WWU MÜNSTER

DUNE – Collaborating via Interfaces

Christian Engwer, University of Münster

SIAM CSE — February 26th, 2019

living.knowledge

Collaborative Development

...for Scientific Software

Goals:

- Involve other developers
- Basis for future research
- Improved code quality
- Reproducibility

...

Source: wikimedia, License: CC BY-SA 4.0

Collaborative Development

...for Scientific Software

Goals:

- Involve other developers
- Basis for future research
- Improved code quality
- Reproducibility

...

Issues:

- Creadability of work
- Non-scientific tasks
- Long-term maintainance
- Funding

...

Collaborative Development
...for Scientific Software

Goals:

- Involve other developers
- Basis for future research
- Improved code quality
- Reproducibility

...

Issues:

- Creadability of work
- Non-scientific tasks
- Long-term maintainance
- Funding

...

DUNE http://www.dune-project.org/

[...] a modular toolbox

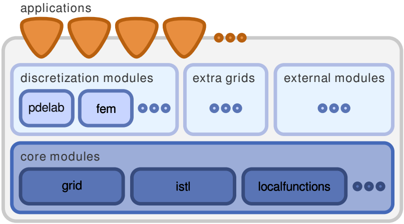
for solving partial differential equations (PDEs)

with grid-based methods [...]

Outline

What is DUNE?

- Domain specific interfaces
- Modular Code structure
  - Pick what you need.
  - Separation of concerns
- Generic programming techniques
- Portable (C++, Unix, cmake)
- Open Development Process
- Free Software Licence (LGPL + runtime exception)



[Bastian, Blatt, Dedner, Engwer, Klöfkorn, Kornhuber, Ohlberger, Sander 2008]

A basis for high quality research

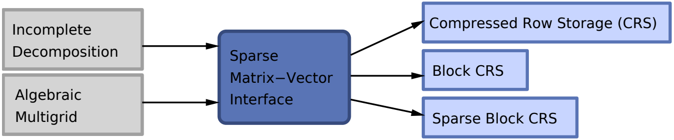
- Papers directly citing DUNE: $\sim$ 430 (since 2008)
- Higher-level packages built on-top of DUNE:
  - DuMu^X: $\sim$ 230 citations (since 2011)
  - BEM++: $\sim$ 130 citations (since 2015)
  - Kaskade-7: 31 publications (since 2008)
  - duneuro, ...
- Adoption in Industry:
  - Open-Porous-Media project (IRIS, SINTEF, Equinor, Ceetron Solutions)
  - BETL *(reimplemented DUNE grid interface)*
  - severaler smaller projects ...

Design Goals

Flexibility: Seperation of data structures and algorithms.

Efficiency: Generic programming techniques.

Legacy Code: Reuse existing finite element software.

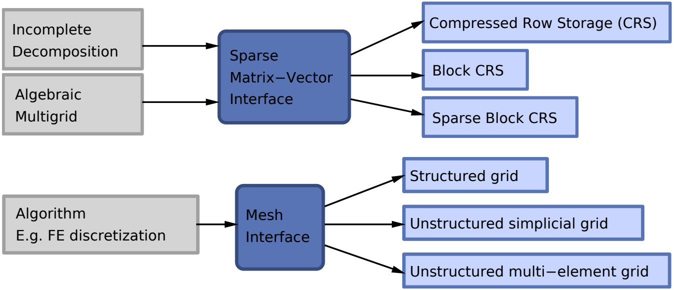


Design Goals

Flexibility: Seperation of data structures and algorithms.

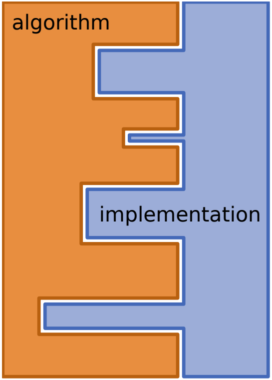
Efficiency: Generic programming techniques.

