The Distributed and Unified Numerics Environment, Version 2.4

Markus Blatt\textsuperscript{1}, Ansgar Burchardt\textsuperscript{2}, Andreas Dedner\textsuperscript{3}, Christian Engwer\textsuperscript{4}, Jorrit Fahlke\textsuperscript{4}, Bernd Flemisch\textsuperscript{5}, Christoph Gersbacher\textsuperscript{6}, Carsten Gräser\textsuperscript{7}, Felix Gruber\textsuperscript{8}, Christoph Grüninger\textsuperscript{9}, Dominic Kempf\textsuperscript{9}, Robert Klöfkorn\textsuperscript{10}, Tobias Malkmus\textsuperscript{6}, Steffen Müthing\textsuperscript{9}, Martin Nolte\textsuperscript{6}, Marian Piatkowski\textsuperscript{9}, and Oliver Sander\textsuperscript{2}

\textsuperscript{1}Dr. Blatt - HPC-Simulation-Software & Services, Heidelberg, Germany
\textsuperscript{2}Institute for Numerical Mathematics, TU Dresden, Germany
\textsuperscript{3}Mathematics Institute, University of Warwick, UK
\textsuperscript{4}Institute for Computational und Applied Mathematics, University of Münster, Germany
\textsuperscript{5}Institute for Modelling Hydraulic and Environmental Systems, Univ. of Stuttgart, Germany
\textsuperscript{6}Department of Applied Mathematics, University of Freiburg, Germany
\textsuperscript{7}Institute of Mathematics, FU Berlin, Germany
\textsuperscript{8}Institute for Geometry and Applied Mathematics, RWTH Aachen, Germany
\textsuperscript{9}Interdisciplinary Center for Scientific Computing, Heidelberg University, Germany
\textsuperscript{10}International Research Institute of Stavanger, Stavanger, Norway

Received: December 17th, 2015; final revision: February 25th, 2016; published: May 10th, 2016.

Abstract: The Dune project has released version 2.4 on September 25, 2015. This paper describes the most significant improvements, interface and other changes for the Dune core modules Dune-Common, Dune-Geometry, Dune-Grid, Dune-ISTL, and Dune-LocalFunctions.

1 Introduction

Dune, the Distributed and Unified Numerics Environment (cf. Bastian et al. [2008a,b], Blatt and Bastian [2007], Bastian and Blatt [2008]), is a modular toolbox for solving partial differential equations (PDEs) with grid-based methods. It supports the easy and flexible implementation of Finite Elements (FE), Finite Volumes (FV), and further discretization methods. Version 2.4 of the Dune core modules Dune-Common, Dune-Geometry, Dune-Grid, Dune-ISTL, and Dune-LocalFunctions was released on September 25, 2015. This paper provides an overview of the most significant improvements and interface changes for each core module.

Version 2.4 of the Dune modules is available in binary form in several major Linux distributions including Debian, Ubuntu, and openSUSE. Source tarballs and anonymous git access can be found on the project homepage www.dune-project.org. The software is available under version 2
of the GNU General Public License, with a special exception for linking and compiling against DUNE. (For details see http://www.dune-project.org/license.)

1.1 Overview

The most notable change concerns the build system. Instead of the GNU Autotools (Calcote [2010]), CMake (Martin and Hoffman [2015]) is now the default build system. Autotools has been deprecated and will be removed in the next DUNE release.

In DUNE-GRID, Release 2.4 introduces interface changes that allow copying entities and intersections. As a result we will remove the class EntityPointer in the near future. Furthermore, YaspGrid has gained many new features. As a consequence we will remove the SGrid grid manager before the next release, as all features of this grid are now covered by YaspGrid. DUNE-GEOMETRY, DUNE-ISTL, and DUNE-LOCALFUNCTIONS only received minor changes. In particular, various deprecated interfaces have been removed.

The DUNE-GRID-DEV-HOWTO module, which was part of the core modules for the 2.3 release, has been abandoned. It was intended as a module to document and teach how to implement DUNE grids. However, the module never saw any work. The IdentityGrid implementation, whose purpose is to serve as a template for new grid implementations, has been moved from DUNE-GRID-DEV-HOWTO to DUNE-GRID, and will continue to be maintained there.

This paper dedicates one chapter each to the changes in each module. More specifically, changes to DUNE-COMMON are discussed in Section 2, DUNE-GEOMETRY in Section 3, DUNE-GRID in Section 4, DUNE-ISTL in Section 5, and DUNE-LOCALFUNCTIONS in Section 6. Finally, we list a few prominent known issues in Section 7.

1.2 System requirements

The minimal required compilers are GCC 4.4 and Clang 3.4. On OS X, GCC 4.7 is required, see Bugtracker issue 1590 for the details. We try to stay compatible with ICC 15 and newer. ICC 14.0.3 works, but needs patches to system headers.

