1. We make copy of the iga-ads source code

student@ubuntu:~$ mkdir iga-ads2

student@ubuntu:~$ cp iga-ads/* iga-ads2 -r -f

student@ubuntu:~$ cd iga-ads2

student@ubuntu:~$ rm CMakeCache.txt

student@ubuntu:~$ cmake .

student@ubuntu:~$ make

The examples of the simulations are in src/problems

Some documentation on the code can be find here

https://www.researchgate.net/publication/313532745_IGA-ADS_Isogeometric_analysis_FEM_using_ADS_solver

2. Let us make our own 2D heat example problem

student@ubuntu:~/iga-ads2$ ls src/problems/heat/

heat_1d.cpp heat_1d.hpp heat_2d.cpp heat_2d.hpp heat_3d.cpp heat_3d.hpp

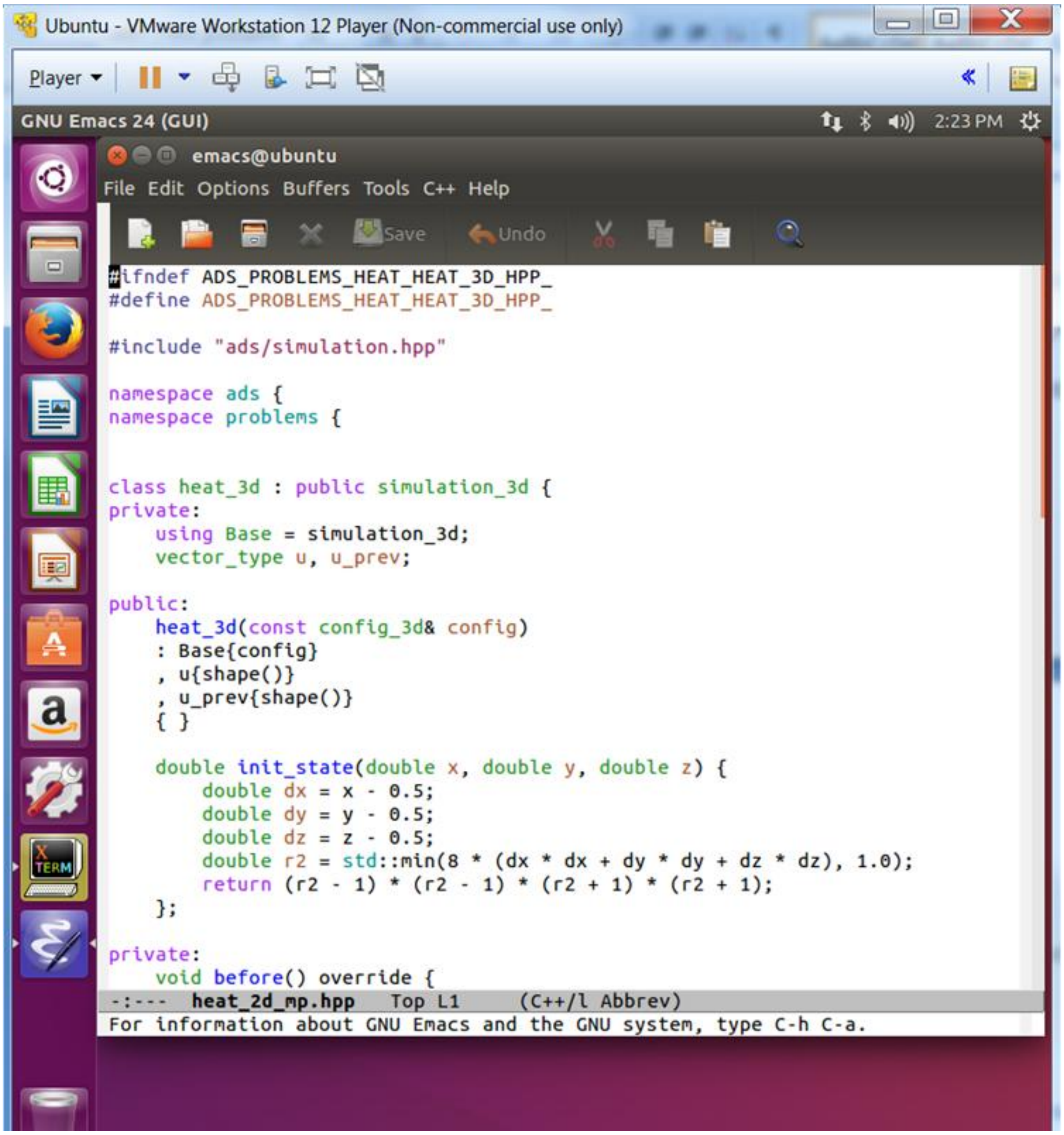
Let us "downgrade" 3d heat simulation into 2d heat simulation

