Technische Universität München
Institut für Informatik
Benjamin Rüth
Benjamin Uekermann
Andreas Reiser
Stefan Gavranovic

Summer Term 2018
Worksheet 4

Praktikum Wissenschaftliches Rechnen
Computational Fluid Dynamics

Worksheet 4
Conjugate Heat Transfer

Deadline: June 19, 12:00

In the last regular part of the lab course, we extend our convective heat transfer solver from the second worksheet by a coupling interface to external solid solvers. This allows us to simulate conjugate heat transfer scenarios – coupled heat problems between a fluid and a solid domain. To this end, we use the coupling library preCICE and OpenFOAM as external solid solver.

We recommend to build all software packages on a Linux system, ideally Ubuntu. If you are not confident with building software with multiple dependencies on Linux, you may as well use a virtual machine so that you don't mix them with your main system. Please download and use the provided additional resources from Moodle.

Please submit your code (but no preCICE source files and no OpenFOAM source files), all your case files (but no vtks), screenshots of your results, and a brief `README` text file describing the code usage and your findings in a single compressed file.

1 Conjugate Heat Transfer

On Worksheet 2, we have already considered heat transfer in fluids. Now, we want to simulate heat transfer for scenarios with fluids and solids. This coupled problem is often referred to as *conjugate-heat transfer* and is of significant importance for many practical examples. As an example[1] , consider the cooling of electric components by a fan – as realized in every laptop. We already know how to model heat transfer in fluids (we have used the Boussinesq Approximation). In solids, conduction

[1]For more details and more examples, have a look at `https://www.comsol.com/blogs/conjugate-heat-transfer/`

dominates and convection is negligible:

$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (\kappa_S \nabla T) + Q \ , \tag{1}$$

with the specific heat capacity c_p, the thermal conductivity of the solid κ_S, and a heat source Q.

Coupling condition and coupling approaches

At the common interface between fluid and solid (the so-called coupling interface), we demand continuity in temperature (Equation 2) and heat flux (Equation 3):

$$T_F = T_S \tag{2}$$

$$-\kappa_F \frac{\partial T_F}{\partial n} = \kappa_S \frac{\partial T_S}{\partial n} = q_S \tag{3}$$

where n is the normal vector pointing from the fluid into the solid domain.

We realize these coupling conditions by a partitioned Neumann-Dirichlet coupling approach: In the fluid equations, the solid heat flux q_S is prescribed as a Neumann boundary condition at the coupling interface, whereas in the solid equations, the fluid temperature T_F is prescribed as a Dirichlet boundary condition at the coupling interface. There are other variants, such as Dirichlet-Neumann or Robin-Robin, but these are out of scope for this worksheet.

2 The Coupling Library preCICE

To realize the coupling between our CFD solver and an external solid solver, we use the coupling library preCICE. For information about preCICE, please have a look at its webpage[2] and at its GitHub repository[3] including the user documentation in the wiki. To become acquainted with preCICE, please do the following:

- Run the interactive tutorial `run.precice`[4]
- Build preCICE v1.1.1 and run the tests (see wiki)
- Understand the API by stepping though the adapter example (see wiki)
- Have a look at the C API (see wiki → Non-standard API)

If you have questions concerning preCICE, please use the proper channels of communication: the Gitter chat for smaller problems (including building), the mailing list for larger problems, and the GitHub issues to report proper bugs. Please also give us feedback on the usability and the documentation. Additionally, we will have a **building help desk on Friday, June 8, 12-2.30 pm, in 02.05.060**.