For the new CMake build system, we require CMake 2.8.6 or newer. To get support for the macro dune_enable_all_packages we require at least CMake 2.8.12. The minimal version requirements for the GNU Autotools build system have not changed.

2 dune-common

The major new feature in DUNE-COMMON is the introduction of a new CMake-based build system. But there are also a few other noteworthy improvements.

2.1 CMake as the default build system

The default build system for DUNE has been switched from the GNU Autotools to CMake which is the new default build system and GNU Autotools are still supported until the next DUNE core modules release. Because we provide the dunecontrol script to build several interdependent DUNE modules at once, most differences in the build system are hidden from the user. In particular, option files for dunecontrol can continue to be written as if the Autotools build system was used. However, it is also possible to directly set CMake variables in option files. For example, the line

```
CMAKE_FLAGS="-DCMAKE_CXX_FLAGS=' -g -Wall' 
```
sets the default compiler options. Setting CMake variables directly is strongly recommended, as
the Autotools build system is scheduled for removal before the next release to avoid the overhead
of maintaining two different build systems. For DUNE 2.4 passing the option --no-cmake to
dunecontrol enable Autotools.

As a result of using CMake, out-of-source builds are now the default. Besides GNU make, other
make tools like Ninja-build or Visual Studio’s MSBuild can be used. It is possible to generate
project files for various IDEs like Xcode, Visual Studio or Eclipse.

The Autotools build system provided a special target, make headercheck, that checked whether
each file contained all required #include directives. With CMake, this target is disabled by
default, as it creates a large number of files in the build directory. It can be re-enabled by adding
-DENABLE_HEADERCHECK=1 to the CMake flags. The headercheck target can only be run in the
project’s root build directory, no longer in sub-directories. Calling headercheck for single headers
is no longer supported.

New DUNE projects can be set up using the duneproject script. The script now adds CMake files
so that any new project can be directly used with the new build system. CMake support can
be added to existing modules with the help of a script provided in the DUNE-COMMON module.
Calling dune-common/bin/am2cmake.py -d <module-source-dir> will lead to the creation of
CMakeLists.txt from existing Makefile.am files, but note that all existing CMakeLists.txt within
the module will be overridden. For some modules this will be all it takes. For others some manual
adaptation of the CMakeLists.txt will be required.

2.2 MPI is enabled by default

Previous releases of DUNE would not enable MPI by default even if an MPI implementation was
found by the build system. This has changed with DUNE-COMMON 2.4. Now, MPI support is
enabled unless deliberately switched off by passing -DCMAKE_DISABLE_FIND_PACKAGE_MPI=TRUE
for CMake and --disable-parallel for Autotools. This change implies that even sequential
programs should now start with

```cpp
#include <dune/common/parallel/mpihelper.hh>

int main(int argc, char** argv)
{
    // this statement needs to be first, because MPI_Init is called
    Dune::MPIHelper::instance(argc, argv);
    // main program follows
    return 0;
}
```

because otherwise such programs will abort at run-time if MPI is present but not explicitly
disabled.

Also related to MPI, support for very old MPI implementations, specifically implementations
without support for the MPI-2.1 standard (Gropp et al. [1999]), has been removed. Support for
such installations was deprecated in DUNE 2.3. This change should affect very few users because
all major MPI implementations have been compatible with that standard since at least 2009.

2.3 Bash completion

DUNE-COMMON now provides a simple implementation of bash-completion for the dunecontrol
command. That means that if a DUNE-COMMON module is globally installed, the bash shell will
auto-complete dunecontrol commands and options in the usual way when double-pressing the
tab key. This improves dunecontrol usability greatly.
2.4 Miscellaneous improvements and cleanup

Further changes to DUNE-COMMON include a few minor features, and the deprecation or removal of several obsolete interfaces.

- The container classes FieldMatrix, FieldVector, and DiagonalMatrix can now be constructed from C++11 initializer lists.
- The class CollectiveCommunication is now default-constructible.
- The macros HAVE_DUNE_HASH, HAVE_INTEGRAL_CONSTANT, HAVE_RVALUE_REFERENCES, HAVE_STD_HASH, HAVE_TYPE_TRAITS, HAVE_VARIADIC_CONSTRUCTOR_SFINAE, and HAVE_VARIADIC_TEMPLATES, defined in config.h, are deprecated and will be removed after DUNE 2.4. The same holds for various HAVE_TR1_* macros. The corresponding features are provided by all supported compilers.
- The class SelectType (from the file dune/common/typetraits.hh), deprecated in DUNE 2.3, has been removed. Use std::conditional from the standard library instead.
- The deprecated file misc.hh has been removed. Most math-related functionality has moved to math.hh. The two methods hasPrefix and hasSuffix are now in the new header stringutility.hh. The methods SQR and genFilename disappear with no replacement.

