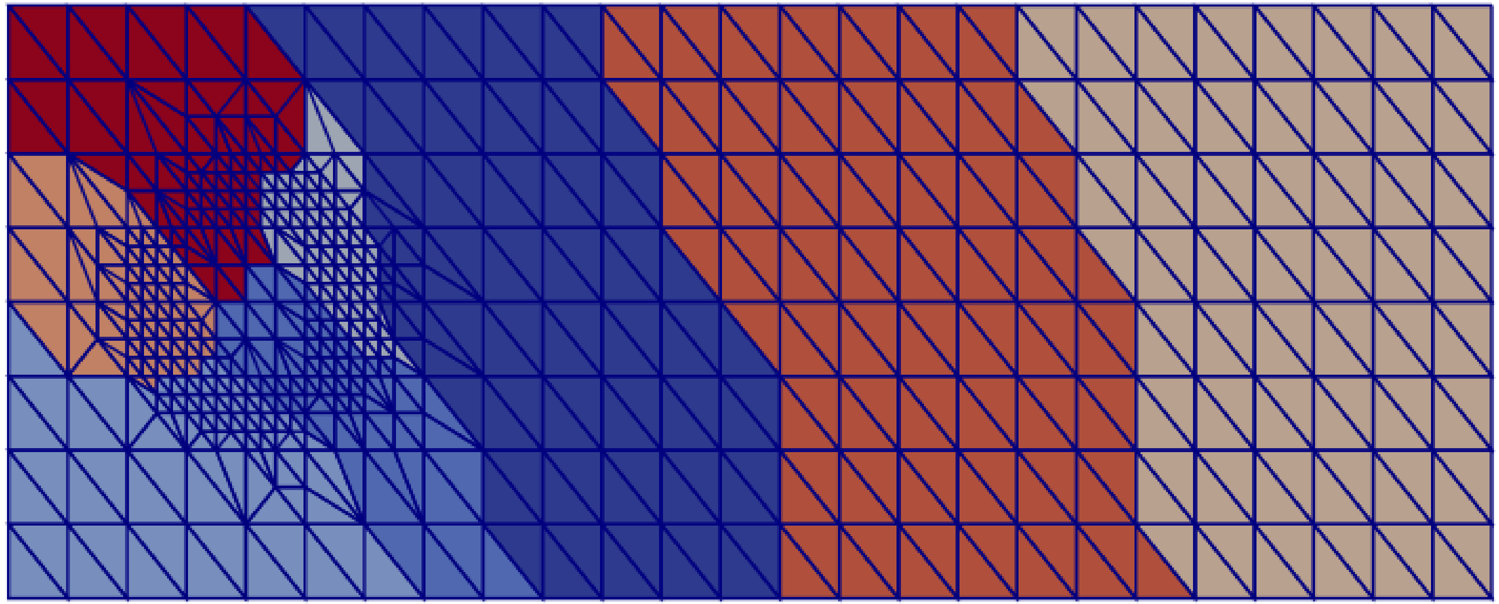
Parallel Grids and Parallel Solvers

Parallel Grids and Parallel Solvers

- AMDiS is by default aware of parallelization
- Parallel grid **and** parallel linear algebra backend required



