The Distributed and Unified Numerics Environment (DUNE)

Outline

Andreas Dedner,

Department of Mathematics,
University of Warwick
www.warwick.ac.uk/go/dune

THE UNIVERSITY OF WARWICK

Outline

Construction of higher order approximation U_h

for solution $U : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^m$ of

$$\partial_t U(x,t) + \nabla \cdot (F(U(x,t),x,t) - D(U(x,t),x,t)\nabla U(x,t)) + A(U(x,t),x,,t)\nabla U(x,t) = S(U(x,t),x,t) + \mathcal{L}[U(\cdot,t)](x)$$

convection dominated case with non-local operator $\mathcal{L}$

Discretization with little restriction on

- space dimension:
  including $d > 3$ and problems on manifolds
- grid structure:
  including structured, unstructured, hanging nodes, distributed, networks...
- problem formulation:
  reuse of basic schemes requiring only problem data (e.g. Lax-Friedrichs)
  combined with specialized methods (if necessary)...

for applications we need all that including high efficiency

New Package?

Many PDE software packages, each with a particular set of features:

- Alberta: unstructured, simplicial, bisection refinement
- FEAST: block-structured, parallel
- DEALII: cube elements, shared memory parallelization
- Many more: DiffPack, IPARS, libMesh++, ...

Using one package it may be

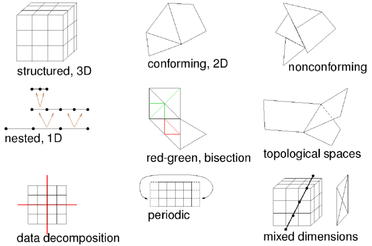
- impossible to have a certain feature
- very inefficient implementation for a certain applications

Extending the feature set is very difficult

Reason

Data and grid structure are very closing entangled and algorithms are implemented direclty on the basis of this particular grid data structure.

Grid Structures



- Cartesian
- conforming local adaptation
- adaptation with hanging nodes
- block adaptive
- hybrid element types
- ....

Numerical Methods

- Continuous Finite-Elements
- Discontinuous Finite-Elements
- Finite-Volumes
- Spectral methods
- Boundary Element methods
- ....

Solver

- Direct linear solvers
- Krylov type iterative solvers
- Large range of preconditioner
- Newton type methods
- Runge-Kutta ODE Solvers
- ....

Possible goals

Use available grid managers (e.g. Alberta, UG, p4est...), use available advantages (e.g. $O(1)$ storage) and use available packages (e.g. laspack, umfpack, Petsc...)

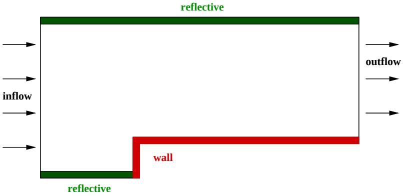
Compressible Euler equations

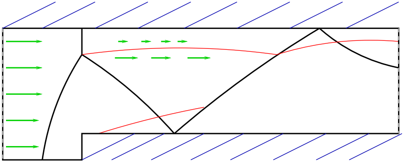
$$\vec{f}_1(\vec{u}) = \begin{pmatrix} \rho v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ \rho v_1 v_3 \\ (\rho\mathcal{E} + p)v_1 \end{pmatrix}, \quad \vec{f}_2(\vec{u}) = \begin{pmatrix} \rho v_2 \\ \rho v_2 v_1 \\ \rho v_2^2 + p \\ \rho v_2 v_3 \\ (\rho\mathcal{E} + p)v_2 \end{pmatrix}, \quad \vec{f}_3(\vec{u}) = \begin{pmatrix} \rho v_3 \\ \rho v_3 v_1 \\ \rho v_3 v_2 \\ \rho v_3^2 + p \\ (\rho\mathcal{E} + p)v_3 \end{pmatrix}.$$

conservative variables $\vec{u} = (\rho, \rho\vec{v}, \rho\mathcal{E})^T$, equations of state $p = p(\vec{u})$.

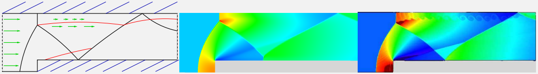
Test case: Mach 3 flow in a channel with a step (flow, shocks, contacts)

constant initial data: $\vec{u}_0(\dots) = (\rho_0, (\rho\vec{v}_1)_0, 0, 0, \rho\mathcal{E}_0) = (1.4, 4.2, 0, 0, 8.8)$





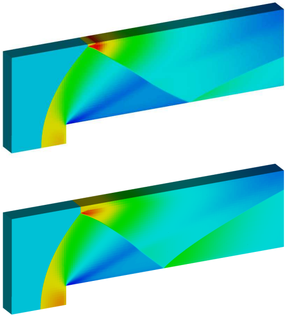
Forward facing step



Movie removed

Movie removed

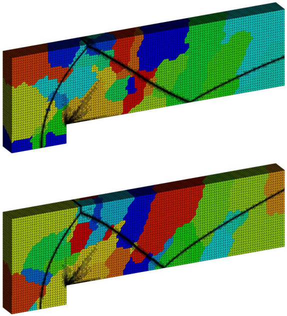
Density ρ



$t = 1.5$

$t = 2$

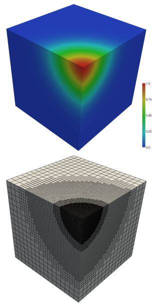
Grid on 32 processors



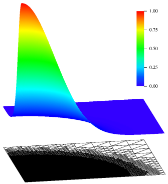
You should be able to do all the previous simulation on Wednesday...