3 dune-geometry

In the DUNE-GEOMETRY module, the quadrature rules and reference elements have been improved.

3.1 Quadrature rules

The code for quadrature rules was improved. First of all, the class QuadratureRules, which implements a cache for quadrature rules, is now thread-safe. One can therefore request rules by writing, e.g.,

```cpp
const auto& quad = Dune::QuadratureRules<double, dim>::rule(element.type(), quadOrder);
```

concurrently from more than one thread. The implementation uses std::thread, and therefore compilers are needed that support std::thread. This includes all compilers listed in Section 1.2, with the notable exception of GCC before version 4.7 on OSX.

Second, the programmer interface for quadrature rules has been cleaned up. The enumeration values Gauss, Jacobian_1_0, and Jacobian_2_0 for the different types of Gaûß rules have been replaced by GaussLegendre, GaussJacobi_1_0, and GaussJacobi_2_0, respectively. The new names were chosen because they better match the names of the corresponding rules used in the mathematical literature.

Finally, the class QuadraturePoint exported both the dimension of the domain of integration and the type used for coordinates twice. The values d and CoordType have been removed. The dimension and Field parameters should be used, respectively, instead.
3.2 Cleanup of the ReferenceElement implementation

- The methods global, mapping, volumeOuterNormal, and initializeTopology have been removed from the ReferenceElement class.

- The methods ReferenceElement::global have been removed; use ReferenceElement::geometry<codim>(i).global instead. The non-interface method ReferenceElement::mapping, which is superseded by the geometry method, has been deprecated, as well.

- The GenericReferenceElement* classes were renamed to ReferenceElement* in Dune 2.3. The old, deprecated names have been removed.

3.3 Miscellaneous improvements and cleanup

- There are new types Codim<cd> and Dim<d> that can be used to encapsulate a (co)dimension. They inherit from std::integral_constant<int> and are useful when an interface should accept both a dimension and a codimension.

- The class MockGeometry, deprecated in Dune 2.3, has been removed. In most cases, replacing it by MultiLinearGeometry and updating the includes is sufficient.

- The header genericgeometry/geometry.hh and the classes Geometry, GenericGeometry, and LocalGeometry contained therein, which were all deprecated in Dune 2.3, have been removed.

4 dune-grid

The major changes in Dune-Grid are the deprecation of the EntityPointer class along with the corresponding transfer of some EntityPointer functionality to the Entity class. One grid implementation has been added and two others have been deprecated. Loops over entities and intersections become much easier to write and read using the C++11 range-based for mechanism.

4.1 Changes to the set of grid implementations

There is one new grid implementation in the Dune-Grid module, and two old ones are being deprecated. The SGrid structured grid implementation has been deprecated, because the competing YaspGrid implementation has been improved so much that SGrid is now obsolete (see Section 4.5). By request of the ALUGrid maintainers, the ALUGrid implementation of an unstructured parallel grid will now be provided as a separate Dune module Dune-ALUGrid outside of the set of Dune core modules (Alkämper et al. [2016], https://gitlab.dune-project.org/extensions/dune-alugrid). The ALUGrid bindings in Dune-Grid itself are deprecated. Support for the external Alberta grid library version 2 and older has been abandoned without deprecation. Alberta 3 is now required. The minimum required version of UG is now UG-3.11.0. Note that building UG from source with Clang requires a bugfix provided by UG-3.11.1.

Finally, the IdentityGrid implementation has been moved from the abandoned Dune-Grid-Dev-Howto module to Dune-Grid itself. IdentityGrid is a meta grid that simply forwards everything to its host grid. Its main purpose is to serve as a template for the development of new grid implementations.
4.2 Range-based for loops

With recent enough compilers, loops over entities, intersections, and other algorithms can now be written using the new C++11 range-based for formalism. This leads to remarkable improvements in code readability. For example, in DUNE-Grid 2.3, a loop over all elements had to be written using iterators

```cpp
GridView::Codim<0>::Iterator it = gridView.begin<0>();
GridView::Codim<0>::Iterator endIt = gridView.end<0>();
for (; it != endIt; ++it)
{
    // Do something with the entity in "it"
}
```

In DUNE-Grid 2.4, this simplifies to

```cpp
for (const auto& element : elements(gridView))
{
    // Do something with the entity in 'element'
}
```

Note how the clumsy type specifications of the old version have disappeared, and how it is obvious even to the untrained eye that this is a loop over the grid elements. Code using the new syntax runs at the same speed as code using iterator loops.

