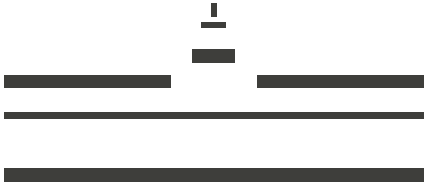


Westfälische
Wilhelms-Universität
Münster

Extending DUNE: The `dune-xt` modules

living.knowledge
WWU Münster

T. Leibner, R. Milk, M. Ohlberger, F. Schindler

Juli 5th, 2016

Introduction

What is dune-xt??

- A much cooler name than `dune-stuff`

What is dune-xt??

- A much cooler name than dune-stuff
- 4 modules: dune-xt-common, dune-xt-la, dune-xt-grid, dune-xt-functions

What is dune-xt??

- A much cooler name than `dune-stuff`
- 4 modules: `dune-xt-common`, `dune-xt-la`, `dune-xt-grid`, `dune-xt-functions`
- re-usable collections of utilities that solve everyday problems

What is dune-xt??

- A much cooler name than `dune-stuff`
- 4 modules: `dune-xt-common`, `dune-xt-la`, `dune-xt-grid`, `dune-xt-functions`
- re-usable collections of utilities that solve everyday problems
- dual-licensed as BSD-2 and GPL 2+ w/ runtime exception (the core modules' license)

What is dune-xt??

- A much cooler name than `dune-stuff`
- 4 modules: `dune-xt-common`, `dune-xt-la`, `dune-xt-grid`, `dune-xt-functions`
- re-usable collections of utilities that solve everyday problems
- dual-licensed as BSD-2 and GPL 2+ w/ runtime exception (the core modules' license)
- read more: `https://arxiv.org/abs/1602.08991`

What is dune-xt??

- A much cooler name than `dune-stuff`
- 4 modules: `dune-xt-common`, `dune-xt-la`, `dune-xt-grid`, `dune-xt-functions`
- re-usable collections of utilities that solve everyday problems
- dual-licensed as BSD-2 and GPL 2+ w/ runtime exception (the core modules' license)
- read more: `https://arxiv.org/abs/1602.08991`
- these slides: `https://goo.gl/0dbhp6` `https://dune-community.github.io/files/16_pdesoft_presentation.pdf`

Where to get it

- Hosted on Github `https://github.com/dune-community`

Where to get it

- Hosted on Github `https://github.com/dune-community`
- Overview page `https://dune-community.github.io/`

Where to get it

- Hosted on Github `https://github.com/dune-community`
- Overview page `https://dune-community.github.io/`
- Automatically generated documentation for all branches: `https://dune-community.github.io/docs/index.html`