Results for the Poisson equation $-\Delta u = f$

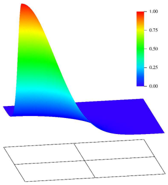
FEM Q_1



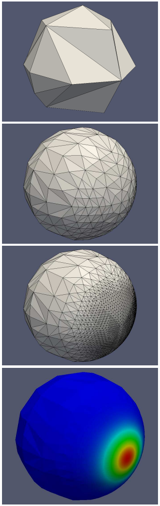
FEM P_1



FEM Q_6



FEM P_1



Moving Surface

Movie removed

Same finite-element code of different order on different realizations of the grid interface...

Outline

1. Introduction
2. DUNE– Grid Interface Library
3. DUNE-FEM– Discretization Interface Library
4. Generic Implementation of Numerical Schemes
5. Conclusions

History Overview

2002 Initiating DUNE meeting in Bonn

2003 First DUNE grid implementation was `SGrid` (structured)

2004 First implementation for an adaptive grid `AlbertaGrid`

2005 `ALUSimplexGrid` and `UGGrid`

2006 DUNE-GRID 1.0

2007 First DUNE Summer School (followed by at least one per year)

2008 Paper on DUNE-GRID (published in Computing)
(P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn., M. Ohlberger, O. Sander)

2009 Further grids: GEOMETRYGRID, CORNERPOINTGRID, GRIDGLUE

2009 DUNE-GRID 1.2.2, DUNE-FEM 1.0 (a major discretization module)
(A. Dedner, R. Klöfkorn., M. Nolte, M. Ohlberger)

2010 DUNE-GRID 2.0

2010 1. DUNE user meeting (planned about once every 1.5 - 2 years)

2011 DUNE-GRID 2.1, DUNE-FEM 1.2

2011 DUNE School (in Heiderlberg, Freiburg, Warwick, and Novosibirsk)

History Overview

2002 Initiating DUNE meeting in Bonn

2003 First DUNE grid implementation was `SGrid` (structured)

2004 First implementation for an adaptive grid `AlbertaGrid`

2005 `ALUSimplexGrid` and `UGGrid`

2006 DUNE-GRID 1.0

2007 First DUNE Summer School (followed by at least one per year)

2008 Paper on DUNE-GRID (published in Computing)
(P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn., M. Ohlberger, O. Sander)

2009 Further grids: GEOMETRYGRID, CORNERPOINTGRID, GRIDGLUE

2009 DUNE-GRID 1.2.2, DUNE-FEM 1.0 (a major discretization module)
(A. Dedner, R. Klöfkorn., M. Nolte, M. Ohlberger)

2010 DUNE-GRID 2.0

2010 1. DUNE user meeting (planned about once every 1.5 - 2 years)

2011 DUNE-GRID 2.1, DUNE-FEM 1.2

2011 DUNE School (in Heiderlberg, Freiburg, Warwick, and Novosibirsk)

History Overview

2002 Initiating DUNE meeting in Bonn

2003 First DUNE grid implementation was `SGrid` (structured)

2004 First implementation for an adaptive grid `AlbertaGrid`

2005 `ALUSimplexGrid` and `UGGrid`

2006 DUNE-GRID 1.0

2007 First DUNE Summer School (followed by at least one per year)

2008 Paper on DUNE-GRID (published in Computing)
(P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn., M. Ohlberger, O. Sander)

2009 Further grids: GEOMETRYGRID, CORNERPOINTGRID, GRIDGLUE

2009 DUNE-GRID 1.2.2, DUNE-FEM 1.0 (a major discretization module)
(A. Dedner, R. Klöfkorn., M. Nolte, M. Ohlberger)

2010 DUNE-GRID 2.0

2010 1. DUNE user meeting (planned about once every 1.5 - 2 years)

2011 DUNE-GRID 2.1, DUNE-FEM 1.2

2011 DUNE School (in Heiderlberg, Freiburg, Warwick, and Novosibirsk)

History Overview

2002 Initiating DUNE meeting in Bonn

2003 First DUNE grid implementation was `SGrid` (structured)

2004 First implementation for an adaptive grid `AlbertaGrid`

2005 `ALUSimplexGrid` and `UGGrid`

2006 DUNE-GRID 1.0

2007 First DUNE Summer School (followed by at least one per year)

2008 Paper on DUNE-GRID (published in Computing)
(P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn., M. Ohlberger, O. Sander)

2009 Further grids: GEOMETRYGRID, CORNERPOINTGRID, GRIDGLUE

2009 DUNE-GRID 1.2.2, DUNE-FEM 1.0 (a major discretization module)
(A. Dedner, R. Klöfkorn., M. Nolte, M. Ohlberger)

2010 DUNE-GRID 2.0

2010 1. DUNE user meeting (planned about once every 1.5 - 2 years)

2011 DUNE-GRID 2.1, DUNE-FEM 1.2

2011 DUNE School (in Heiderlberg, Freiburg, Warwick, and Novosibirsk)

Distributed and Unified Numerics Environment

DUNE– http://www.dune-project.org

- project language C++
- portability via ISO standard conformity (GCC 4.x, ICC 10.x)
- open source software (GPL with linking exception – same as GCC)
- current stable release: DUNE 2.1 (about to be released)

Distributed and Unified Numerics Environment

DUNE– http://www.dune-project.org

- project language C++
- portability via ISO standard conformity (GCC 4.x, ICC 10.x)
- open source software (GPL with linking exception – same as GCC)
- current stable release: DUNE 2.1 (about to be released)

DUNE Developers

Heidelberg

- Peter Bastian
- Markus Blatt
- Jorrit Fahlke

Berlin

- Oliver Sander
- Carsten Gräser

Warwick

- Andreas Dedner

Freiburg

- Robert Klöfkorn
- Martin Nolte

Münster

- Mario Ohlberger
- Christian Engwer

DUNE users in

- Aachen, Germany
- Berlin, Germany
- Magdeburg, Germany
- Stuttgart, Germany
- Graz, Austria
- Zürich, Switzerland
- Torntheim, Norway
- ...