Similarly, a loop over the vertices of a grid view is written as

```cpp
for (const auto& vertex : vertices(gridView))
{
    // Do something with the entity in 'element'
}
```

Finally, if `element` is a codimension-0 entity from the grid view in `gridView`, then

```cpp
for (const auto& intersection : intersections(gridView, element))
{
    // Do something with the intersection in 'intersection'
}
```

is a loop over all intersections of this entity. Note again how much simpler this is compared to the iterator loops used previously.

4.3 EntityPointer is deprecated, entities and intersections become copyable

Previous versions of the DUNE grid interface have included the EntityPointer class, which was intended as a way to store references to grid entities. The grid entities themselves were seen as mere views of actual objects, and could not be stored as separate objects. More specifically, the Entity interface class did not allow copying of Entity objects. Based on similar reasoning, the copying of Intersection objects was prohibited.

However, the distinction between EntityPointer and Entity became increasingly blurred in the grid interface. Some methods would require Entity arguments, others required EntityPointer arguments, with no real reason to prefer one over the other. In an effort to clean up the grid interface, it has therefore been decided to deprecate the EntityPointer interface class completely,
and to get rid of it eventually. To preserve the overall functionality of the grid interface, the Entity class gains additional features; in particular, starting with DUNE-Grid 2.4, Entity objects can be copied, and two entities can be checked for equality using `operator==`. Additionally, to allow containers of Entity objects, such objects become default-constructible.

All interface methods that previously returned an EntityPointer now return an Entity instead. This change applies to the following methods:

```cpp
// C++ code
1 Entity::father() // for entities of codimension 0.
2 Entity::subEntity<codim>() // for entities of codimension 0.
3 Intersection::inside()
4 Intersection::outside()
5 // This method has been deprecated, please use the new method
6 // Grid::entity(const EntitySeed&) instead, which returns an Entity
7 Grid::entityPointer(const EntitySeed&).
```

Smoothly transitioning to the new Entity implementation has been a challenge. For the 2.4 release, Entity and EntityPointer have temporarily gained additional features that should make Entity objects look like EntityPointer objects and vice versa. Entity and EntityPointer have a certain amount of interoperability code to ease the transition. In particular, an Entity can be dereference with `*entity` and call member methods with `entity->foo()`, so that code works with both grids that have been ported to the new interfaces as well as unported grids. All of the compatibility methods do however raise deprecation warnings. The following code sketches the additional methods of the Entity interface class.

```cpp
// C++ code
1 class Entity
2 {
3 ...
4 public:
5  //!! default constructor to allow for creation of empty entities
6  Entity () {}
7  
8  //!! convenience operators to make entity behave like entity pointer
9  const Entity& operator* () const { return *this; }  
10  Entity& operator* () { return *this; }
11  
12  //!! public assignment operator to allow for copying
13  Entity& operator= (const Entity& rhs);
14  
15  //!! for entities of codimension 0.
16  //!! father and subEntity return Entity objects instead of EntityPointer
17  Entity father() const;
18  Entity subEntity<codim>() const;
19  }
```

Dereferencing an EntityPointer now returns an Entity by value, rather than by reference.

```cpp
// C++ code
1 class EntityPointer
2 {
3 ...
4 public:
5  //!! Returns entity object
6  Entity operator*() const;
7  }
```

All DUNE iterators now return objects when dereferenced.
C++ code

```cpp
class EntityIterator {
...
public:
  // Dereferencing operator returning Entity object
  Entity operator* () const;
  // deprecated: the access to pointers is deprecated and will be removed
  proxy<Entity> operator-> () const;
};
```

EntitySeed objects are converted to Entity rather than to EntityPointer. On grids the method `entityPointer(const EntitySeed&)` has been deprecated. Use the new method `entity(const EntitySeed&)` instead, which returns an Entity.

C++ code

```cpp
class Grid {
...
public:
  // the method entity replaces the method entityPointer( seed )
  Entity entity( const EntitySeed& seed ) const;
};
```

Consequences of this change are:

- Iterators are now allowed to return temporary Entity or Intersection objects instead of references. Code that captures the result in a const reference will still work in both cases, because the lifetime of the temporary is bound to this reference. However: code that forwards such references may fail due to dangling references. In order to avoid this the correct return type can be forwarded using `decltype(*it)`. This means for example, changing

```cpp
const Entity& foo(const Iterator& it) { return *it;}
```

to

```cpp
auto foo(const Iterator& it) -> decltype(*it) { return *it;}
```

- Keep in mind that Entity and Intersection objects can be large. If a list of entities is needed `EntitySeed` should be used instead.

- Meta grids using the new interface (like `GeometryGrid`) do not work with host grids that still use the old interface.