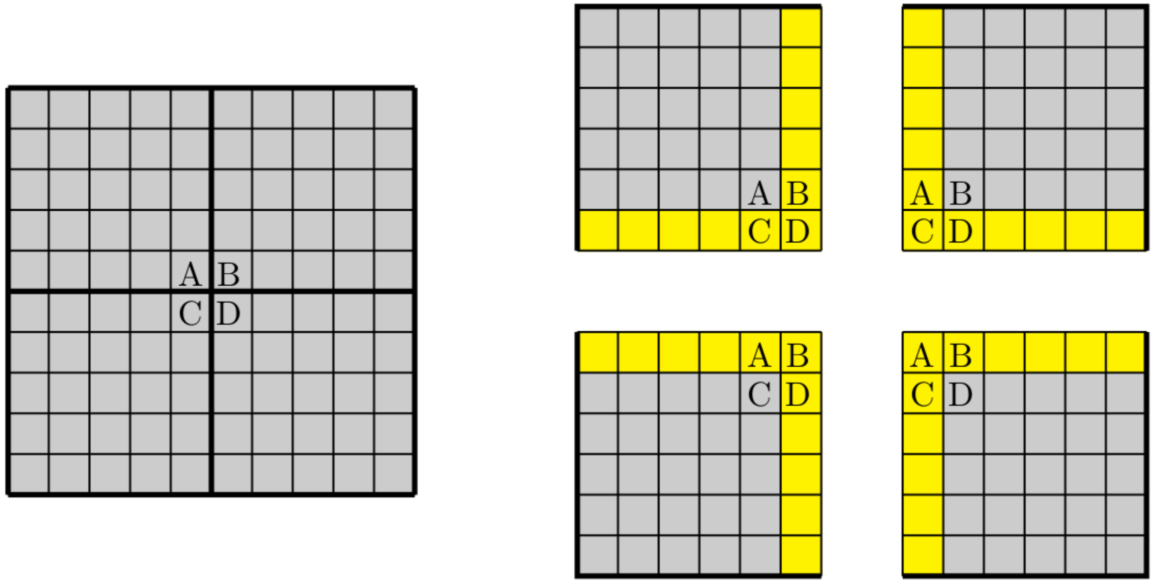
Decomposition of an adaptively refined grid into eight subdomains

(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids

Distributed Grids

- Grids with *ghost* elements

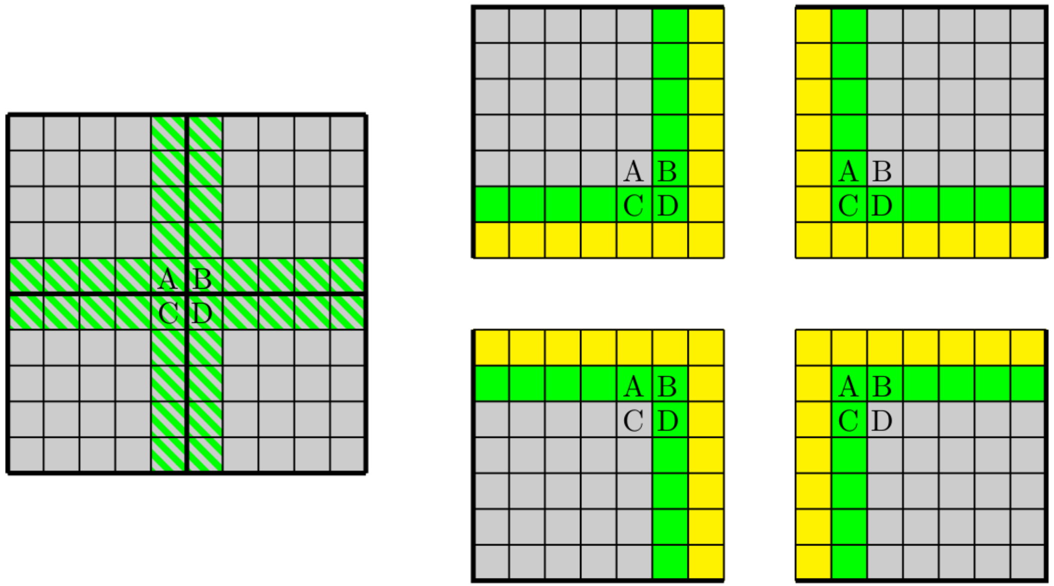


(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids and Parallel Solvers

Distributed Grids

- Grids with *overlapping* elements



(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids

Creating a Parallel Grid

- Some grids can be constructed distributed directly
  - Structured grids (`YaspGrid` or `SPGrid`) with parameters for overlap
  - Unstructured grid `ALUGrid` from special distributed grid files, e.g., with dune-vtk
- Other grids require the initial construction on rank 0 and perform a distribution afterwards, using `grid.loadBalance()`.
  - Example: `UGGrid`
- Some grids are not parallel on its own, e.g. `AlbertaGrid`, `FoamGrid`
  - Require "parallelization wrapper" in form of a Meta-Grid, see dune-metagrid.