Distributed and Unified Numerics Environment

Project Infrastructure:

- Subversion repository
- Doxygen in class docu
- Project homepage
- Mailing list
- Bug tracker
- *Automated testing system*
- Wiki for user discussion
- Fixed coding style

Testing environment

- *Central nightly builds* with detailed graphical reports
- *Decentralized testing environment* allowing user to test their own system and automatically submit reports
- *Performance testing environment* testing impact of changes using user define benchmark problems (not yet realized)

Decision Process:

1. Lots of discussions (mailing list, bug tracker, phone, meetings)
2. Annual developer meeting
3. Adding and removing feature relies on formal vote of *core* developers

Interface Changes:

1. Conservative in adding new feature (avoid feature creep)
2. Features are deprecated for one release before removal

Distributed and Unified Numerics Environment

Project Infrastructure:

- Subversion repository
- Doxygen in class docu
- Project homepage
- Mailing list
- Bug tracker
- *Automated testing system*
- Wiki for user discussion
- Fixed coding style

Testing environment

- *Central nightly builds* with detailed graphical reports
- *Decentralized testing environment* allowing user to test their own system and automatically submit reports
- *Performance testing environment* testing impact of changes using user define benchmark problems (not yet realized)

Decision Process:

1. Lots of discussions (mailing list, bug tracker, phone, meetings)
2. Annual developer meeting
3. Adding and removing feature relies on formal vote of *core* developers

Interface Changes:

1. Conservative in adding new feature (avoid feature creep)
2. Features are deprecated for one release before removal

The DUNE development is decentralized

Advantages:

- More manpower
- More points of view and applications
- More platforms

The DUNE development is decentralized

Advantages:

- More manpower
- More points of view and applications
- More platforms

Challenges:

Due to Spatial Separation (SS) and Academia (Ac)

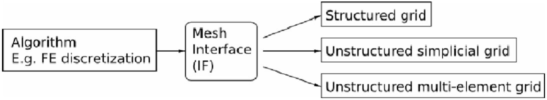
- True discussions are difficult (SS)
- Decision processes are difficult (SS)
- Feature creep (Ac)
- Difficult to produce documentation (Ac)
- No dedicated developers or manager (Ac)
- No real funding (Ac)

Distributed and Unified Numerics Environment

Design goals: Flexibility and Efficiency and Modularity

- Separate grid structure and data
- Define abstract interfaces for each part (grid, discrete functions...)
- Base interface on mathematical formulism

1. Determine what algorithms require from grid and data structure to operate efficiently
2. Formulate algorithms based on this interface
3. Provide different implementations of the interface



Distributed and Unified Numerics Environment

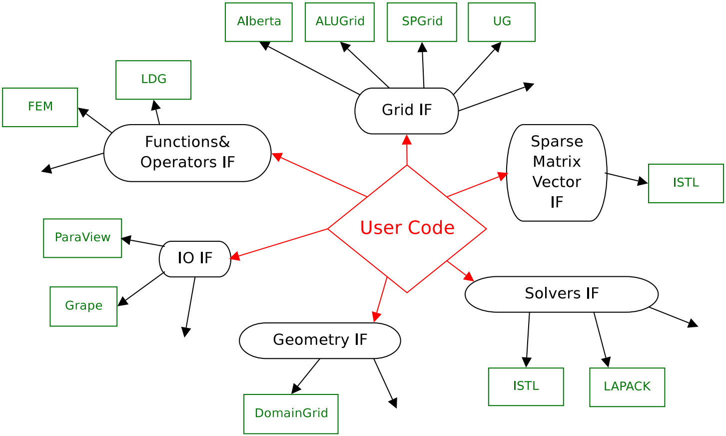
Design goals: Flexibility and Efficiency and Modularity

- Compile time selection of data structures (static polymorphism)
- Compiler generates code for each algorithm / data structure combination
- All optimizations apply, in particular inlining
- Possible through use of C++ templates

```
typedef GridImplementation Grid;
typedef Grid::LeafGridView GridView;
typedef FiniteElementSpace< GridView, order > FESpace;
typedef DiscreteFunction< FESpace > DiscreteFunction;
typedef EllipticOperator< Model, DiscreteFunction > Operator;
typedef CGInverseOperator< Operator > InverseOperator;

Model model;
RHSFunction f;
DiscreteFunction uh;
InverseOperator operator( model );
operator(f,uh);
```

Design goals: Flexibility and Efficiency and Modularity

Through interfaces, existing software can be easily used in own code - implement binding between external software and interface.