All grid implementations in DUNE-GRID have been ported to the new interface, except for the deprecated bindings for ALUGrid 1.52. For the usage of ALUGrid one should switch to the new DUNE-ALUGRID module.

### 4.4 Speed increase in meta grids

As a consequence of the changes to the Entity and Intersection interface classes, meta grids can now be implemented with much less overhead than before.

In a meta grid, instances of DUNE grids are stacked on top of each other to increase the feature set in a modular way. Previously, to implement a meta grid entity, an entity pointer of the host grid had to be stored in the meta grid entity to ensure the validity of the host entity during the lifetime.
of the meta grid entity. Since until recently entities were not default constructible, a pointer to and EntityPointer had to be stored which led to an increase of 8 bytes per meta grid layer in the entity and entity pointer meta implementations. In Figure 1 we present a comparison of SPGrid and layers of IdGrid, a reimplementation of IdentityGrid, which simply forwards all method calls to the host grid. We can see that the memory footprint of the EntityPointer increases by 8 bytes for each meta grid layer while the memory of the meta entity is only 8 bytes for all layers. As Entity objects can now be copied, storing the additional EntityPointer object is no longer necessary. The results are impressive. When repeating the same experiment with the DUNE 2.4 release, we see that the memory consumption for each extra grid layer does not increase anymore. As a result of the reduced memory footprint (a layer of IdGrid does not add memory overhead) we see that the performance overhead the interface introduced by another layer of IdGrid becomes almost negligible, i.e. stays around 100% which corresponds to the run time of the program without any IdGrid layers (variations due to compiler effects do occur, though).

Figure 1: Comparison of the previous DUNE interface (2.3, return as reference) with the new interface (2.4, return as objects) that allows for a copyable class Entity. Left, the change in the memory footprint of the classes EntityPointer and Entity. Right, the comparison of run times for the Finite Volume test case from Klöfkorn and Nolte [2012]. Note that for the 2.4 release the overhead of the interface introduced by another layer of IdGrid becomes almost negligible, i.e. stays around 100% which corresponds to the run time of the program without any IdGrid layers (variations due to compiler effects do occur, though).

4.5 YaspGrid

YaspGrid is a standalone implementation of the DUNE grid interface providing a parallel Cartesian grid. In this release the YaspGrid grid manager has received an important overhaul. YaspGrid objects can now be used for grids of any dimension. More importantly, YaspGrid now implements entities of all codimensions and is able to communicate on these entities. This allows implementation of higher-order methods on distributed YaspGrid objects. Before DUNE-GRID 2.4, YaspGrid could only manage grids that were axis-aligned, with cube elements of a single size, and the lower-left corner of the grid bounding box at the origin. With DUNE 2.4, YaspGrid becomes more flexible. To maintain runtime efficiency, this extra flexibility is controlled by a new template parameter of the YaspGrid class. This second parameter is a policy class that specifies how coordinates are stored. Possible choices to create a tensor product grid are:

- Dune::EquidistantCoordinates<ctype, dim>, where dim is the grid dimension and ctype is the type used internally for coordinates. This is the default (with ctype being double),
Figure 2: Example of a 2D tensor product grid

as it recreates the previous behavior. The fact that the coordinate type \( \texttt{ctype} \) is changeable through the template parameter is a new feature in itself.

- **Dune::EquidistantOffsetCoordinates<\texttt{ctype}, dim>** with \( \texttt{ctype} \) and \( \text{dim} \) as above. This implements domains \((a_i,b_i)^d\) with arbitrary \( a_i \), where YaspGrid was previously limited to \( a_i = 0, i = 1, \ldots, d \). There is a small but measurable performance penalty.

- **Dune::TensorProductCoordinates<\texttt{ctype}, dim>** with \( \texttt{ctype} \) and \( \text{dim} \) as above. This choice allows one to define a tensor product grid, i.e., a semi-structured grid defined by a monotone coordinate sequence per direction. This allows the definition of locally refined coarse grids while maintaining the computational efficiency of a structured grid implementation. See Figure 2 for an example of a two dimensional tensor product grid.

The list of constructors for the YaspGrid class has been rewritten. There is one dedicated constructor for each policy class listed above.