Example 1

```
std::bitset<dim> periodic("00");  // Not periodic in either of the two directions
int overlapSize = 1;              // Thickness of the overlap layer
Dune::YaspGrid<2> grid({1.0,1.0}, {10,10}, periodic, overlapSize, MPI_COMM_WORLD)
```

Parallel Grids

Creating a Parallel Grid

- Some grids can be constructed distributed directly
  - Structured grids (`YaspGrid` or `SPGrid`) with parameters for overlap
  - Unstructured grid `ALUGrid` from special distributed grid files, e.g., with dune-vtk
- Other grids require the initial construction on rank 0 and perform a distribution afterwards, using `grid.loadBalance()`.
  - Example: `UGGrid`
- Some grids are not parallel on its own, e.g. `AlbertaGrid`, `FoamGrid`
  - Require "parallelization wrapper" in form of a Meta-Grid, see dune-metagrid.

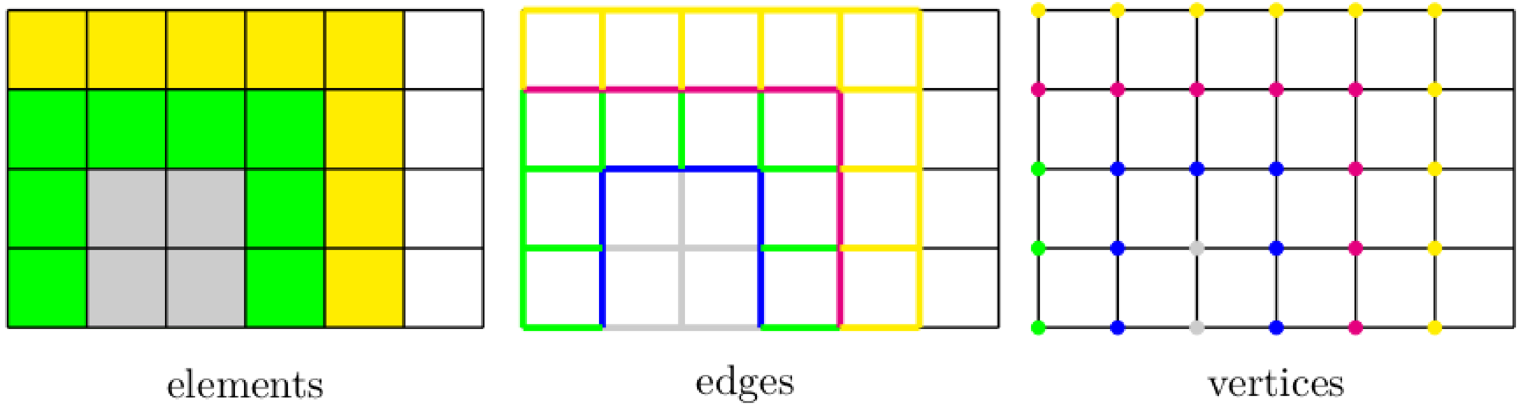
Example 2

```
using Factory = Dune::StructuredGridFactory< Dune::UGGrid<2> >;
auto gridPtr = Factory::createSimplexGrid({0.0,0.0}, {1.0,1.0}, {10u,10u});
gridPtr->loadBalance();
```

Parallel Grids

Partition Type of Entities

- Each entity in a process has a **partition type** assigned to it.
- There are five different possible partition types: `interior` (grey), `border` (blue), `overlap` (green), `front` (magenta), and `ghost` (yellow).