student@ubuntu:~/iga-ads2$ cp src/problems/heat/heat_3d.hpp src/problems/heat/heat_2d_mp.hpp

student@ubuntu:~/iga-ads2$ emacs src/problems/heat/heat_2d_mp.hpp &

Below I comment on some features of the class in blue color

In red color I denote the changes we need to make




```
#ifndef ADS_PROBLEMS_HEAT_HEAT_2D_MP_HPP_

#define ADS_PROBLEMS_HEAT_HEAT_2D_MP_HPP_

#include "ads/simulation.hpp" <- base class for any simulation

namespace ads {

namespace problems {

class heat_2d_mp : public simulation_2d {

private:

    using Base = simulation_2d;
```

```
    vector_type u, u_prev; <-solution from actual/previous
time step

public:

    heat_2d_mp(const config_2d& config)

    : Base{config} <-simulation configuration parameters

    , u{shape()}

    , u_prev{shape()}

    { }

//Setting up initial state

//downgrade to 2D

//    double init_state(double x, double y, double z) {

    double init_state(double x, double y) {

        double dx = x - 0.5;

        double dy = y - 0.5;

//downgrade to 2D

//      double dz = z - 0.5; //remove z

//      double r2 = std::min(8 * (dx * dx + dy * dy + dz * dz),
1.0);

        double r2 = std::min(8 * (dx * dx + dy * dy), 1.0);

        return (r2 - 1) * (r2 - 1) * (r2 + 1) * (r2 + 1);

    };

private:

//called one before the entire simulation

    void before() override {

        prepare_matrices(); <-computes factorized Gramm matrix

// computation of the initial state

//downgrade to 2D
```

```
//         auto init = [this](double x, double y, double z)
//             { return init_state(x, y, z); };

        auto init = [this](double x, double y)
            { return init_state(x, y); };

        projection(u, init);

        solve(u);

    }

//called once at the beggining of each time step

    void before_step(int /*iter*/, double /*t*/) override {

        using std::swap;

        swap(u, u_prev);

    }

//execution of a single time step

   void step(int /*iter*/, double /*t*/) override {

        compute_rhs();

        solve(u);

    }

//here we code the RHS operator

    void compute_rhs() {

        auto& rhs = u;

        zero(rhs);

        for (auto e : elements()) {

            double J = jacobian(e);

            for (auto q : quad_points()) {

                double w = weigth(q);

                for (auto a : dofs_on_element(e)) {

                    value_type v = eval_basis(e, q, a);

                    value_type u = eval_fun(u_prev, e, q);
```

```
//downgrade to 2D

// double gradient_prod = u.dx * v.dx + u.dy * v.dy + u.dz *
v.dz;

      double gradient_prod = u.dx * v.dx + u.dy * v.dy;

       double val = u.val * v.val - steps.dt * gradient_prod;

//downgrade to 2D

//                    rhs(a[0], a[1], a[2]) += val * w * J;

                    rhs(a[0], a[1]) += val * w * J;

                }

            }

        }

    }

};

#endif /* ADS_PROBLEMS_HEAT_HEAT_2D_MP_HPP_ */
```