- For equidistant tensor product YaspGrids:

  ```cpp
  YaspGrid(Dune::FieldVector<\texttt{ctype}, \text{dim}> L,
           \text{std::array<int, \text{dim}}> s,
           \text{std::bitset<\text{dim}> periodic = std::bitset<\text{dim}>()},
           \text{int overlap = 1},
           \text{CollectiveCommunicationType comm = CollectiveCommunicationType()},
           \text{const YLoadBalance<\text{dim}>* lb = defaultLoadbalancer()})
  ```

- For an equidistant tensor product grid with arbitrary lower left corner:

  ```cpp
  YaspGrid(Dune::FieldVector<\texttt{ctype}, \text{dim}> lowerleft,
            \text{Dune::FieldVector<\texttt{ctype}, \text{dim}> upperright},
            \text{std::array<int, \text{dim}}> s,
            \text{std::bitset<\text{dim}> periodic = std::bitset<\text{dim}>()},
            \text{int overlap = 1},
            \text{CollectiveCommunicationType comm = CollectiveCommunicationType()},
            \text{const YLoadBalance<\text{dim}>* lb = defaultLoadbalancer()})
  ```

- For an arbitrary tensor product grid:

  ```cpp
  YaspGrid(\text{std::array<\text{std::vector<\texttt{ctype}}> dim} coords,
            \text{std::bitset<\text{dim}> periodic = std::bitset<\text{dim}>()},
            \text{int overlap = 1},
            \text{CollectiveCommunicationType comm = CollectiveCommunicationType()},
            \text{const YLoadBalance<\text{dim}>* lb = defaultLoadbalancer()})
  ```
Some of the other constructors are now deprecated, and will eventually be removed. Using the wrong combination of second template parameter and constructor will result in a failed static assertion.

Note that YaspGrid now uses CollectiveCommunication. As these classes have just been made default-constructible, the communicator parameter can usually be omitted on the constructor call. This also implies that if MPI has been found by the build system, a parallel YaspGrid will be constructed.

All partitioning code is now located in dune/grid/yaspgrid/partitioning.hh. Note that previously implemented own partitioning methods need to be update to accommodate the use of std::array<int, dim> instead of Dune::FieldVector<int, dim>. YaspFixedSizePartitioner is a new implementation of a partitioner with a predefined number of processors per direction.

A specialization of Dune::BackupRestoreFacility for YaspGrid is now provided in the header dune/grid/yaspgrid/backuprestore.hh. Equidistant grids write a single file for all processes, tensor product grids write one file per processor, that contains only the coordinate range relevant to that processor. The output format is a human-readable ASCII format.

4.6 UGGrid

As the most prominent improvement, UGGrid now allows load balancing with the element partitioning being provided by third-party software. There is no restriction on what software to use here. All that is required is to set up a std::vector which for each element contains the rank where this element is supposed to be sent. Everything else happens automatically.

As a first way to use this interface, a class ParMetisGridPartitioner has been added that provides the required repartitioning information from the ParMetis partitioner, Karypis and Kumar [1998b]. Hence, partitioning an existing UGGrid object using ParMetis (Schloegel et al. [2002]) requires only the following few lines of code

```
C++ code
1 // Create initial partitioning using ParMETIS
2 std::vector<unsigned> part(ParMetisGridPartitioner<GridView>::partition(gridView, mpihelper));
3
4 // Transfer partitioning from ParMETIS to our grid
5 grid->loadBalance(part, 0);
```

Note how the information transfer goes through a std::vector, which is neither specific to UGGrid nor to ParMetis. It is therefore easy to implement the mechanism for other grids or partitioners. Note however, that this feature is experimental, and the interface may change again without much prior notice.

In addition to this, edge and face geometries have been implemented in UGGrid. Therefore, UGGrid now offers geometries for all of its entities. Apart from achieving a more complete feature set, these geometries are required in several applications such as DUNE-PDELab, once degrees of freedom are associated with edge or face entities. The implementation is based on MultiLinearGeometry.

4.7 Miscellaneous improvements and cleanup

- Geometry::jacobianTransposed and Geometry::jacobianInverseTransposed return their results now by value rather than by reference. The result types are guaranteed to be copyable and assignable. Existing code which stored references or pointers to these returned values will likely no longer work.
• The class `Entity<0>` has a new method `subEntities(unsigned int codim)`, which returns the number of subentities of the given codimension. It has the same functionality as the method `count<codim>`, but the `codim` argument of the new method is a normal parameter, rather than a template parameter. Also, we believe that the new name is more appropriate. The `count` method is deprecated.

• The mapper classes `SingleCodimSingleGeometryMapper` and `MultipleCodimMultipleGeometryMapper` now use the number type used by the grid index set to return indices. Previously, `int` was hard-wired.

• All iterators over entities are now forward iterators in the sense of the standard template library (STL). In particular, they can now be default-constructed and postfix-incremented, which wasn’t possible previously. The usual caveat concerning postfix increments applies: postfix incrementing may be noticeably slower than prefix incrementing.

• The various mapper classes in `dune/grid/common` now have methods `index` and `subIndex`, which do the same thing as the `map` methods. With this renaming the `Mapper` interface is more consistent with the conceptually similar `IndexSet` interface. The old `map` methods are still present, but they are marked as deprecated and will be removed after the 2.4 release.