[2] `https://www.precice.org`
[3] `https://github.com/precice/precice/releases/tag/v1.1.1`
[4] `http://run.coplon.de/`

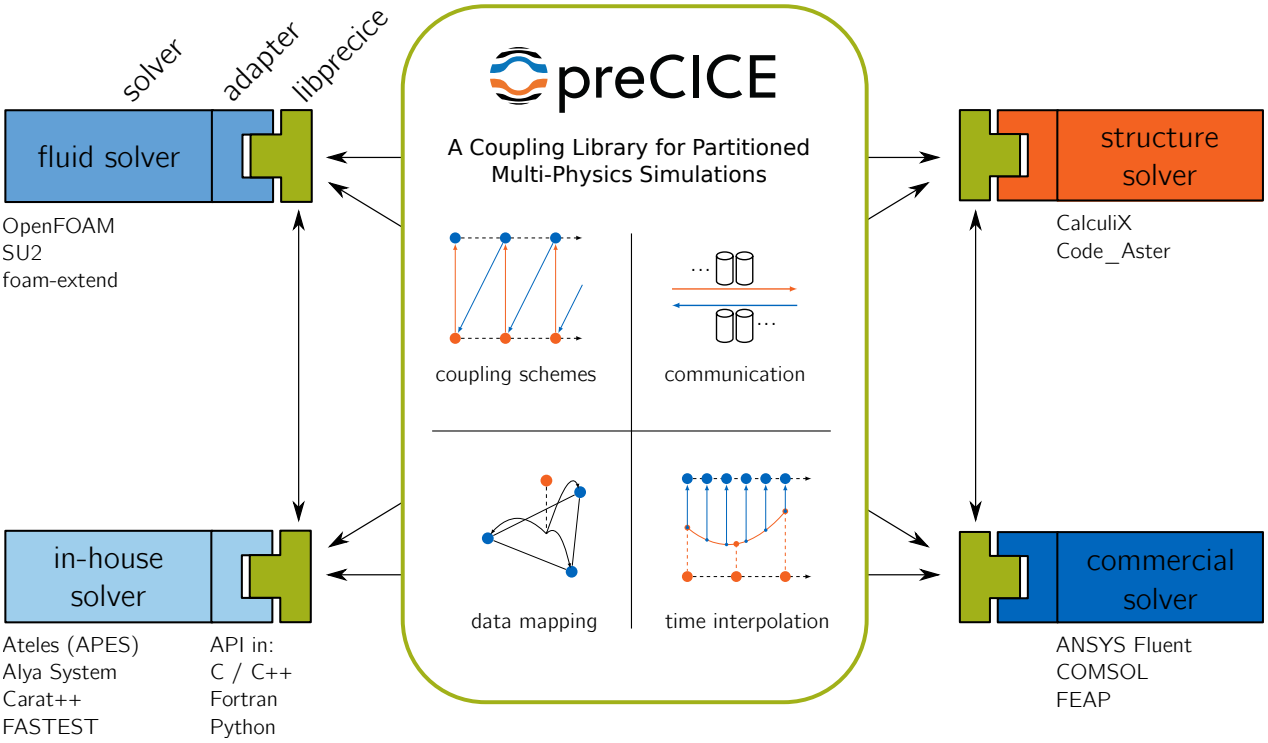

Figure 1: preCICE in a nutshell

3 The Solid Solver

As solid solver, we use OpenFOAM, a very wide-spread collection of numerical solvers, mainly for CFD. OpenFOAM can, however, also be used for simple solid problems, such as our pure conduction problem. Please install version 5.0 from `openfoam.org`. For Ubuntu systems, this is fairly easy[5], using its official PPA repository. Make sure to also read the sections "User Configuration" and "Getting Started" in the installation instructions. To become acquainted with OpenFOAM, you could run a simple testcase as the driven cavity[6], which you already know.

In particular, we use `laplacianFoam` as solid solver. It discretizes the solid equations (1) with the finite volumes method in space and implicit Euler method in time. OpenFOAM only supports 3D simulations. Therefore, we work with quasi-2D setups having only one layer of cells in z direction.

To visualize the output of OpenFOAM, you can load the (intentionally empty) `*.foam` file in ParaView, or convert the output to VTK by using `foamToVTK -case Solid`. From our experience, this works best for visualizing it together with the output of your own solver. Using `paraFOAM`, a wrapper around `Paraview`, which is typically shipped alongside OpenFOAM, is not necessary.

