Exercises for Tutorial06

Parallelism

This exercise will show you two things:

1. For common problems it is really easy to run your code in parallel. Almost everything is done by PDELab and you only have to make some small changes compared to the sequential code.
2. If you have a more complicated problem this exercise will give you some hints how to start implementing it.

Some remarks regarding parallel programs in DUNE:

- Parallel DUNE programs utilize the *Message Passing Interface* MPI. Many different MPI implementations are known to work well with DUNE.
- By default CMake builds your programs for parallel use so you can build the exercises for this tutorial the same way as usual.
- To actually run the parallel program, you need to call the `mpirun` command of your MPI implementation. If you want to run exercise06-1 with two processes you could use

```
mpirun -np 2 ./exercise06-1
```

- VTK output from `VTKWriter` does not need any special treatment in the parallel case. When applied to a `GridView` of a parallel grid each process will produce output corresponding to his local sub grid (`*.vtu` files). Furthermore the process of zero id will create a file (`*.pvtu`) which corresponds to the global grid.

Exercise 1 Running the Code from the Tutorial

In this exercise you will run the code from the tutorial in parallel and make some simple changes.

1. First run the code from `exercise06-1` and try out different settings. Don't forget to use `mpirun` if you want to use more than one process.
2. Right now the program calculates the squared L2 error separately on every process without communicating. Use `CollectiveCommunication` [1] to calculate the sum over the local L2 errors (squared).

[1] `https://www.dune-project.org/doxygen/pdelab/master/classDune_1_1Communication.html`

3. You could also try different solver backends.

Exercise 2 Communication over the Grid

In this exercise you will learn how to communicate data using overlapping grids and a `DataHandle`. This can be useful if you need to write a parallel application and if you need to do something that is not covered by the `CollectiveCommunication` class.

Communication in the overlap happens entity-wise. That means you have entity related data and each entity in the overlap can send this data to the corresponding entity of the other processor. You can control which codim-entities should send data and for which part of the overlap communication will happen (see partition types in the slides for tutorial06).

The source code for this exercise can be found in the file `exercise06-2.cc`. Right now the code does the following:

- It creates an overlapping `YaspGrid` in two dimensions. You can set the overlap in the ini file `datahandle.ini`.
- The communication happens in the file `communication.hh`. We create the vector `std::vector<int> data` that stores one int for every entity of a given codimension. If you for example choose `codim=1` in the ini file the data vector will have one entry for every edge of the current process.
- We set all entries of the data vector equal to the rank of the current process, i.e. on rank zero all entries will be zero on rank one all entries will be one and so on.
- With

```
// Data handle to communicate the data
// of each edge to the next processor.
using DH = ExampleDataHandle<IndexSet, decltype(data)>;
DH dh(indexSet, data, cdim);
```

  we create a data handle. The class `ExampleDataHandle` is defined above.

- Communication happens by calling the `communicate` method of the `GridView`:

```
gv.communicate(dh,Dune::All_All_Interface,
               Dune::ForwardCommunication);
```

  This methods needs a `DataHandle`, a communication `InterfaceType` and a `CommunicationDirection`. More information about these can be found in the documentation [2] but we don't need that for completing the exercise.

[2] `https://www.dune-project.org/doxygen/pdelab/master/group__GIRelatedTypes.html`

Our goal is to communicate the rank stored in the data vector to the other process in the overlap regions. Right now the `ExampleDataHandle` on top of the file does not communicate anything. Let's change that!

1. Remove the comments from these lines:

```
// switch (communicationType){
// case 1: gv.communicate(...); break;
// case 2: gv.communicate(...); break;
// case 3: gv.communicate(...); break;
// case 4: gv.communicate(...); break;
// default: gv.communicate(...);
// }
```

2. All methods the `DataHandle` needs are already there but the implementations are missing. With the comments from the source code it shouldn't be too hard to fill in the missing pieces.

3. After implementing the missing parts or copying from the solution try to figure out what the communication `InterfaceType` means. Run your program with two processes:

```
mpirun -np 2 ./exercise06-2
```

   Try different communication interfaces by changing the value of `type` in the ini file:

```
[communication]
type = 5
```

   Find out where communication happens for different communication interfaces by looking at the program output and the comments in the source code. Drawing the grid with overlap is highly recommended. When you know what's going on try different codimensions or larger grids with more overlap.