• There is a new method `types` on index sets that returns an iterator range visiting all geometry types of a given codimension contained in the domain of the index set. Its type is implementation defined and exported as typedef `Types`. The method `geomTypes` on index sets is deprecated and will be removed after the 2.4 release. It is replaced by the new `types` method.

• `TensorGridFactory`, a factory class for tensor product grids can be found in `dune/grid/utility/tensorgridfactory.hh`. It is implemented through the `GridFactory` for all unstructured grids and has a specialization for `YaspGrid`. The factory class provides a multitude of methods to fill coordinate ranges. Check the Doxygen documentation for details.

• The class `Geometry` does not export the type `Jacobian` anymore. It is replaced by the type `JacobianInverseTransposed`.

• The class `Entity` no longer exports the type `ctype`. Use the type `Entity::Geometry::ctype` instead.

• The class `Entity` does not export the number `dimensionworld` anymore. Please use `Entity::Geometry::coorddimension` instead.

• The methods `EntityIterator::level` and `EntityPointer::level` have been deprecated. To obtain the level of an entity pointed to by an iterator or `EntityPointer`, please dereference the iterator(pointer) and call the method `level` on the entity directly.

• The class `EntityPointer` does not export the number `codim` anymore. Use `codimension` instead.

• The values `Geometry::dimension` and `Geometry::dimensionworld` are deprecated, and will be removed after the release of DUNE-GRID 2.4.

• The capability class `Dune::Capabilities<GridType>::isParallel` is deprecated because its meaning was never well-defined. It will be removed after the 2.4 release. To suppress the deprecation warning define the macro `DUNE_AVOID_CAPABILITIES_IS_PARALLEL_DEPRECATION_WARNING`.

• The methods `lbegin`, `lend`, `leafbegin`, and `leafend` on grids are deprecated and will be removed after the 2.4 release. Instead, use the methods `begin` and `end` from the grid’s level and leaf grid views.
Comparisons between different types of entity iterators (level vs. leaf) and EntityPointer have been deprecated. Those kinds of comparisons should be replaced by comparisons between the entities pointed at by the iterators, so \texttt{leafIt == levelIt} becomes \texttt{*leafIt == *levelIt}. Iterators of a single type can of course still be compared with each other.

Support for Grape has been deprecated and will be removed after the 2.4 release. Use the DUNE module DUNE-Grape (http://users.dune-project.org/projects/dune-grape) instead.

### 4.8 Changes for maintainers of grid implementations

Most changes described in the previous sections concern all users of the DUNE grid interface. However, there are also a few changes that implementors and maintainers of third-party DUNE grid implementations should be aware of.

- All grid implementations not using the DefaultGridView must rename their implementations from \texttt{leafView/levelView} to \texttt{leafGridView/levelGridView}.

- Grids must implement the new copyable entities and intersections. See Section 4.3 for details.

- Grids are now allowed to return temporary objects from their entity and intersection iterators. This is mostly interesting for meta grid developers, as it allows for a much more straightforward implementation.

### 5 dune-istl

DUNE-ISTL, DUNE’s Iterative Solvers Template Library, received improved support for complex numbers and its algebraic multigrid (AMG) code became more flexible.

- We have fixed several issues when repartitioning matrices. Thus the parallel AMG method is now usable with the ParMETIS (Karypis and Kumar [1998b]) bindings of the free PT-Scotch (Chevalier and Pellegrini [2008]) library.

- When using AMG with the SymmetricDependency the sparsity pattern of the matrix is not assumed to be symmetric any more.

- We have fixed several issues when using our solvers with \texttt{std::complex<double>}. In particular RestartedGMResSolver and MINRES now fully support complex numbers.

#### 5.1 Deprecated and removed features

- The DiagonalMatrix class is now only available at dune/common/diagonalmatrix.hh. The former transition header at dune/istl/diagonalmatrix.hh has been removed.

- The constructors RestartedGMResSolver do not take the argument \texttt{bool recalc_defect} any more. It indicated whether the defect should be recalculated on restart. The old constructors were deprecated.
6 dune-localfunctions

- All LocalFiniteElement classes now have a method size, which returns the number of shape functions of the finite element. This method is for convenience: previously, the localBasis had to be fetched to get the same information.

- The MonomLocalFiniteElement class has been renamed to MonomialLocalFiniteElement. Correspondingly, its header monom.hh has been renamed to monomial.hh. The old class and header are still there for backward-compatibility.

- New Raviart–Thomas elements were added but only for interpolation as the Jacobians are missing. The new elements are Raviart–Thomas 3 and 4 for 2d quadrilaterals. The code for this was contributed by Jizhou Li.

- Second derivatives of the shape functions of the Pk2DLocalFiniteElement are now available. The code for this was contributed by Elisa Friebel.