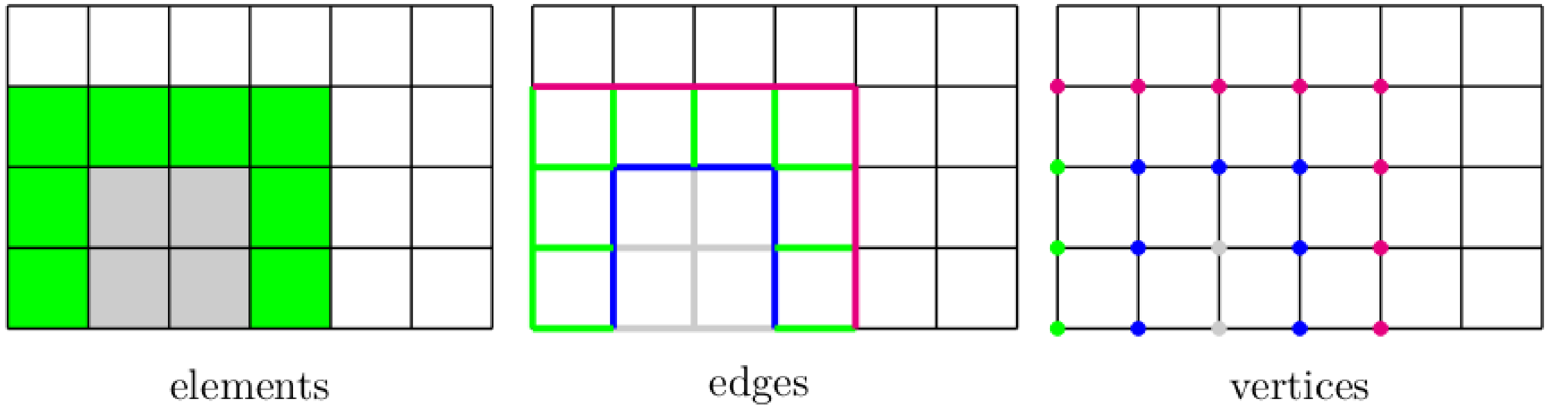
Partition with `interior`, `overlap`, and `ghost` elements

(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids

Partition Type of Entities

- Each entity in a process has a **partition type** assigned to it.
- There are five different possible partition types: `interior` (grey), `border` (blue), `overlap` (green), `front` (magenta), and `ghost` (yellow).



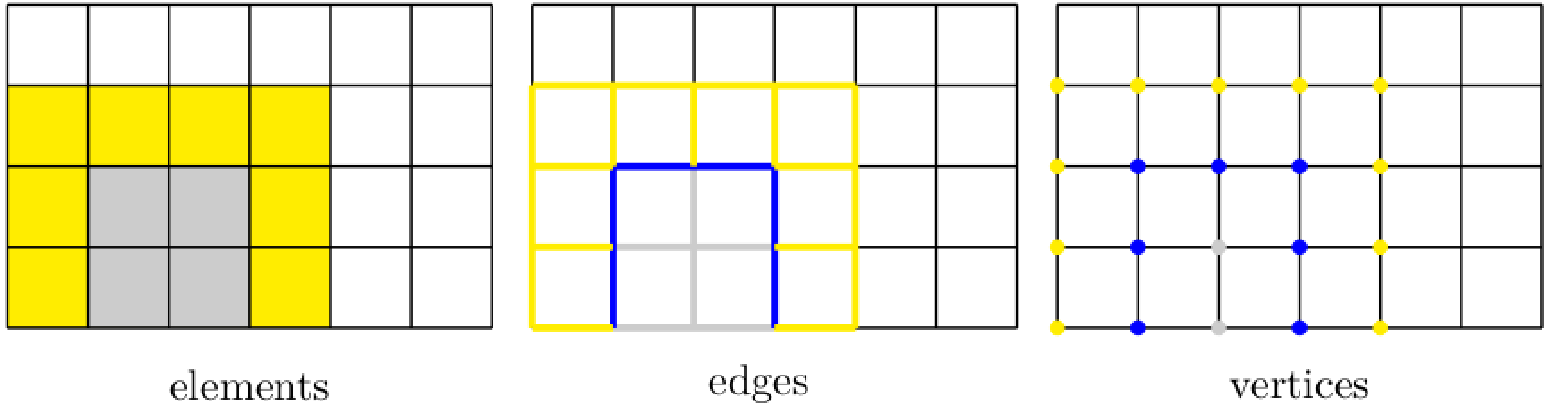
Partition with `interior` and `overlap` elements

(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids

Partition Type of Entities

- Each entity in a process has a **partition type** assigned to it.
- There are five different possible partition types: `interior` (grey), `border` (blue), `overlap` (green), `front` (magenta), and `ghost` (yellow).