Distributed and Unified Numerics Environment

DUNE Core Modules (see www.dune-project.org)

- DUNE-COMMON – Basic Classes (MPI communicator, build-system, ...)
- DUNE-GRID – abstract Grid Interface and Implementations (ALBERTA, ALUGrid, UG, YaspGrid)
- DUNE-ISTL – Iterative Solver Template Library (BCRSMatrix, ILU, BiCG-Stab, AMG, ...)
- DUNE-LOCALFUNCTIONS – Basis Functions and Mappers (Lagrange basis functions, Raviart-Thomas, DG, DoF mappers, ...)
- DUNE-GRID-HOWTO – Tutorial for the DUNE-GRID module

DUNE Discretization Modules (see also www.dune-project.org)

- DUNE-FEM – developed in Freiburg, Warwick, and Münster
- DUNE-PDELAB – basically developed in Heidelberg

External libraries, e.g.,

- KASKADE-7 – developed in Berlin
- DUMUX – developed in Stuttgart
- OPM – developed in Trondheim

Distributed and Unified Numerics Environment

DUNE Core Modules (see www.dune-project.org)

- DUNE-COMMON – Basic Classes (MPI communicator, build-system, ...)
- DUNE-GRID – abstract Grid Interface and Implementations (ALBERTA, ALUGrid, UG, YaspGrid)
- DUNE-ISTL – Iterative Solver Template Library (BCRSMatrix, ILU, BiCG-Stab, AMG, ...)
- DUNE-LOCALFUNCTIONS – Basis Functions and Mappers (Lagrange basis functions, Raviart-Thomas, DG, DoF mappers, ...)
- DUNE-GRID-HOWTO – Tutorial for the DUNE-GRID module

DUNE Discretization Modules (see also www.dune-project.org)

- DUNE-FEM – developed in Freiburg, Warwick, and Münster
- DUNE-PDELAB – basically developed in Heidelberg

External libraries, e.g.,

- KASKADE-7 – developed in Berlin
- DUMUX – developed in Stuttgart
- OPM – developed in Trondheim

Distributed and Unified Numerics Environment

- DUNE consists of a set of highly integrated modules (libraries and applications).
- The buildsystem is based on `autoconf, automake & libtool`.
- Interaction and dependencies between the different modules is handled by `dunecontrol`.
- all necessary DUNE modules are assumed to be in the same directory
- each module contains file `dune.module` giving module name and dependency

```
Module: navier-stokes
Version: 0.9
Maintainer: dune@mathematik.uni-freiburg.de
Depends: dune-common (>= 2.0) dune-grid (>= 2.0) dune-fem (>= 1.1)
Suggests: dune-istl (>= 2.0) dune-localfunctions (>= 2.0) dune-spgrid
```

- a script `duneproject` for easy setup of new module

Main Module DUNE-GRID: Realization of Grid Interface

- ALUGrid: simplex grid in 2d/3d and cube grid in 3d with non-conform grid adaption, parallelization and dynamic load balancing
- AlbertaGrid: simplex grid in 2d/3d with conform grid adaption (bisection)
- GeometryGid: replace geometry of each element
- NetworkGrid: grid for 1D networks
- PrismGrid: tensor product prismatic grid ($\Omega \times [0, h]$)
- PSGrid: parallel simplex grid also on manifolds
- UGGrid: hybrid grid with non-conform adaption and red-green closure
- YaspGrid: parallel cartesian grid

Outline

DUNE-FEM: A Discretization Module

DUNE-FEM (dune.mathematik.uni-freiburg.de, release 1.1)

Idea: base implementation of numerical scheme on mathematical formalism

Interfaces for

- Function spaces and functions
- Discrete function spaces (combining function space and vector valued finite base function set)
- Discrete functions (with element wise representation, dof handling)
- Discrete spatial operators (with efficient operations, e.g., $+$ and $\circ$)
- Inverse operators (Newton, Krylov methods...)
- IMEX Runge-Kutta methods for time dependent problems
- Automatic handling of grid adaptation, parallelization, and load balancing

DUNE-FEM: A Discretization Module

1. Discrete spaces and discrete functions
   - discrete function spaces (Lagrange, DG, ...)
   - discrete functions (adaptive DF, block vector DF, ...)
   - caching of basis functions
2. Discretization schemes
   - Lagrange FEM (generic, arbitrary order)
   - Finite Volume (first and second order)
   - Discontinuous Galerkin (orthonormal basis functions, up to order 8)
3. implemented Runge Kutta solvers
   - explicit Strong-Stability-Preserving Runge Kutta (SSP-RK) up to ord. 3
   - Diagonally Implicit Runge Kutta (DIRK) methods up to order 3
   - Semi Implicit Runge Kutta (SIRK) methods up to order 3
4. Misc
   - Restriction/prolongation strategies
   - DoF handling (automatic resize and DoF-compress)
   - Data I/O and check-pointing
   - Communication patterns
   - ...

A D, R. Klöfkorn, M. Nolte, M. Ohlberger.
A generic interface for parallel and adaptive scientific computing: Abstraction principles and the DUNE-FEM module.
Computing, 2010.
other contributors: S. Brdar, M. Kränkel, Ch. Gersbacher, ...

DUNE-FEM: A Discretization Module

The DUNE-FEM-HOWTO

- Getting started, or how to calculate a Lagrange interpolation
- A Finite Volume scheme demonstrates the implementation of a first order Finite Volume scheme using DUNE-FEM.
- The Poisson problem is an example for calculating a solution of the Poisson problem using conforming Finite-Elements
- LDG for Advection-Diffusion equations is an example for implementing a Local Discontinuous Galerkin solver for advection-diffusion problems
- The Stokes problem implement a Stokes solver in the DUNE-FEM context.
- Data I/O and check pointing shows how to incorporate data I/O and check pointing into your simulation code.