6.1 Deprecated and removed features

- The class Q2LocalFiniteElement, deprecated in DUNE 2.3 has been removed. Please use the more general QkLocalFiniteElement instead.

- The class RannacherTurek2DLocalFiniteElement<D, R>, deprecated in DUNE 2.3, has been removed. Please use RannacherTurekLocalFiniteElement<D, R, 2> instead.

- Most of the Raviart–Thomas and Brezzi–Douglas–Marini elements not following the new naming scheme have been removed.

7 Known bugs

As is typical for a software of this complexity, DUNE contains a certain number of bugs. We list here a few of the most relevant ones. A lot of them are related to the build system. A complete list of known bugs can be found online in our bug-tracker https://gitlab.dune-project.org/flyspray/FS/issues.

7.1 Known issues regarding the build system

- Building shared libraries with CMake might break if used with external static libraries compiled without support for position independent code (g++ -fpic).

- DUNE-Grid fails to build with GCC 4.4 when configured with UGGrid. This is due to an overload resolution failure in GCC 4.4 that is no longer present in newer compilers. (See FS #1695.)

- Can only be build DUNE with GCC 4.9 without the -pedantic flag because it will otherwise reject some compatibility code needed to support GCC 4.4. (See FS #1634.)

- DUNE does not build on Debian-based systems if all the following conditions are satisfied: The system uses GCC 4.9 or newer as default compiler, GCC 4.8 or below and MPICH are used. The reason is Debian bug #624349 which makes MPI propose flags from the newer default compiler also for older compilers where they are not supported.
7.2 Further known issues

- The pseudo inverse which is used in the implementations of MultiLinearGeometry and AffineGeometry might fail for nearly singular matrices. This is not really a problem unless the grid is nearly degenerate.

- The parallel UGGrid may return wrong ids on very complex adaptively refined and load-balanced grids. The reason is that the Dune grid interface mandates that two entities on different levels have the same id if they are copies. Therefore the UGGrid id/subId methods look through the ancestry of a given entity to see if there are copies. However, UG does so-called vertical load-balancing, which means that the ancestry may be distributed across different processors. Since the current code does not take that into account, wrong ids will be returned in the presence of vertical load-balancing. This is a potentially severe issue, because users do not get error messages, only ids that are tacitly wrong.

- Dune-ISTL does not work with version 5 of METIS (Karypis and Kumar [1998a]), see FS #1212.

8 Recommended optional third party scientific software

Some of Dune’s functionality can only be used with additional third party scientific software. We recommend to install the following external packages for an extended feature set available through the Dune core modules:

- The supported external grid managers: ALBERTA (Schmidt and Siebert [2004]), Dune-ALUGrid (Alkämper et al. [2016]), or UG (Bastian et al. [1997]).

- A (parallel) load balancer for parallel grids and for the parallel algebraic multigrid method: METIS/ParMETIS (Karypis and Kumar [1998a,b]) and/or PT-Scotch (Chevalier and Pellegrini [2008]).

- A viewer for VTK (Schroeder et al. [1996]) for better post processing of the simulation output.

- Gmsh (Geuzaine and Remacle [2009]) for generating meshes read by some of Dune’s grid implementations.

- A direct linear solver for better convergence of the algebraic multigrid method: SuperLU (Li [2005]) and/or UMFPack (Davis [2004]).

- Your favorite optimized implementation of BLAS (Blackford et al. [2002]) and LAPACK (Anderson et al. [1990]).

Further information can be found in the installation instructions available from the Dune homepage www.dune-project.org.

9 How to cite DUNE

If using one of the Dune core modules please cite the appropriate papers from the list of original Dune papers (Bastian et al. [2008a,b], Blatt and Bastian [2007], Bastian and Blatt [2008]) and the current release notes. Please note, that other Dune modules might require citation of further papers, such as Dune-ALUGrid (Alkämper et al. [2016]), Dune-Fem (Dedner et al. [2010]), or Dune-PDELab (Bastian et al. [2010]). Further Dune modules are described in Dedner et al. [2012] and on the Dune web page www.dune-project.org.
10 Acknowledgments

We would further like to thank Martin Alkämper, Timo Betcke, Andreas Buhr, Marco Cecchetti, Elisa Friebel, Stefan Girke, Claus-Justus Heine, Emmanouil Kagiass, Ole Klein, Angela Klevinghaus, Andreas Lauser, Jizhou Li, Arne Morten Kvarving, Andreas Nüssing, Steffen Persvold, Elias Pipping, Uli Sack, Bård Skaflestad, and Jonathan Youett for their contributions to the DUNE project. Robert Klöfkorn acknowledges the National IOR Centre of Norway for financial support.

References