Legacy Code: Reuse existing finite element software.



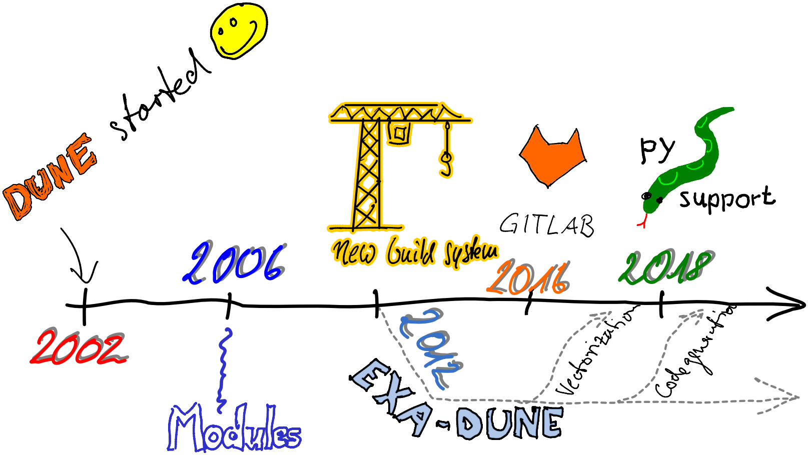
Fine- vs. Coarse-grained interfaces

Implementation with generic programming techniques.



- Static Polymorphism → Compile-time selection of data structures.
- Compiler generates code for each (algorithm,data structure) combination.
- Allows interfaces with fine granularity.
- All optimizations apply, in particular function inlining.
- see i.e. STL, Blitz++, MTL,...
- and Thesis of Gundram Berti (2000): Concepts for grid based algorithms.

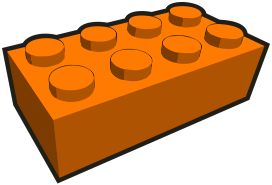
Some historic remarks



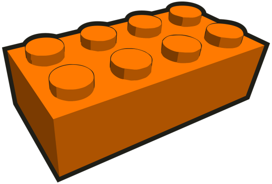
2 Interface & Modules

DUNE ecosystem

- modular structure
- write your own DUNE modules
- available under different licenses



DUNE ecosystem



- modular structure
- write your own DUNE modules
- available under different licenses

- Discretization Modules

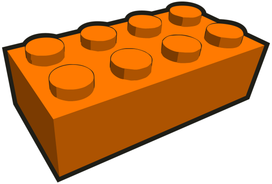
  dune-pdelab: discretization module based on dune-localfunctions.

  dune-fem: Alternative implementation of finite element functions.

  dune-functions: A new initiative to provide unified interfaces for functions and function spaces.

DUNE ecosystem

- modular structure
- write your own DUNE modules
- available under different licenses



- Discretization Modules
- Additional Grid Implementations

dune-grid-glue: allows to compute overlapping and nonoverlapping couplings of Dune grids, as required for most domain decomposition algorithms.

dune-subgrid: allows you to work on a subset of a given DUNE grid.

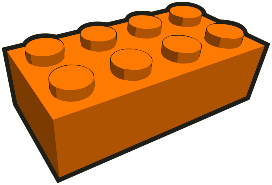
dune-foamgrid: non-manifold grids of 1d or 2d entities in higher-dimensional world.

dune-prismgrid: is a tensorgrid of a 2D simplex grid and a 1D grid.

dune-cornerpoint: a cornerpoint mesh, compatible with the grid format of the ECLIPSE reservoir simulation software.

...

DUNE ecosystem

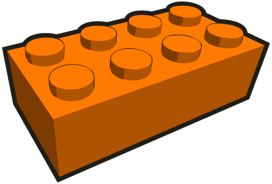


- modular structure
- write your own DUNE modules
- available under different licenses

- Discretization Modules
- Additional Grid Implementations
- Extension Modules

dune-python	python bindings for centrral DUNE components
dune-typetree	classes to organise types in trees
dune-dpg	construct optimal Discontinuous-Petrov-Galerkin test spaces
dune-tpmc	cut-cell construction using level-sets
...	