[5] `https://openfoam.org/download/5-0-ubuntu/`
[6] `https://cfd.direct/openfoam/user-guide/cavity/`

The OpenFOAM adapter

To use OpenFOAM with preCICE, you further have to install the OpenFOAM adapter. For step-by-step instruction, please look at the corresponding wiki page[7]. We recommend to further run the CHT tutorial *Flow over a heated plate* as a final introductory step.

4 Implementation

Preparation

In order to adapt the fluid solver for preCICE, we need to call a few methods of the preCICE C interface. For example, the solver needs to give the nodes of the coupling boundary to preCICE and it needs to write and read their values. We group together the adapter-specific functions in the files `precice_adapter.h` and `precice_adapter.c`, a partial implementation of which you can find in the additional resources. Copy them, together with the provided `Makefile`. You should then include `#include "adapters/c/SolverInterfaceC.h"` in `main.c` and check that the code compiles. The environment variable `PRECICE_ROOT` needs to be set externally, as specified in the installation instructions. Some code additions will also be needed in the `main.c` and other source files. See listing 1 for a rough skeleton.

New parameters

We need a few more parameters to be read from the `config.dat` file.

1. Read one double each for the `x_origin` and `y_origin` (see figure 2). Adjust the VTK output to take the origin into account. We also need the origin later to set the right coordinates for the coupling vertices.
2. Read the five strings:
   - `precice_config`, the path to the `precice-config.xml` file,
   - `participant_name`, which should typically be `Fluid`,
   - `mesh_name`, which should typically be `Fluid-Mesh`,
   - `read_data_name`, which should typically be `Heat-Flux`, and
   - `write_data_name`, which should typically be `Temperature`.

   These are needed for the preCICE setup later.

We also need a new option in the `geometry.pgm` file to indicate a coupling boundary. Adjust `init_flag()` accordingly. You already count the number of fluid cells. Start also counting the number of coupling cells. At this point this should not affect anything else, the coupling boundary should behave exactly as a no-slip boundary.

[7] `https://github.com/precice/openfoam-adapter/wiki`

```
read_parameters(...)
// initialize preCICE
precicec_createSolverInterface(participant_name, precice_config, 0, 1);
int dim = precicec_getDimensions();

// define coupling mesh
int meshID = precicec_getMeshID(mesh_name);
int num_coupling_cells = ... // determine number of coupling cells
int* vertexIDs = precice_set_interface_vertices(...); // get coupling cell ids
// define Dirichlet part of coupling written by this solver
int temperatureID = precicec_getDataID(write_data_name, meshID);
double* temperatureCoupled = (double*) malloc(sizeof(double) * num_coupling_cells);
// define Neumann part of coupling read by this solver
int heatFluxID = precicec_getDataID(read_data_name, meshID);
double* heatfluxCoupled = (double*) malloc(sizeof(double) * num_coupling_cells);

// call precicec_initialize()
double precice_dt = precicec_initialize();

// initialize data at coupling interface
precice_write_temperature(...);
precicec_initialize_data(); // synchronize with OpenFOAM
precicec_readBlockScalarData(...); // read heatfluxCoupled

while (precicec_isCouplingOngoing()) { // time loop
        //1. calculate time step
        // use dt = min(solver_dt, precice_dt)

        //2. set boundary values
        set_coupling_boundary();

        //3 - 6. calculate temp, F and G | RHS of P eq. | pressure | new U,V

        //7. coupling
        precice_write_temperature(...); // write new temperature to preCICE buffers
        precice_dt = precicec_advance(dt);          // advance coupling
        precicec_readBlockScalarData(...); // read new heatflux from preCICE buffers

        //8. output U, V, P for visualization and update iteration values
}
precicec_finalize();
```

Listing 1: Skeleton of the changes required in the `main.c`.

