepts

- Intersections
  - Two codim 0 entities $e, e'$ intersect if $\partial e \cap \partial e'$ is $d - 1$ dimensional.
  - Intersections need not be common faces.

- Refinement
  - Refinement is hierarchical, resulting in a forest structure.
  - Refinement rules are arbitrary.
  - Grid is structured into levels and leaf.

- Grid as container.
  - Entities and intersections are traversed via iterators (as in the STL).
  - A grid can be modified only through refinement and load balancing.
Parallel Data Decomposition

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

template<
class M, class V> // mapper type and vector type
class VectorExchange
  : public Dune::CommDataHandleIF<VectorExchange<M,V>, typename V::value_type> {
public:
  typedef typename V::value_type DataType; // Type in MPI buffer

  bool contains (int dim, int codim) const {
    return (codim==0); // only element data
  }

  bool fixedsize (int dim, int codim) const {
    return true; // same size for all entities
  }

  template<class EntityType>
  size_t size (EntityType& e) const {
    return 1; // one object of type DataType per entity
  }

  template<class MessageBuffer, class EntityType>
  void gather (MessageBuffer& buff, const EntityType& e) const {
    buff.write(v[mapper.map(e)]); // sender action
  }

  template<class MessageBuffer, class EntityType>
  void scatter (MessageBuffer& buff, const EntityType& e, size_t n) {
    DataType x; buff.read(x); v[mapper.map(e)]=x; // receiver action
  }

  ... 
};

VectorExchange<M,V> dh(mapper,update);
grid.communicate<VectorExchange<M,V>>(dh,Dune::InteriorBorder_All_Interface,
Dune::ForwardCommunication);
Available Implementations

- **SGrid**: structured, \( n \)-dimensional, on-the-fly.
- **YaspGrid**: structured, parallel, \( n \)-dimensional, on-the-fly.
- **AlbertaGrid**, the FE package Alberta in DUNE: 1D/2D/3D, unstructured, simplex, bisection.
- **UGGrid**, the FE package UG in DUNE: 2D/3D, unstructured, (not yet) parallel, multi-element, red-green & hanging nodes.
- **ALU3DGrid**: 3D, unstructured, tet/hex, hanging node refinement, parallel.
- **NeuronGrid**: 1D in \( n \)-D.
Inhalt