DUNE ecosystem



- modular structure
- write your own DUNE modules
- available under different licenses

- Discretization Modules
- Additional Grid Implementations
- Extension Modules

→ allow people to...

  - get credit for their invoations
  - experiment without breaking the core
  - develop at different speeds

A Package System

dunecontrol

- control of module-interplay
- suggestions & dependencies
- intergrates with cmake & git
- works with Linux, Mac and Mingw

Source: gnome

A Package System

`dunecontrol`

- control of module-interplay
- suggestions & dependencies
- intergrates with cmake & git
- works with Linux, Mac and Mingw

Note: Dependencies should form a DAG

Source: gnome

A Package System

`dunecontrol`

- control of module-interplay
- suggestions & dependencies
- intergrates with cmake & git
- works with Linux, Mac and Mingw

Source: gnome

Note: Dependencies should form a DAG

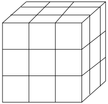
`dunecontrol cmake`
: configure packages via cmake, include necessary path information

`dunecontrol make`
: build packages in correct order

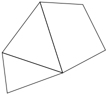
... works without `make install`

The Grid Interface

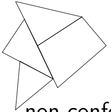
Designed to support a wide range of Grids



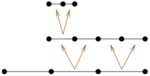
structured



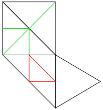
conforming



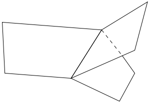
non conforming



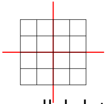
nested, 1D



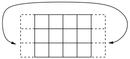
red-green, bisektion



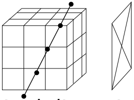
manifolds



parallel data decomposition



periodic



mixed dimensions

The Grid Interface

Main contribution: a well defined generic interface

- General hierarchic meshes
- Dimension independent algorithms
- Model grid entities (Cells, Vertices, Faces, ...)
- Codimension 1 Intersections (e.g. for dG or FV methods)
- Separation of topological structure and geometry information
- Separation of mesh and user data

Code wise...

- 5 grid implementations in `dune-grid`
- $\gg$ 10 external implementations
- including things like polyhedral meshes & cornerpoint meshes

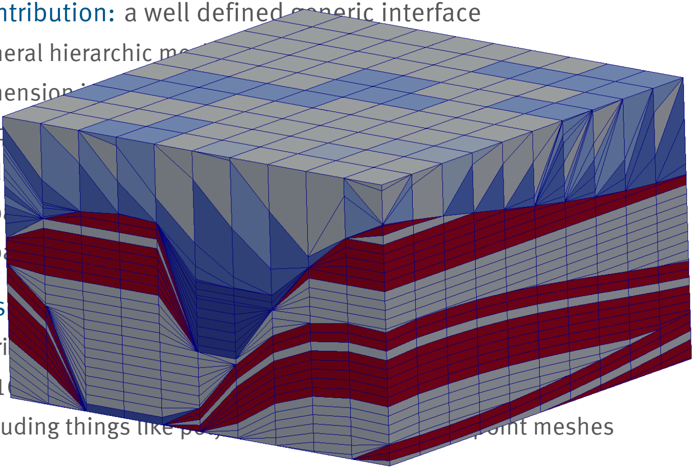
The Grid Interface

Main contribution: a well defined generic interface

- General hierarchic m
- Dimension i
- Mo
- Cod
- Sep
- Sep

Code wis

- 5 gri
- $\gg$ 1
- including things like p… …oint meshes



Generic Algorithms

```
#include <dune/grid/yaspgrid.hh>
...

using Grid = Dune::YaspGrid<2>;
Grid grid({4,4},{1.0,1.0},{false,false});
auto gv = grid.leafGridView();

double value = 0.0 , volume = 0.0;
for (const auto& cell : elements(gv)) {
  auto geo = cell.geometry();
  // compute average
  value += f(cell.center());
  volume += cell.volume();
  // access neighbours
  for (const auto& is : intersections( gv , cell )) {
    if (is.boundary()) {
      // handle potential Neumann boundary
    }
    if (is.neighbor()) {
      // code for Discontinuous Galerkin or Finite Volume
    }
  }
}
```