The DUNE-FEM-SCHOOL

- Introduction to generic programming in C++
- Introduction to the DUNE-GRID module
- Introduction to the DUNE-FEM module
- Finite-Volume for conservation laws
- Finite-Element for linear elliptic and parabolic problems
- Discontinuous-Galerkin for non-linear evolution equations

Outline

Discontinuous Galerkin Method, Approach I

find piecewise polynomial approximation U_h of

$$\partial_t U(x,t) + \nabla \cdot (F(U(x,t),x,t) - D(U(x,t),x,t)\nabla U(x,t)) = 0$$

$$\begin{aligned}\int_K \partial_t u_K \varphi = &\int_K (F(u_K) \cdot \nabla\varphi) - \int_{\partial K} \widehat{F}(u) \cdot \mathbf{n}_K \varphi \\ &- \int_K (D(u_K)\nabla u_K \cdot \nabla\varphi) + \int_{\partial K} \widehat{D}_1(u) \cdot \mathbf{n}_K\varphi + \widehat{D}_2(u) \cdot \nabla\varphi \\ &=: - < \mathcal{L}_K^A[u_h], \varphi > - < \mathcal{L}_K^D[u_h], \varphi >\end{aligned}$$

- hyperbolic operator $\mathcal{L}_K^A \approx \nabla \cdot F(u)$ on one element K, possibly with explicit time step
  $\widehat{F}$: suitable upwind flux for advection requiring only data on direct neighbors
- elliptic operator: $\mathcal{L}_K^D \approx -\nabla \cdot D(u)\nabla u$ on element K, possibly with implicit time step
  $\widehat{D}_1, \widehat{D}_2$: suitable flux for diffusion requires only data on direct neighbors
- use IMEX Runge-Kutta scheme

Discontinuous Galerkin Method, Approach I

find piecewise polynomial approximation U_h of

$$\partial_t U(x,t) + \nabla \cdot \big(F(U(x,t),x,t) - D(U(x,t),x,t)\nabla U(x,t)\big) = 0$$

$$\partial_t u_h = -\left(\mathcal{L}^A[u_h] + \mathcal{L}^D[u_h]\right)$$

$$\begin{aligned} \mathcal{L}_K^A[u_h] &\approx \nabla \cdot F(U(x,t),x,t) \\ \mathcal{L}_K^D[u_h] &\approx \; - \nabla \cdot D(U(x,t),x,t)\nabla U(x,t) \end{aligned}$$

- $\mathcal{L}^A$ is an approximation for a first order hyperbolic equation (e.g., Euler equation)
- $\mathcal{L}^D$ is an approximation for an elliptic or parabolic equation (e.g., Laplace/Heat equation)
- Suitable ODE for time integration

Description

Model: function F and D

Discrete Model: $\widehat{F}$ and $\widehat{D}_1, \widehat{D}_2$ (using F, D)

Problem: other functions, e.g., initial data

Discontinuous Galerkin Method, Approach II

find piecewise polynomial approximation U_h of

$$\partial_t U(x,t) + \nabla \cdot (f(U(x,t),x,t) - D(U(x,t),x,t)\nabla d(U(x,t))) = 0$$

Rewritte as first order system for (σ, u) and use $\mathcal{L}^A$:

$$\sigma(x,t) + \nabla d(U(x,t)) = 0, \qquad \partial_t U(x,t) + \nabla \cdot (f(U(x,t),x,t) + D(U(x,t),x,t)\sigma(x,t)) = 0.$$

$$\sigma_h = -\mathcal{L}_1^A[u_h], \qquad \partial_t u_h = -\mathcal{L}_2^A[u_h, \sigma_h].$$

or $\qquad \partial_t u_h = \mathcal{L}^{AD}[u_h] := \mathcal{L}_2^A[u_h, \mathcal{L}_1^A[u_h]].$

Use the same operator tow times with different flux F:

$$\mathcal{L}_1^A[u_h] \approx \nabla \cdot F(U(x,t),x,t) \qquad \text{with } F(U,x,t) = d(U))$$

$$\mathcal{L}_2^A[u_h] \approx \nabla \cdot F(U(x,t),x,t) \qquad \text{with } F(U,\sigma,x,t) = f(U(x,t),x,t) + D(U(x,t),x,t)\sigma$$

Description

Model: function F, D and d

Discrete Model 1: $\widehat{F}_1$ (using d)

Discrete Model 2: $\widehat{F}_2$ (using F, D)

Problem: other functions, e.g., initial data

Approach I and II Compared

DG Spatial Operators for

$$\mathcal{L}[u] = -\nabla \cdot \left(f(U(x,t),x,t) - D(U(x,t),x,t)\nabla d(U(x,t))\right)$$

Approach I

$$\mathcal{L}^{AD}[u_h] := \mathcal{L}^{D}[u_h] + \mathcal{L}_{A}[u_h]$$

Model: function F and D ($d \equiv 1$)

Discrete Model: $\widehat{F}$ and $\widehat{D}_1, \widehat{D}_2$ (using F, D)

Problem: other functions, e.g., initial data

Approach II

$$\mathcal{L}^{AD}[u_h] := \mathcal{L}_2^{A}[u_h, \mathcal{L}_1^{A}[u_h]]$$

Model: function f, D and d

Discrete Model 1: $\widehat{F}_1$ (using d)