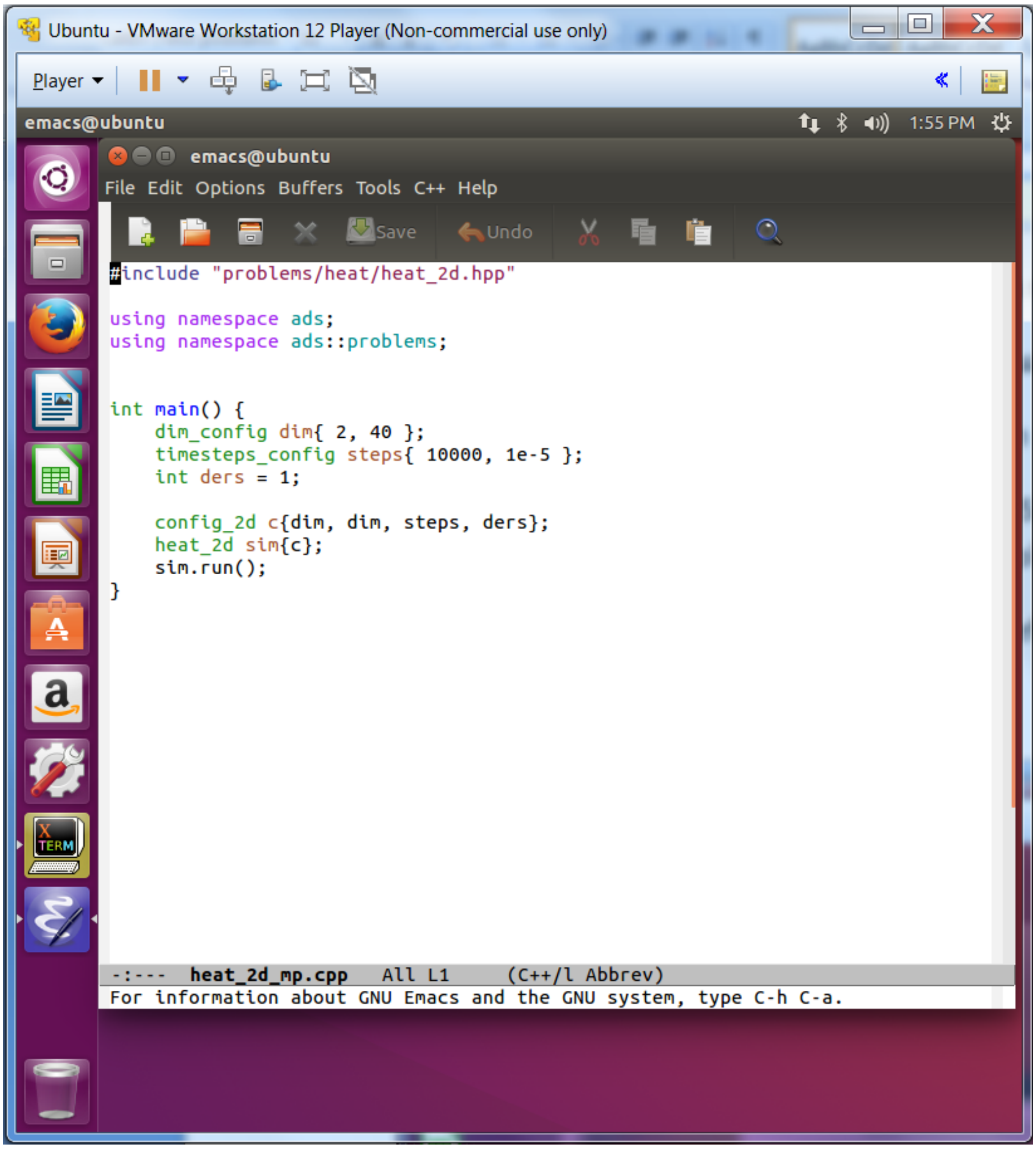
3. We also need to create a pilot running the simulation

student@ubuntu:~/iga-ads2$ cp src/problems/heat/heat_2d.cpp src/problems/heat/heat_2d_mp.cpp

student@ubuntu:~/iga-ads2$ emacs src/problems/heat/heat_2d_mp.cpp &

Below I comment on some features of the class in blue color

In red color I denote the changes we need to make




```
#include "problems/heat/heat_2d_mp.hpp"

using namespace ads;

using namespace ads::problems;

//Suppose we want to run 1000 time steps with time step size
1e-5

//using quadratic B-splines and a mesh of 40x40 elements

int main() {

    dim_config dim{ 2, 40 };

    timesteps_config steps{ 1000, 1e-5 };
```

```
    int ders = 1;
    config_2d c{dim, dim, steps, ders};
    heat_2d_mp sim{c};
    sim.run();
}
```

4. Compilation

The problem that are going to be compiled are listed in CMakeLists.txt e.g.

define_problem(heat_1d

src/problems/heat/heat_1d.cpp)

define_problem(heat_3d

src/problems/heat/heat_3d.cpp)