Grid Interface, Modules and development

Interface

- Major effort was to define such an interface

Grid Interface, Modules and development

Interface

- Major effort was to define such an interface
- Clear requirement list for new implementations
- Tests allow to verify new implementations
- Well defined entry point for new developers

Grid Interface, Modules and development

Interface

- Major effort was to define such an interface
- Clear requirement list for new implementations
- Tests allow to verify new implementations
- Well defined entry point for new developers

Modules

- Allow experimenting with new implementations / concepts
- Allow different licenses and even commercial use

Grid Interface, Modules and development

Interface

- Major effort was to define such an interface
- Clear requirement list for new implementations
- Tests allow to verify new implementations
- Well defined entry point for new developers

Modules

- Allow experimenting with new implementations / concepts
- Allow different licenses and even commercial use
- Improved visibility for new new development
- Integration, but possibly separate distribution paths

3 Discussion

Turning Points, Open Challenges, Lessons learned

Turning Points

- Modularization
- Switch to *gitlab* $\rightarrow$ new workflow + fully stable master
- EXA-DUNE $\rightarrow$ vectorization, threading
- Code-generation for FEM kernels
- python bindings $\rightarrow$ incorporate DUNE in under-grad education

Lessons learned

- Interfaces allow separation of responsibilities
- Interfaces as entry points for new contributors
- Modularization help visibility of new contributors
- Modularization to avoid license issues

Lessons learned

- Interfaces allow separation of responsibilities
- Interfaces as entry points for new contributors
- Modularization help visibility of new contributors
- Modularization to avoid license issues

- Quality toolchain improves productivity
- Try to avoid long term forks
  (we had reasons, but the EXA-DUNE fork turned out to be a stupid idea)

Open Challenges

- Interoperability with other C++ codes
  - Fine grained interface...
  - ... require header libraries

- Not one configuration for everybody
- Growing project size

Open Challenges

- Interoperability with other C++ codes
  - Fine grained interface...
  - ... require header libraries
  - Clear interface, but...
  - ... more complicated that C libraries
- Not one configuration for everybody
- Growing project size

Open Challenges

- Interoperability with other C++ codes
  - Fine grained interface...
  - ... require header libraries
  - Clear interface, but...
  - ... more complicated that C libraries
  - No immediate conflicts, but...
  - ... interface impose rich data types.
- Not one configuration for everybody
- Growing project size

Open Challenges

- Interoperability with other C++ codes
- Not one configuration for everybody
  - difficult to provide preinstalled packages
  - increased complexity for tests (test with and without features enabled)
- Growing project size

Open Challenges

- Interoperability with other `C++` codes
- Not one configuration for everybody
  - difficult to provide preinstalled packages
  - increased complexity for tests (test with and without features enabled)

  → not easy to integrate with `spack`
- Growing project size

Open Challenges

- Interoperability with other C++ codes
- Not one configuration for everybody
- Growing project size
  - Growing complexity for new developers
  - Increased maintenance effort

Open Challenges

- Interoperability with other C++ codes
- Not one configuration for everybody
- Growing project size
    - Growing complexity for new developers
    - Increased maintenance effort
    - Funding for support and maintenance
    - Costs of dev-ops and infrastructure

Summary

Lessons learned

- Interfaces allow separation of responsibilities
- Interfaces as entry points for new contributors
- Modularization help visibility of new contributors
- Modularization to avoid license issues

Open Challenges

- Interoperability with other C++ codes
- Not one configuration for everybody
- Growing project size

Thanks you for your attention