Where to get it

- Hosted on Github `https://github.com/dune-community`
- Overview page `https://dune-community.github.io/`
- Automatically generated documentation for all branches: `https://dune-community.github.io/docs/index.html`
- ```
  git clone https://github.com/dune-community/dune-xt-super
  cd dune-xt-super
  git submodule update --init --recursive
  ./setup.sh config.opts/gcc
  ```
  ```

Testing

- dune-testtools for buildsystem level type parametrisations

Testing

- dune-testtools for buildsystem level type parametrisations
- googletest for actual unit test, reporting

Testing

- dune-testtools for buildsystem level type parametrisations
- googletest for actual unit test, reporting
- travis.ci: per module, matrix of compilers and module configurations, autodeploys testlogs to git repos, deploys documentation

dune-xt-common

Timings

Coarse grained code profiling

- Automatic + manual stop/start
- Measures usr/wall/sys using `boost::cpu_timer`
- Csv output
- Analysis + visualization scripts[1] with pandas + matplotlib

[1] `https://goo.gl/ZUiN9F`

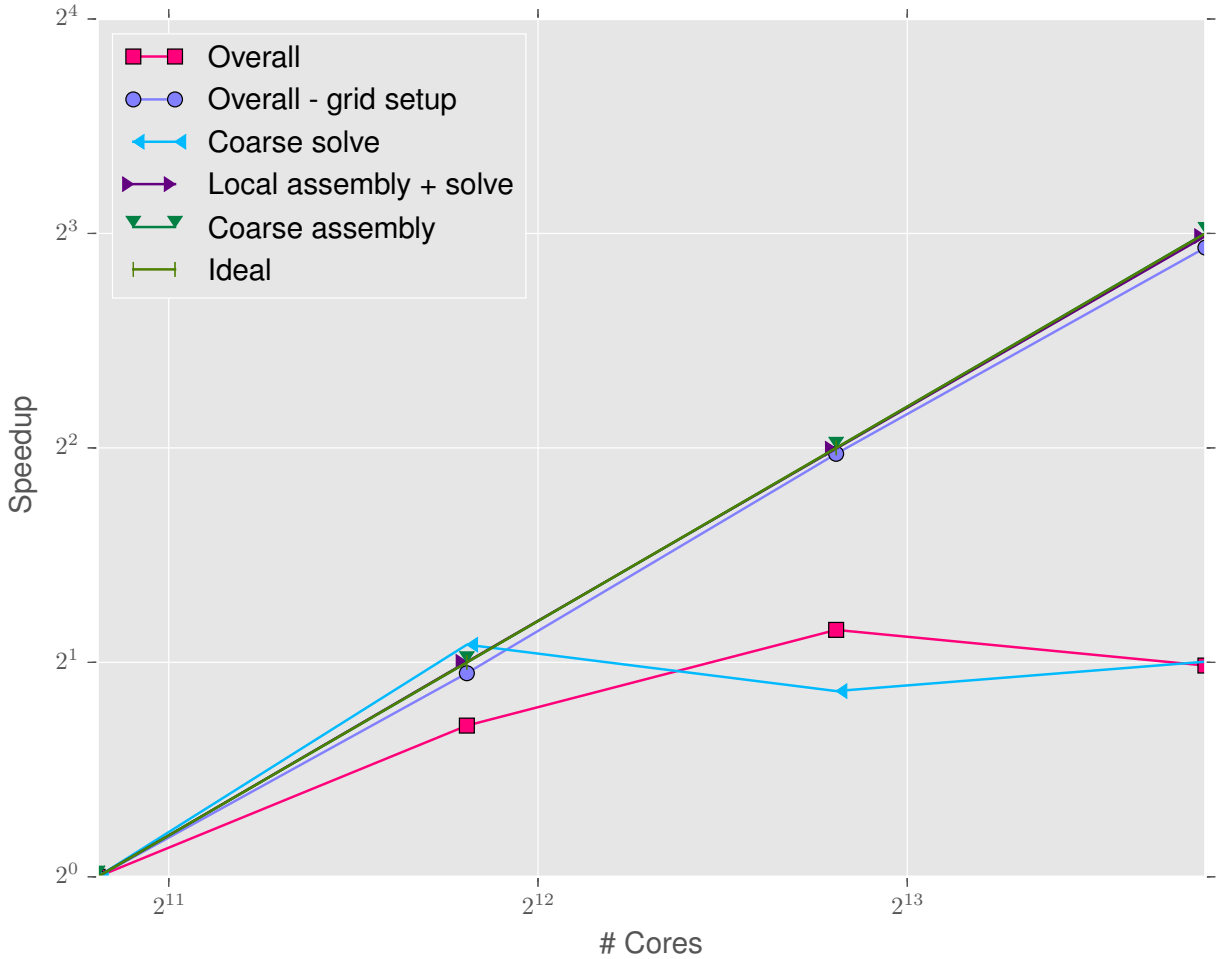
Timings (Code)

```
using namespace Dune::XT::Common;
timings().start("sec");
for (auto i : value_range(5)) {
  ScopedTiming scoped_timing("sec.inner");
  busywait(i * 200);
}
timings().stop("sec");
busywait(1000);
auto file = make_ofstream("example.csv");
timings().output_all_measures(*file);
```

Timings (CSV)