Partition with `interior` and `ghost` elements

(4) Image from: O. Sander, *DUNE - The Distributed and Unified Numerics Environment*, 2020

Parallel Grids

Partition Type of Entities

- You can traverse all entities of a specific group of partition types
- Groups are defined as (unions of) sets: e.g. `Partititons::interior`, `Partititons::border`, or the union `Partititons::interior + Partititons::border`, or all (local) entities: `Partititons::all`

```
std::size_t counter = 0;
for (const auto& vertex : vertices(gridView, Partitions::interior + Partitions::border))
{
  if (vertex.partitionType() == InteriorEntity)
    counter++;
}
```

Note: `gridView.indexSet()` enumerates the entities of the `all` partition!

TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Parallel Grids

Grid Partititoning

- Grids provide default partititon, when calling `loadBalance()`
- Sometimes, you want to use a more powerful partitioning algorithms
- Some grids provide extended interface (not unified)

Example: UGGrid

```
bool loadBalance(const std::vector<Rank>& targetProcesses,
                 unsigned int fromLevel);
```

Create initial partitioning using *ParMETIS*

```
#include <dune/grid/utility/parmetisgridpartitioner.hh>
...
std::vector<unsigned int> part
  = ParMetisGridPartitioner<GridView>::partition(gridView, mpiHelper);
gridPtr->loadBalance(part, 0);
```

Parallel Grids

Grid Partititoning

- Grids provide default partititon, when calling `loadBalance()`
- Sometimes, you want to use a more powerful partitioning algorithms
- Some grids provide extended interface (not unified)

Example: UGGrid

```
bool loadBalance(const std::vector<Rank>& targetProcesses,
                 unsigned int fromLevel);
```

Refine partititoning by redistribution:

```
#include <dune/grid/utility/parmetisgridpartitioner.hh>
...
std::vector<unsigned int> part
  = ParMetisGridPartitioner<GridView>::repartition(gridView, mpiHelper, 1000f);
gridPtr->loadBalance(part, 0);
```

Parallel Grids and Parallel Solvers

- Data structures and linear algebra backend must be aware of distributed grids
- Two paradigms:
  1. **Domain decomposition** (local data structures, solvers handle parallelization)
  2. **Distributed data structures** (data structures handle parallelization, "standard" solvers)

In the first category falls dune-istl, in the second category PETSc (and in the future PMTL)

- In AMDiS, domain decomposition requires grid **overlap**:
  `grid.overlapSize() > 0`
- In AMDiS, distributed data structured allows grids with **ghost** cells:
  `grid.ghostSize() >= 0`

Parallel Grids and Parallel Solvers

ISTL Backend

- Create a parallel grid with overlap, e.g. `Dune::YaspGrid`, `Dune::SPGrid`.
- Assembling happens on the `Partition::all` (local) entities.
- Only iterative solvers can be used!
- Local data structures are always up-to-date, since overlapping ensures synchronization

Examples

- Choose parallel preconditioner: ParSSOR (`pssor`), BlockJacobi (`bjacobi`), or AMG

```
prob->solver: pcg
prob->solver->precon: bjacobi
prob->solver->precon->sub precon: ilu   % precon to be applied on subdomains
```

Parallel Grids and Parallel Solvers

PETSc Backend

- Create a parallel grid w/ or w/o overlap/ghost, e.g. `Dune::UGGrid`, `Dune::ALUGrid`,...
- Basis provides `DistributedCommunication` that holds a parallel DOF-map, mapping local to global indices.
- Assembling happens on the `Partition::interior` entities.
- Ghost/Overlapping entities are used to collect data from neighbouring processors during synchronization.
- Classification of each DOF as owner and non-owner DOF on each processor, i.e., a DOF belongs to exactly one process.
- Vectors and Matrices perform communication of data on insertion and before access.
- Data is automatically synchronized. But: Do not mix read and write access to data $\rightarrow$ expensive

Parallel Grids and Parallel Solvers

PETSc Backend

Examples

- PETSc Matrix type: *MATMPIAIJ* (parallel sparse matrix), PETSc Vector type: *VECMPI* (parallel vector with ghost padding)
- Support all solvers and preconditioners for these matrix/vector types

```
prob->solver: cg
prob->solver->pc: bjacobi
prob->solver->pc->sub ksp: preonly    % solver to be applied on subdomains
prob->solver->pc->sub ksp->pc: ilu    % preconditioner to use for ths sub solver
```