Discrete Model 2: $\widehat{F}_2$ (using F, D)

Problem: other functions, e.g., initial data

Two-phase flow in porous media

Global pressure, global velocity (incompressible, no gravity)

$$
\begin{aligned}
-\nabla \cdot \big(\lambda(s)\mathbf{K}\nabla p\big) &= 0, \quad \text{in } \Omega \\
\vec{u} &= -\lambda(s)\mathbf{K}\nabla p, \quad \text{in } \Omega \\
\phi\, \partial_t s + \nabla \cdot \vec{u} f_w(s) - \nabla \cdot \big(\vec{D}(s)\nabla s\big) &= 0, \quad \text{in } (0,T] \times \Omega \subset \boldsymbol{R}^d, \\
s(0,\cdot) &= s_0(\cdot), \quad \text{in } \Omega.
\end{aligned}
$$

Pressure p, velocity $\vec{u}$, and saturation s

Given s^n, $n \geq 0$, we calculate:

1. $-\nabla \cdot \big(\lambda(s^n)\mathbf{K}\nabla p^{n+1}\big) = 0$, $(\mathcal{L}^D)$
2. $\vec{u}^{n+1} = \mathcal{P}_{\text{H-div}}(-\lambda(s^n)\mathbf{K}\nabla p^{n+1})$ (speciallized: enforce continuous normal velocity)
3. $s^{n+1} = \mathcal{RK}(s^n; p^{n+1}, \vec{u}^{n+1})$ $(\mathcal{L}^A\mathcal{L}^D$ or $\mathcal{L}^{AD})$

Description

Model for 1: $\vec{D}, \lambda, K$, and f_w

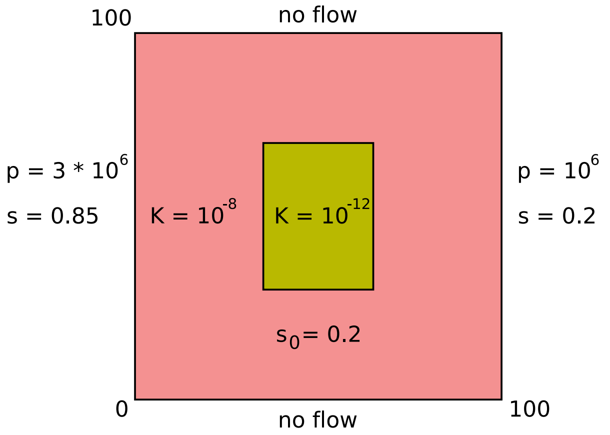
Discrete Model 1: $D = \lambda(s)K$

Discrete Model 2: $D = \vec{D}(s), \widehat{F} = \vec{u} f_w(s)$

Problem: other functions, e.g., initial data

Two-phase flow in porous media

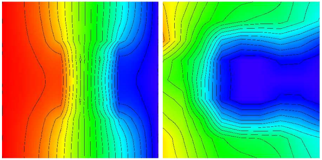
Initial and boundary data



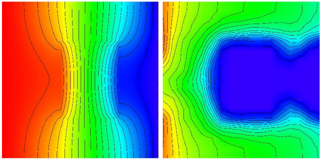
(Epshteyn & Riviere, Appl. Numer. Math., 2007)

Two-phase flow in porous media

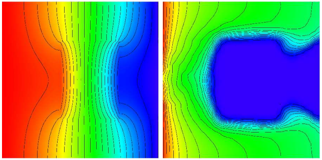
$(k_p, k_s) = (2, 1), T = 550$



16×16

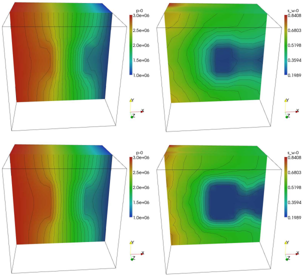


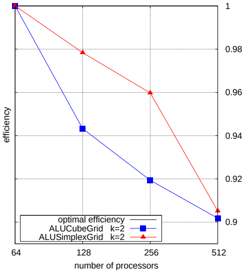
32×32



64×64

phd thesis R. Klöfkorn (2009)





Free surface hydrostatic flow

Time dependent domain

$$\Omega(t) = \left\{ (\boldsymbol{x}, z)^T \in \boldsymbol{R}^d \ : \ \boldsymbol{x} \in \Omega_{\boldsymbol{x}},\ b(\boldsymbol{x}) < z < b(\boldsymbol{x}) + h(\boldsymbol{x}, t) \right\}$$

$\Omega_{\boldsymbol{x}} \subset \boldsymbol{R}^{d-1}$, $b\colon \Omega_{\boldsymbol{x}} \to \mathbb{R}$ is then bottom topography, and $h(t, \cdot)\colon \Omega_{\boldsymbol{x}} \to \mathbb{R}$ is the free surface. The 3d velocity field $\mathbf{u} = (\mathbf{u}_{\boldsymbol{x}}, w)^T$ satisfies

$$\begin{aligned}
\partial_t \mathbf{u} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) + \nabla p &= (0, 0, -g)^T + \text{visc.} && \text{in } \Omega(t), \\
\nabla \cdot \mathbf{u} &= 0 && \text{in } \Omega(t), \\
\partial_t h + \mathbf{u}_{\boldsymbol{x}} \cdot \nabla (b + h) &= w && \text{in } \Omega_{\boldsymbol{x}}, (z = b(\boldsymbol{x}) + h(\boldsymbol{x}, t)), \\
\mathbf{u}_{\boldsymbol{x}} \cdot \nabla b &= w && \text{in } \Omega_{\boldsymbol{x}}, (z = b(\boldsymbol{x})),
\end{aligned}$$

where $g > 0$ is the gravitational constant.
With $\partial_t w + (\mathbf{u} \cdot \nabla) w \approx 0$ we arrive (with scaling arguments) at the hydrostatic pressure equation:

$$\partial_z p = -g, \qquad p(\boldsymbol{x}, z, t) = -g(z - h(\boldsymbol{x}, t) - b(\boldsymbol{x}))$$

Integration of divergence constraint over z leads to *3D shallow water system*.