```
threads,ranks,sec_avg_usr,sec_max_usr,sec_avg_wall,sec_max_wall, sec_avg_
8,1,2050,2050,2050,2050,0,0,2050,2050,2050,2050,0,0
```

Timings (Pics)



Containers + Strings

VectorAbstraction, MatrixAbstraction

Non-intrusive, user extensible, facility that allows writing generic code for creation, access and string conversion of `FieldVector`, `std::vector`, ...

Containers + Strings (Code)

```
auto fmat = from_string<FieldMatrix<double, 2, 2>>("[1. 2.; 3. 4.]");
auto dmat = from_string<DynamicMatrix<double>>("[1. 2.; 3. 4.]");

std::vector<double> svector ({1., 1.});
Dune::DynamicVector<double> dvector (2, 1.);
Dune::XT::Common::FloatCmp::eq(svector, dvector);
```

Configuration

ParameterTree Extension

- Type-safe matrix/vector extraction
- Value validation (predifined, custom)

Configuration (Code)

```
typedef FieldVector<double, 2> Vector;
Configuration cfg("cells_per_dim", "[2 2]");
const auto validator = ValidateLess<Vector>({1,1});
cfg.get<Vector>("cells_per_dim", validator);
  struct CellLimit : public ValidatorInterface<Vector, CellLimit> {
    bool operator()(const Vector& vec) const {
      const auto count = std::accumulate(vec.begin(), vec.end(), 1u,
            std::multiplies<std::size_t>());
      return count < 16;
    }
  };
cfg.get<Vector>("cells_per_dim", CellLimit());
```

Other

- Logging
- Extended FloatCompare
- RNGs for vectors, strings
- Range-based-for helpers
- ...

dune-xt-grid

Boundary Handling

We needed a user-extensible differentiation of boundary faces into arbitrary many categories. Current implementation uses Types derived from a `BoundaryType` base class and string identifiers. Might not be optimal, but is currently good enough.

Boundary Handling (Code I)

```
class BoundaryType {
protected:
  virtual std::string id() const = 0;

public:
  virtual bool operator==(const BoundaryType& other) const {
    return id() == other.id();
  }
};
class NeumannBoundary : public BoundaryType { /*...*/};

if (boundary_info.type(intersection) == DirichletBoundary()) {
  // ...
} else if (boundary_info.type(intersection) == NeumannBoundary()) {
  // ....
```

Boundary Handling (Code II)

```
template <class IntersectionType>
class BoundaryInfo {
public:
  virtual BoundaryType& type(const IntersectionType& intersection) const
};
Configuration config;
config["type"]      = "xt.grid.boundaryinfo.normalbased";
config["default"]   = "dirichlet";
config["neumann.0"] = "[ 1. 0.]";
config["neumann.1"] = "[-1. 0.]";

typedef typename Dune::YaspGrid<2>::LeafIntersection YI;
typedef typename Dune::SGrid<2, 2>::LeafIntersection SI;

auto boundary_info_y = BoundaryInfoFactory<YI>::create(config);
auto boundary_info_s = BoundaryInfoFactory<SI>::create(config);
```

GridWalker

- Generic facility to stack *N* element-local operations on a `GridView` or `GridPart` as opposed to traversing it *N* times.
- Supports filtering
- Works with special interface based functors for Codim 0/1 or a plain `std::function< void(const EntityType &)>`

GridWalker (Code)