Mesh Vertices

preCICE needs to know where the coupling interface is located. All vertices are located at the mid point of either the horizontal or vertical cell edge, see Figure 2. The vertex v_0 located at the bottom wall has coordinates $(\texttt{x\_origin} + (i - 0.5) \cdot \texttt{dx}, 0, 0)$, vertex v_1 located at the left wall has the coordinates $(0, \texttt{y\_origin} + (j - 0.5) \cdot \texttt{dy}, 0)$. For now, we assume to have coupling boundaries only at the four walls and no coupling boundaries on obstacles inside the domain.

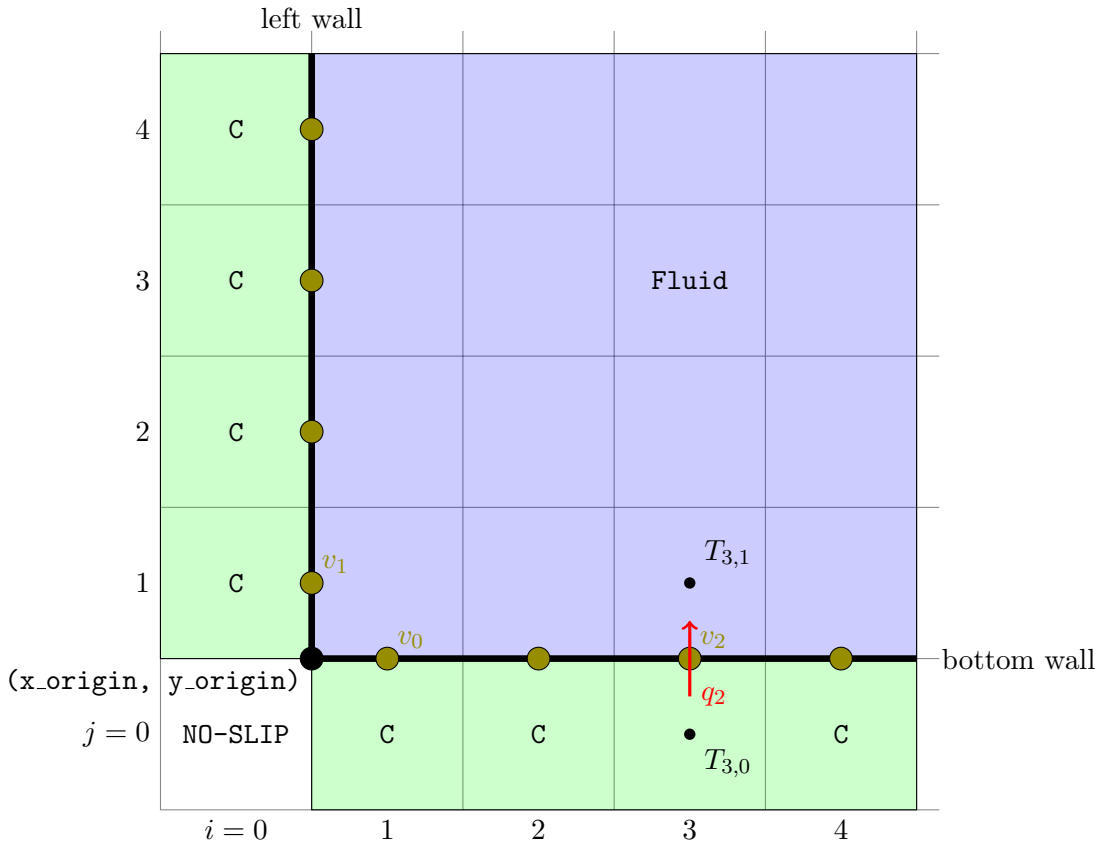

Figure 2: Vertex locations

- Include `#include "adapters/c/SolverInterfaceC.h"` in `main.c`.
- Add two the new files: `precice_adapter.c` and `precice_adapter.h`. You can find them in the provided resources.
- Make sure the code compiles with the new `Makefile`.
- Implement the new function `int* precice_set_interface_vertices(...)`, which calculates the coordinates of the coupling vertices:
  - Allocate a double array `vertices` of size `num_coupling_cells * dimension`, which will

hold the coordinates of all coupling vertices. The layout is `x1, y1, z1, x2, y2, z2, x3, ....`