Free surface hydrostatic flow

Time dependent domain

$$\Omega(t) = \left\{ (\boldsymbol{x}, z)^T \in \boldsymbol{R}^d \ : \ \boldsymbol{x} \in \Omega_{\boldsymbol{x}},\ b(\boldsymbol{x}) < z < b(\boldsymbol{x}) + h(\boldsymbol{x}, t) \right\}$$

With shallow water scaling the free surface $h(t, \cdot) : \Omega_{\boldsymbol{x}} \to \mathbb{R}$ and the 3d velocity field $\mathbf{u} = (\mathbf{u}_{\boldsymbol{x}}, w)^T : \Omega(t) \to \boldsymbol{R}^d$ satisfy

$$\begin{aligned}
\partial_t h + \nabla_{\boldsymbol{x}} \cdot \left(\int_b^{b+h} \mathbf{u}_{\boldsymbol{x}} dz \right) &= 0 && \text{in } \Omega_{\boldsymbol{x}}, \\
\partial_t h\mathbf{u}_{\boldsymbol{x}} + \nabla \cdot (h\mathbf{u} \otimes \mathbf{u}_{\boldsymbol{x}}) + g\nabla_{\boldsymbol{x}} h^2 &= -gh\nabla_{\boldsymbol{x}} b + \text{visc.} && \text{in } \Omega(t), \\
\partial_z w &= -\nabla_{\boldsymbol{x}} \cdot \mathbf{u}_{\boldsymbol{x}} && \text{in } \Omega(t),
\end{aligned}$$

Discretization (in space):

1. compute the integrals of the horizontal velocities $\bar{u} = \int_b^{b+h} \mathbf{u}_{\boldsymbol{x}} dz$ (special operator using special *prism grid*)
2. compute the vertical velocity w (integration in z) (special operator using special *prism grid*)
3. apply advection-diffusion discretization for $(h, \mathbf{u}_{\boldsymbol{x}})$ ($\mathcal{L}^{AD}$)

Description

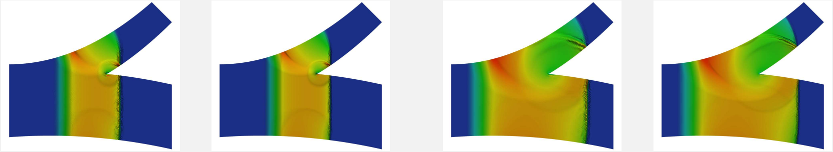
Model: $F = (\bar{u}, h\mathbf{u} \otimes \mathbf{u}_{\boldsymbol{x}} + gh^2)$ and $D = \text{visc}$

Discrete Model: $\widehat{F}, D$

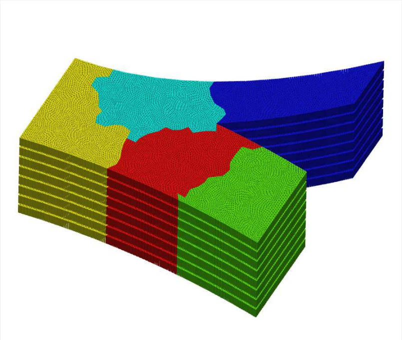
Problem: other functions, e.g., initial data

Simulation Results

early time | later time



parallel prisma grid



diploma thesis C. Gersbacher

Using Interface Classes

Operators < DiscreteModel >: examples $\mathcal{L}^A$, $\mathcal{L}^D$, $\mathcal{L}^{AD}$
DiscreteModel < Model >: part of discretization which is not part of continuous problem (numerical fluxes)
Model< grid dimension >: first part of continuous problem
Problem< grid dimension >: second part of continuous problem
Often only Problem needs to be implemented (e.g., use EulerModel to solve Euler equations)
Otherwise often only Problem and Model needs to be implemented

Difference between Problem and Model

Distinction is somewhat arbitrary, general idea

Problem: use *dynamic polymorphism* to allow runtime selection

Model: use *static polymorphism* for maximal efficiency

ODESolver < Operator >: $u^n \to u^{n+1}$ to solve $\partial_t u = \mathcal{L}[u]$
InverseOperator < Operator >: $u = \mathcal{L}^{-1}[f]$ to solve $\mathcal{L}[u] = f$

Using Interface Classes

Operators < DiscreteModel >: examples $\mathcal{L}^A$, $\mathcal{L}^D$, $\mathcal{L}^{AD}$
DiscreteModel < Model >: part of discretization which is not part of continuous problem (numerical fluxes)
Model< grid dimension >: first part of continuous problem
Problem< grid dimension >: second part of continuous problem
Often only Problem needs to be implemented (e.g., use EulerModel to solve Euler equations)
Otherwise often only Problem and Model needs to be implemented