```
Walker<GV> walker(grid_view);
walker.add(coupling_operator,
    new ApplyOn::InnerIntersections<GV>());
walker.add(boundary_operator,
    new ApplyOn::DirichletIntersections<GV>(boundary_info));
double max_vol = 0;
walker.add([&max_vol](const EntityType& e){
    max_vol = std::max(max_vol, e.geometry.volume();});
// add more, if required...
walker.walk()
```

Other

- GridPovider/Layer: generic way to handle GridViews/GridParts
- PeriodicGridView
- Grid Statistics
- (2d) Grid to Latex output
- EntityInlevelSearch
- VTK based grid visualizations for MPI partitioning, cell volume, ...

dune-xt-la

Generic linear algebra containers

LA::MatrixInterface + LA::VectorInterface

- allows writing generic code using matrices and vectors from ISTL, Eigen, ...
- Mixture of dynamic + CRTP interface
- implements COW + move semantics

Generic linear algebra containers (Code)

```
#include <dune/xt/la/container/container-interface.hh>

using namespace Dune::XT::LA;

template <class C>
    typename std::enable_if<is_container<C>::value, C>::type
assemble_lincomb(const std::vector<C>& components,
                 const std::vector<double>& coefficients)
{
  auto result = components[0].copy();
  result *= coefficients[0];
  for (size_t qq = 1; qq < components.size(); ++qq)
    result.axpy(coefficients[qq], components[qq]);
  return result;
}
```

Runtime selectable solvers

LA::Solver

- Common interface for containers using the VectorInterface/MatrixInterface
- select at runtime, from a Configuration, which type of solver to use
- allows introspection of available types
- no more recompiling to switch a linear solver type

Solvers (Code)

```
XT::LA::Solver<LocalMatrixType> solver(system_matrix);

solver.apply(rhs_vector, solution_vector);
solver.apply(rhs_vector, solution_vector, types()[0]);
solver.apply(rhs_vector, solution_vector, options(type));
```

dune-xt-functions

LocalizableFunctionInterface

Concept

- Functions can be evaluated on a an entity, via a LocalFunction
- Functions are composable
- Strong type guarantees provide good compile errors on trying to use/compose functions with incompatible domains/ranges

LocalizableFunctionInterface

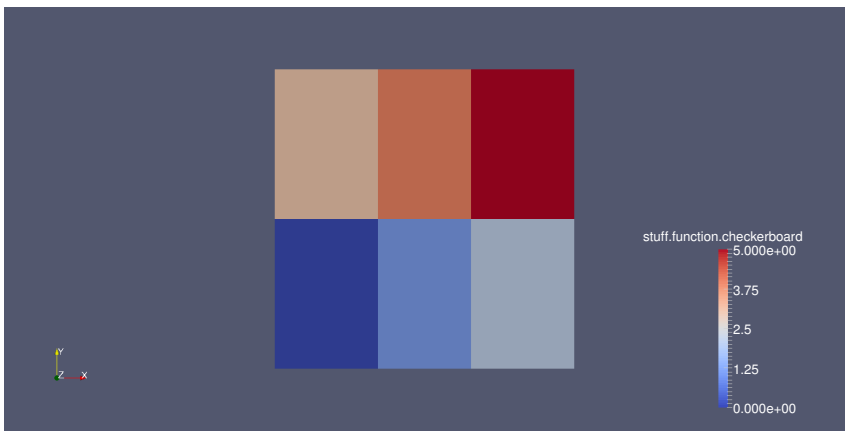
Notable implementations

- CheckerboardFunction: models a piecewise constant function, data setup via `Configuration`
- ExpressionFunction: uses 3rd party string to C++ expression parser
- GlobalLambdaFunction: evaluates a given `std::function< void(const EntityType &)>`

LocalizableFunctionInterface(Code)

```
// with f and g localizable with respect to the same grid_view
(f - g).visualize(grid_view , "difference")
```

```
cfg["type"]         = "stuff.function.checkerboard";
cfg["lower_left"]   = "[0 0]";
cfg["upper_right"]  = "[1 1]";
cfg["num_elements"] = "[3 2]";
cfg["values"]       = "[0 1 2 3 4 5 6]";
```



Thank you for your attention.

Get the code:
`https://github.com/dune-community`
Slides:
`https://goo.gl/0dbhp6`
Paper:
`https://arxiv.org/abs/1602.08991`