- Iterate over the walls one after another, e.g. left - right - top - bottom.
- Use `int* precicec_setMeshVertices(...)` to set the mesh vertices in preCICE. `vertexIDs` is an int array of size `num_coupling_cells` which gets filled by preCICE.
- Return `vertexIDs` from your function.
- Don't forget to `free()` the `vertices` array after calling this function.

Mesh and data access

Allocate two double arrays `temperature` and `heatflux` of size `num_coupling_cells`. Implement the following two functions in `precice_adapter.c`. Keep in mind to iterate over the walls in the same order that you did when calculating the coordinates for the vertices.

1. `void precice_write_temperature(...)` which extracts temperature values (`**TEMP`) and writes them to preCICE (`*temperature`). Even though the temperature values are located in the middle of cells and we need values at the boundary, we just use the values of cells bordering the wall: i. e. for vertex v_2 the temperature value for $T_{3,1}$ should be sent, ignoring the distance $\frac{\delta y}{2}$.

   To write data to preCICE, use `precicec_writeBlockScalarData(...)` inside this function.

2. `void set_coupling_boundary(...)` which uses the heat-flux data (`*heatflux`) from preCICE to set the right temperature values (`**TEMP`) at the coupling boundary. This corresponds to setting Neumann boundary conditions for temperature, i.e. for the coupling interface at vertex v_2 we get $T_{3,0} == T_{3,1} + \delta y \, q_2$.

   To read data from preCICE, use `precicec_readBlockScalarData(...)`. See the adapter example[8] for where/how to call it.

preCICE setup and steering

For now we have all the necessary parts to implement the preCICE adapter. Have a look again at the adapter example[9] and adjust your solver accordingly. Note that the C interface has a few differences from the C++ interface (for example, see the function `precicec_createSolverInterface()`[10]). At this point your solver should be able to handle explicit coupling. Proceed to the **Forced convection over a heated plate** example and make sure everything works before proceeding with the next tasks.

[8] https://github.com/precice/precice/wiki/Adapter-Example, (Section 3)
[9] https://github.com/precice/precice/wiki/Adapter-Example, (Sections 1 - 3)
[10] https://github.com/precice/precice/blob/develop/src/precice/adapters/c/SolverInterfaceC.h

Checkpointing

In each time step, preCICE performs sub-iterations for implicit coupling, hence the solvers must be able to save the state at the current time step and reload it. Add two new functions to `precice_adapter.c`, which write and restore the state. We need to save and restore `U`, `V`, and `TEMP`. Therefore, we need three new matrices of the same sizes as `U`, `V` and `TEMP`, which we name `U_cp`, `V_cp`, `TEMP_cp`. These functions should just copy to and from the arrays.

- `write_checkpoint(double time, double **U, double **V, double **TEMP, double **U_cp, double **V_cp, double **TEMP_cp, int imax, int jmax);`
- `restore_checkpoint(double *time, double **U, double **V, double **TEMP, double **U_cp, double **V_cp,double **TEMP_cp, int imax, int jmax);`

For one last time, have a look at the adapter example[11] to see where to save and restore checkpoints. At this point implicit coupling should be working. Proceed to the **Natural convection in cavity with heat-conducting walls** example.

Coupling inside domain

For the last example, we need to be able to also have the coupling boundary condition inside the domain. To save us some complexity and because it is not needed for the last example, we only consider horizontal coupling boundaries inside the domain, see Figure 3. Extend the necessary functions to handle this case.