Difference between Problem and Model

Distinction is somewhat arbitrary, general idea

- Problem: use *dynamic polymorphism* to allow runtime selection
- Model: use *static polymorphism* for maximal efficiency

ODESolver < Operator >: $u^n \to u^{n+1}$ to solve $\partial_t u = \mathcal{L}[u]$
InverseOperator < Operator >: $u = \mathcal{L}^{-1}[f]$ to solve $\mathcal{L}[u] = f$

Using Interface Classes

Operators < DiscreteModel >: examples $\mathcal{L}^A$, $\mathcal{L}^D$, $\mathcal{L}^{AD}$
DiscreteModel < Model >: part of discretization which is not part of continuous problem (numerical fluxes)
Model< grid dimension >: first part of continuous problem
Problem< grid dimension >: second part of continuous problem
Often only Problem needs to be implemented (e.g., use EulerModel to solve Euler equations)
Otherwise often only Problem and Model needs to be implemented

Difference between Problem and Model

Distinction is somewhat arbitrary, general idea

- Problem: use *dynamic polymorphism* to allow runtime selection
- Model: use *static polymorphism* for maximal efficiency

ODESolver < Operator >: $u^n \to u^{n+1}$ to solve $\partial_t u = \mathcal{L}[u]$
InverseOperator < Operator >: $u = \mathcal{L}^{-1}[f]$ to solve $\mathcal{L}[u] = f$

Example code for adaptation and communication

```
// construct view $\grid$ of a hierarchical grid $\hgrid$ (e.g. leaf view)
GridPartType grid( hgrid );
// construct a discrete function space $\discfuncspace$ (e.g. lagrange space)
DiscreteSpaceType space( grid );
// create the solution $u$ (providing dof storage)
DiscreteFunctionType u( "solution", space );

// communicate $u$ using the space's default communication
u.communicate();

// type of the default restriction and prolongation operator
typedef RestrictProlongDefault< DiscreteFunctionType > RestrictProlongType;
// type of the adaptation manager (more than one discrete functions using
    Tuples
typedef AdaptationManager< HGridType, RestrictProlongType >
    AdaptationManagerType;

// create restriction and prolongation operator for $u$ and the adaptation
    manager
RestrictProlongType uRestrictProlong( u );
AdaptationManagerType adaptationManager( hgrid, uRestrictProlong );

// mark grid for refinement and coarsening using some external method
    \code{mark}
mark( hgrid, u );

// adapt the grid with automatic restriction and prolongation of the discrete
    function $u$
// this also includes dynamic load-balancing if supported
adaptManager.adapt();
```

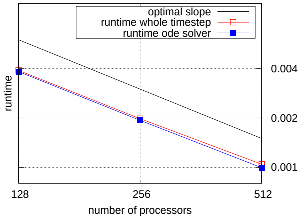
Example code showing mass matrix assembly

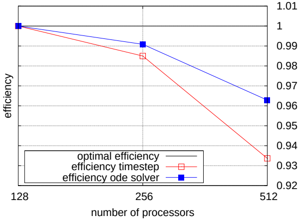
```
// iterate over the grid $\grid$
for( IteratorType it = space.begin(); it != end ; ++it )
{
  const EntityType &entity = *it;

  // get local function $u_\elem$ (proxy object)
  LocalFunctionType uLocal = u.localFunction( entity );
  // obtain local operator $M_{\elem,\elem}$
  LocalMatrixType MLocal = M.localMatrix( entity, entity );

  // obtain the local base function set $\basefuncset_\elem$
  const BaseFunctionSetType &baseFunctionSet = space.baseFunctionSet( entity );
  const unsigned int numBaseFunctions = baseFunctionSet.numBaseFunctions();

  // compute the integrals $\int_\elem \varphi_i\varphi_j$ and $\int_\elem
       f\varphi_i$ using a quadrature with base function caching
  CachingQuadrature<GridPartType, 0> quadrature( entity, 2*space.order()+1 );
  const unsigned int nop = quadrature.nop();
  for( unsigned int qp = 0; qp < nop; ++qp )
  {
    // evaluate all basis functions at once
    baseFunctionSet.evaluateAll( quadrature[ qp ], values );

    // add $\int_\elem \varphi_i\varphi_j$ to the operator $M$
    for( unsigned int i = 0; i < numBaseFunctions; ++i )
      for( unsigned int j = 0; j < numBaseFunctions; ++j )
        MLocal.add( i, j, (values[ i ] * values[ j ]) );
  }
}
```

Outline

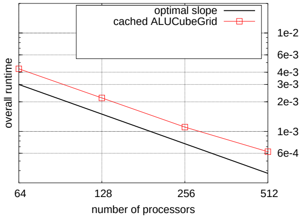
Parallel Efficiency (strong)

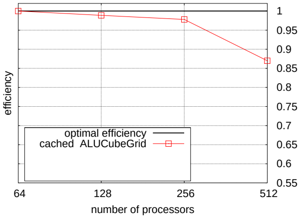
Time explicit DG scheme for Euler's equation





Poisson equation





Newer Developments

1. Definition of meta grids, i.e., use a given DUNE grid to define a new one.
   - prismatic grid (unstructured in xy-plane, structured in z-plane)
   - GeometryGrid which replaces the geometry of each element of a given grid by a new geometric mapping (higher order...)
2. Moving grids
3. grid-glue: combine different grids with each other (in parallel)
4. Generic construction of finite-element spaces based on definition of *nodal variables*

Example of grid-glue (provided by Oliver Sander, FU Berlin)