DUNE Concept

DUNE Grid Interface

Iterative Solver Template Library

Some Numerical Examples
Iterative Solver Template Library

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

---

Block Structure in FE Matrices

- Sparse block matrix
- Blocks are dense
- Blocks have fixed size
- DG fixed p

- 2x2 block matrix
- Each block is sparse
- Taylor-Hood elements

- Blocks are sparse
- Diffusion-reaction systems

- Blocks are dense
- Blocks have variable size
- DG hp version
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;
```
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 $y = y + \alpha x$, 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, 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>
Performance II

- Generic (!) damped Gauß-Seidel solver written using the abstract interface.
- 5-point stencil on 1000 by 1000 grid
- Comparison of generic implementation in ISTL with specialized C implementation in AMGLIB (old C code)

<table>
<thead>
<tr>
<th></th>
<th>ISTL</th>
<th>AMGLIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time per iteration [s] SOR</td>
<td>0.124</td>
<td>0.11</td>
</tr>
<tr>
<td>Time per iteration [s] SSOR</td>
<td>0.158</td>
<td>0.22</td>
</tr>
</tbody>
</table>
ISTL Parallelization Concept

- Parallelization is build **on top** of sequential vector, matrix and solver components.
- The consistency model is **not** prescribed. Algorithms are parametrized to work with different consistency models.
- Parallelization using **distributed index sets**:
  - $I \subseteq \mathbb{N}_0$ denotes an arbitrary index set.
  - $(I_p)_{p \in \mathcal{P}}, \bigcup_{p \in \mathcal{P}} I_p = I$, data distribution.
  - $\alpha_p : I_p \rightarrow A$, attribute map.
  - Communication schedule: $A_s \subseteq A, A_r \subseteq A$:
    For $i \in I_p$ with $\alpha_p(i) \in A_s$ send data to every $i \in I_q$, $q \neq p$ with $\alpha_q(i) \in A_r$.
  - Locally data is stored in using vectors via global to local map: $\gamma_p : I_p \rightarrow \hat{I}_p = \{0, \ldots, |I_p| - 1\}$.
  - User interface similar to grid communication.

---

Solvers Implemented in ISTL

- Krylov methods (parallel and operator based)
  - Gradient method
  - CG
  - BICGStab

- Preconditioners
  - Jacobi, SOR, SSOR
  - ILU(n)
  - Parallel agglomeration based algebraic multigrid
  - Overlapping Schwarz

- Direct solvers
  - Interface to SuperLU
Inhalt

DUNE Concept

DUNE Grid Interface

Iterative Solver Template Library

Some Numerical Examples
Dune Example

Solve

\[-\Delta u = f \quad \text{in } \Omega = (-1/2, 1/2) \times (0, 1) \times (0, 1),\]
\[-\nabla u \cdot n = 0 \quad \text{on } \Gamma_N = \{(x, 0, z) \mid -1/2 < x < 0, 0 < z < 1\},\]
\[u = g \quad \text{on } \partial \Omega \setminus \Gamma_N,\]

with exact solution

\[u(r, \varphi, z) = r^{1/2} \sin \left(\frac{\varphi}{2}\right) 4z(1 - z)\]

using P1 conforming finite elements and residual-based a-posteriori error estimator

\[\eta_e = h_e^2 \|f\|^2_{\omega_e} + \frac{1}{2} \sum_{\lambda=(e,e',...), \omega_\lambda \subseteq \partial \Omega} h_\lambda \|\nabla u \cdot n\|^2_{\omega_\lambda} + \sum_{\lambda=(e,e,...), \omega_\lambda \subseteq \Gamma_N} h_\lambda \|\nabla u \cdot n\|^2_{\omega_\lambda}.\]
Dune Example

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

(Viz.: ParaView/VTK)
Comparison of adaptive grids in 3D

- Simplices, bisection (Alberta)
- Simplices, red, conforming (UG)
- Cubes, red, conforming (UG)
- Simplices, red, hanging (ALU)
- Cubes, red, hanging (ALU)
- Cubes, uniform (Yasp)

L2 error vs. CPU time graph.
Details of the example above.

<table>
<thead>
<tr>
<th>Grid</th>
<th>$N$</th>
<th>$T[s]$</th>
<th>$\frac{T}{N}[\mu s]$</th>
<th>$\text{MAT}$</th>
<th>$\text{ASS}$</th>
<th>$\text{SLV}$</th>
<th>$\text{EST}$</th>
<th>$\text{ADP}$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>s, Alberta</td>
<td>496304</td>
<td>117.8</td>
<td>237</td>
<td>11</td>
<td>14</td>
<td>4.8</td>
<td>39</td>
<td>32</td>
<td>$7.7 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>s, UG</td>
<td>493030</td>
<td>175.3</td>
<td>356</td>
<td>11</td>
<td>17</td>
<td>6.1</td>
<td>29</td>
<td>37</td>
<td>$8.3 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>s, ALUGrid</td>
<td>537515</td>
<td>134.8</td>
<td>251</td>
<td>24</td>
<td>24</td>
<td>6.2</td>
<td>28</td>
<td>18</td>
<td>$12.7 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>c, UG</td>
<td>365891</td>
<td>59.6</td>
<td>163</td>
<td>14</td>
<td>25</td>
<td>8.4</td>
<td>26</td>
<td>26</td>
<td>$13.2 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>c, ALUGrid</td>
<td>360118</td>
<td>42.2</td>
<td>117</td>
<td>26</td>
<td>30</td>
<td>10</td>
<td>22</td>
<td>12</td>
<td>$14.7 \cdot 10^{-5}$</td>
</tr>
<tr>
<td>c, YaspGrid</td>
<td>274625</td>
<td>19.7</td>
<td>72</td>
<td>22</td>
<td>34</td>
<td>14</td>
<td>25</td>
<td>5.1</td>
<td>$59.0 \cdot 10^{-5}$</td>
</tr>
</tbody>
</table>

MAT: construction of the sparsity pattern.

ASS: the matrix assembly.

SLV: the linear solver (CG with SSOR).

EST: the error estimator.

ADP: the adaptation (consisting of grid refinement and vector reorganization but excluding error estimation).
Forward Facing Step (R. Klöfkorn, Freiburg)

- Compressible Euler equation of gas dynamics.
- Time-explicit, locally adaptive finite volume scheme.
- Parallel solution using ALUGrid and dynamic load balancing.

<table>
<thead>
<tr>
<th>Density</th>
<th>Density</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>t=1</td>
<td>t=2</td>
<td>t=3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partitioning and Grid</th>
<th>Partitioning and Grid</th>
<th>Partitioning and Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>t=1</td>
<td>t=2</td>
<td>t=3</td>
</tr>
</tbody>
</table>

Forward Facing Step

Numerics: R. Klöfkorn
Comparison\textsuperscript{4} between original version and DUNE version based on ALUGrid.


<table>
<thead>
<tr>
<th>$P$</th>
<th>$T$ [s]</th>
<th>$S_{4\rightarrow P}$</th>
<th>$E_{4\rightarrow P}$</th>
<th>$P$</th>
<th>$T$ [s]</th>
<th>$S_{4\rightarrow P}$</th>
<th>$E_{4\rightarrow P}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.0089</td>
<td></td>
<td></td>
<td>4</td>
<td>0.0101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.0046</td>
<td>1.93</td>
<td>0.97</td>
<td>8</td>
<td>0.0052</td>
<td>1.95</td>
<td>0.97</td>
</tr>
<tr>
<td>16</td>
<td>0.0024</td>
<td>3.72</td>
<td>0.93</td>
<td>16</td>
<td>0.0027</td>
<td>3.78</td>
<td>0.94</td>
</tr>
<tr>
<td>32</td>
<td>0.0013</td>
<td>7.01</td>
<td>0.88</td>
<td>32</td>
<td>0.0014</td>
<td>7.26</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Two-Body Contact Problem (O. Sander, Berlin)

- Linear elasticity problem with contact.
- $P_1$ finite elements, mortar elements at the contact boundary.
- Coupled UGGrid and SGrid.
- Monotone (geometric) multigrid solver.
Summary

- Dune is a software framework for PDE numerics based on the following principles:
  - Separation of algorithms and data structures by abstract interfaces.
  - Efficient implementation using generic programming techniques in C++.
  - Reuse of existing finite element software.

- Future developments:
  - Abstractions for general finite element methods.
  - Solver framework for coupled multiphysics/multiscale problems.
  - More grids: parallel unstructured, parallel block structured, anisotropic local refinement.