[11] https://github.com/precice/precice/wiki/Adapter-Example, (Section 4)

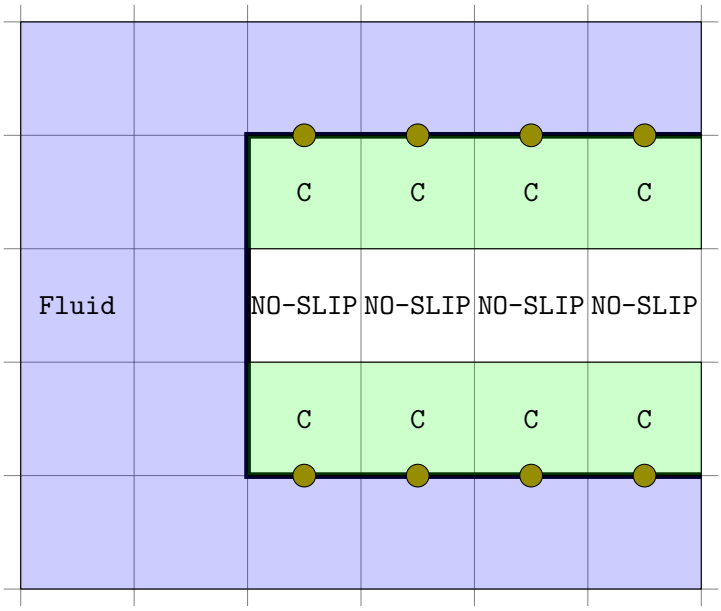

Figure 3: Vertex locations inside domain

5 Example Problems

a) **Forced convection over a heated plate**: The Fluid boundaries are all *adiabatic*, except for inflow, outflow and coupling interface. Use the provided `Solid_plate` setup for OpenFOAM and the `precice_config_plate_explicit.xml` config. Make sure that the coupling interface of your geometry coincides with the interface of the solid. Add a new case `heated-plate` in `spec_boundary_val()` to set the inflow velocity and temperature.

TODO: Line plot to compare and validate results?

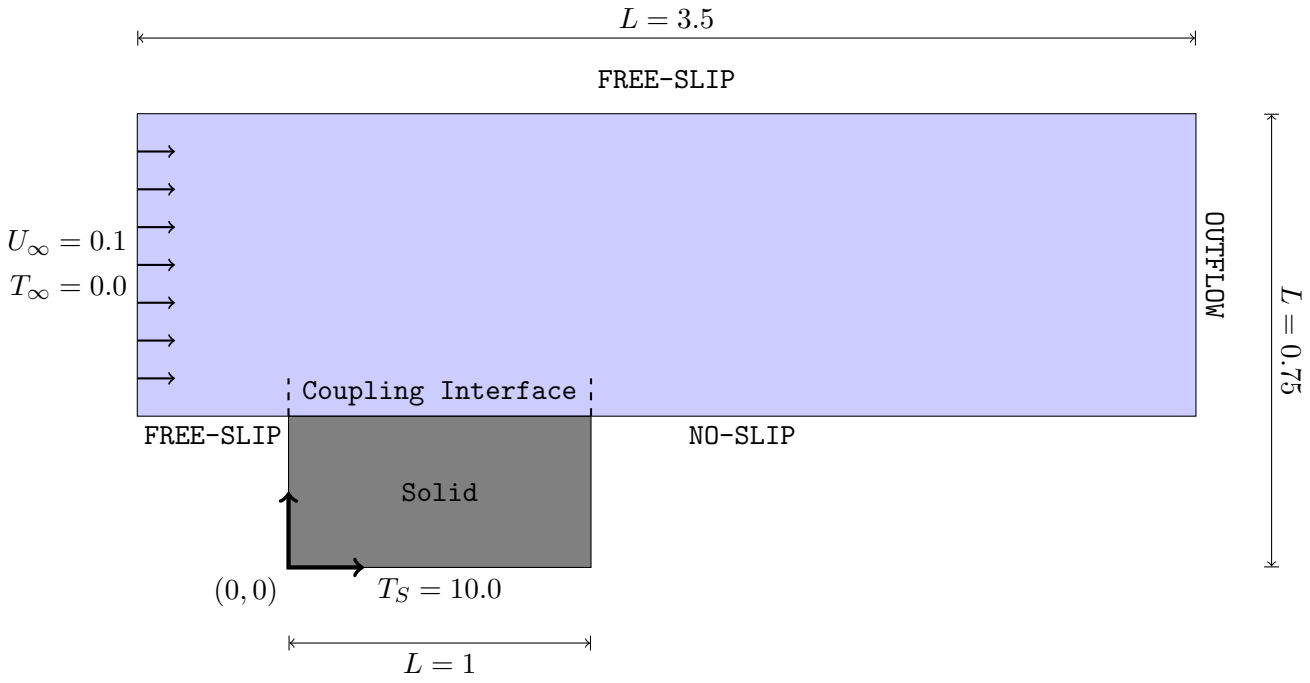

Figure 4: Geometric setup: forced convection over heated plate

b) **Natural convection in cavity with heat-conducting walls**: We revisit the natural convection scenario, this time with a solid enclosure where the walls of the enclosure are heated or cooled. Use implicit coupling, adjust the preCICE-config accordingly.

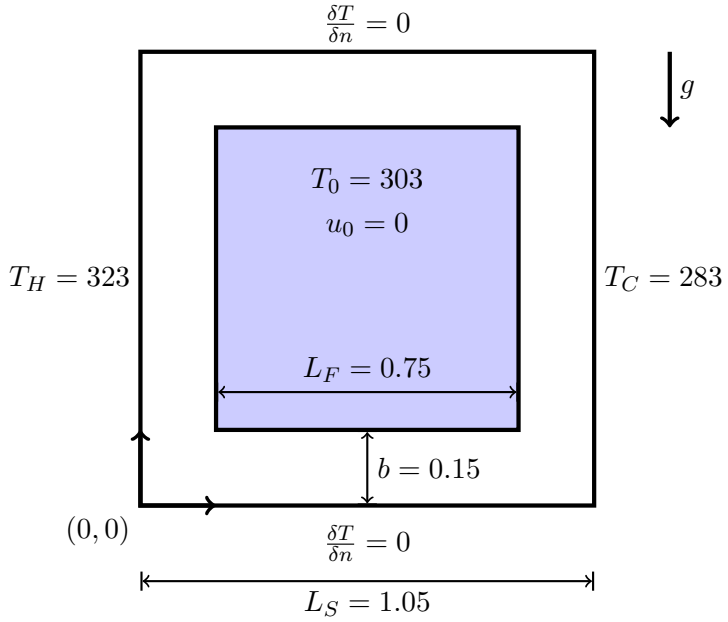

Figure 5: Geometric setup: natural convection with heat conduction walls

c) **2D heat exchanger**: A heat exchanger is a tool that heats up a cold target fluid (here fluid domain 2) by a warm fluid (here fluid domain 1) without mixing both fluids. The two fluid domains, Figure 6 and Figure 7, are aligned at the four thin gray bars. The four bars are the solid participant. Add two new cases `F1-heat-exchange` and `F2-heat-exchange` in `spec_boundary_val()` to set the inflow velocity and temperature. Use the provided case files to run the simulation and make sure it works. Depending on your implementation, you may need to adjust the geometry files. Experiment with different shapes, parameters and 'turning on' gravity, to allow for a better exchange of heat.

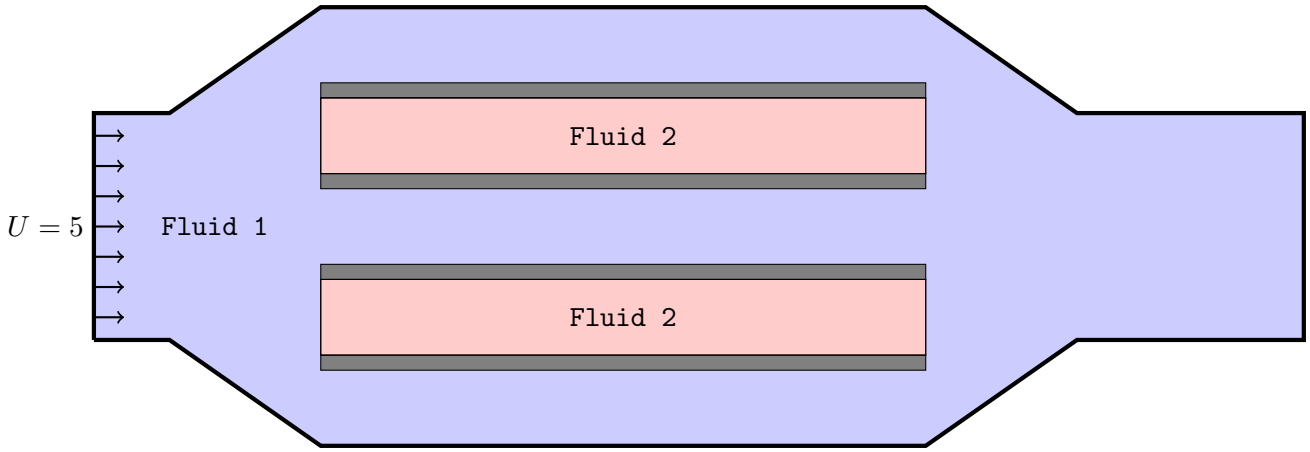

Figure 6: 2D heat-exchanger, `Fluid 1`

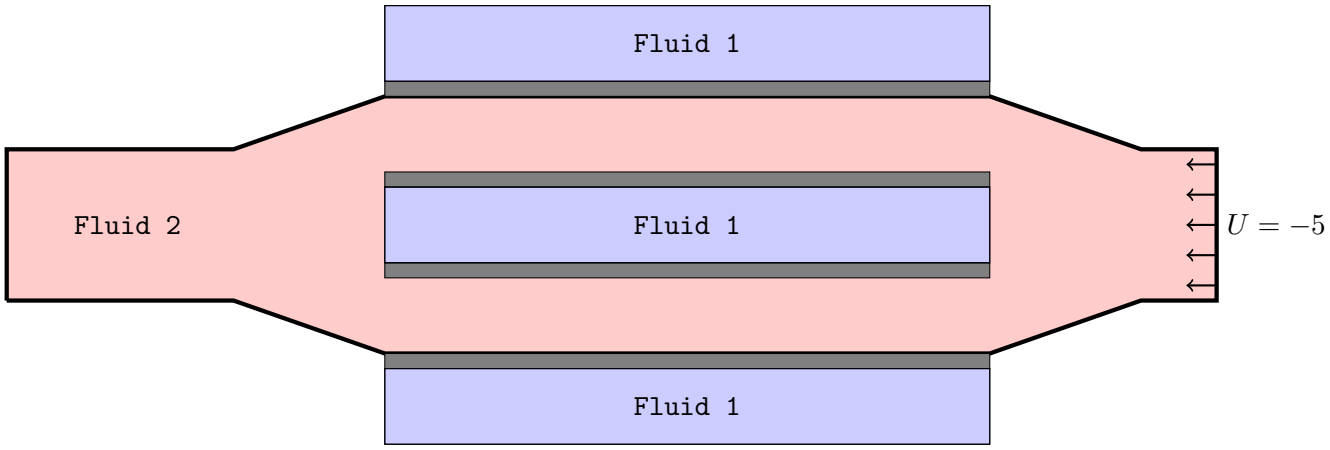

Figure 7: Geometric setup: 2D heat-exchanger, `Fluid 2`