We need to add

define_problem(heat_2d_mp

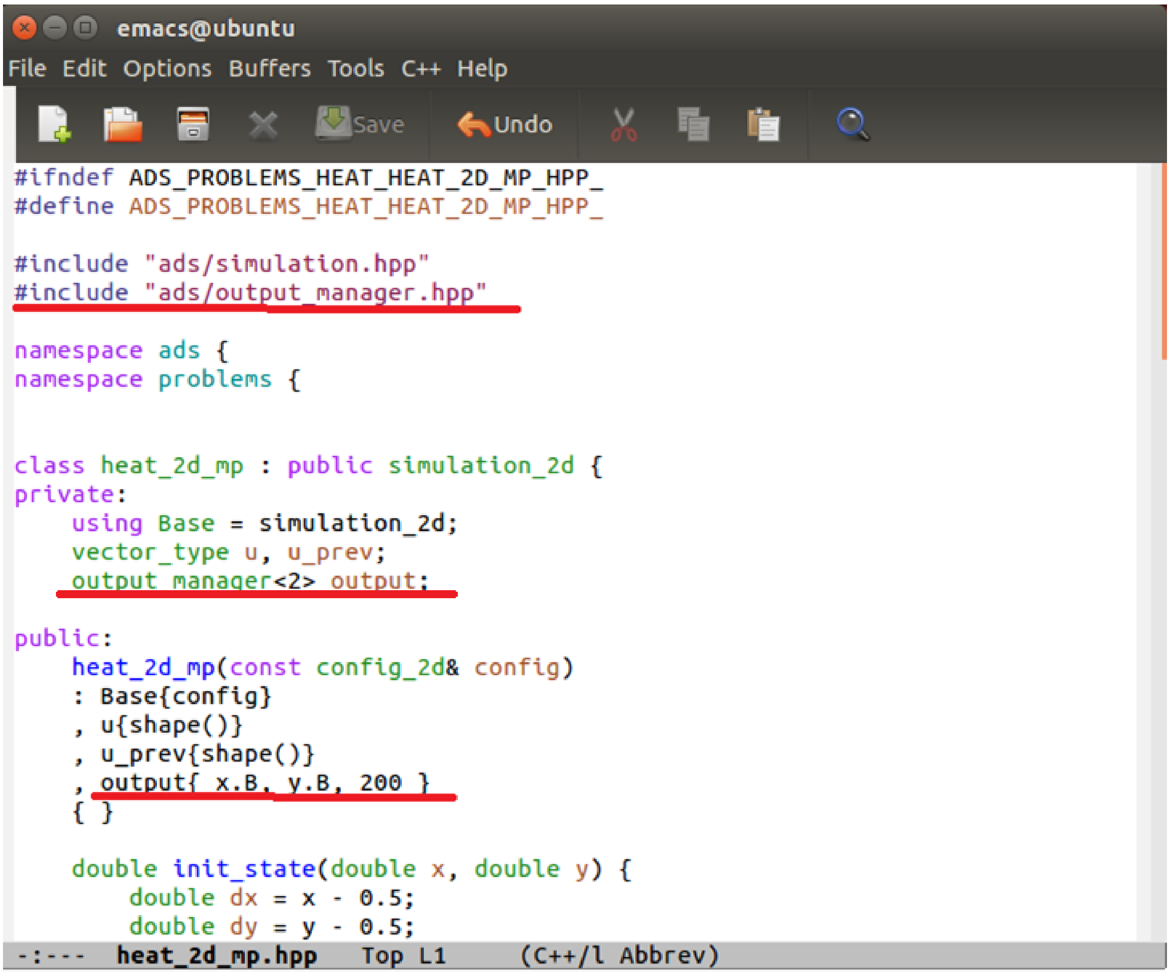
src/problems/heat/heat_2d_mp.cpp)

student@ubuntu:~$ rm CMakeCache.txt

student@ubuntu:~$ cmake .

student@ubuntu:~$ make

5. Generating the output



We include a header to output manager

#include "ads/output_manager.hpp"

We setup a new instance of output manager

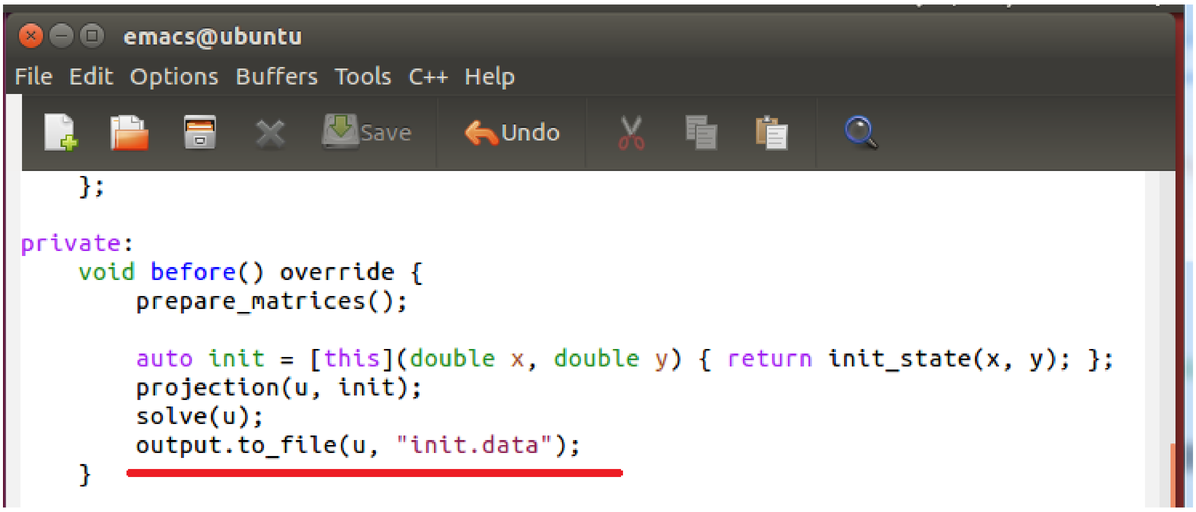
output_manager<2> output;

and we initialize it in the constructor

, output{ x.B, y.B, 200 }

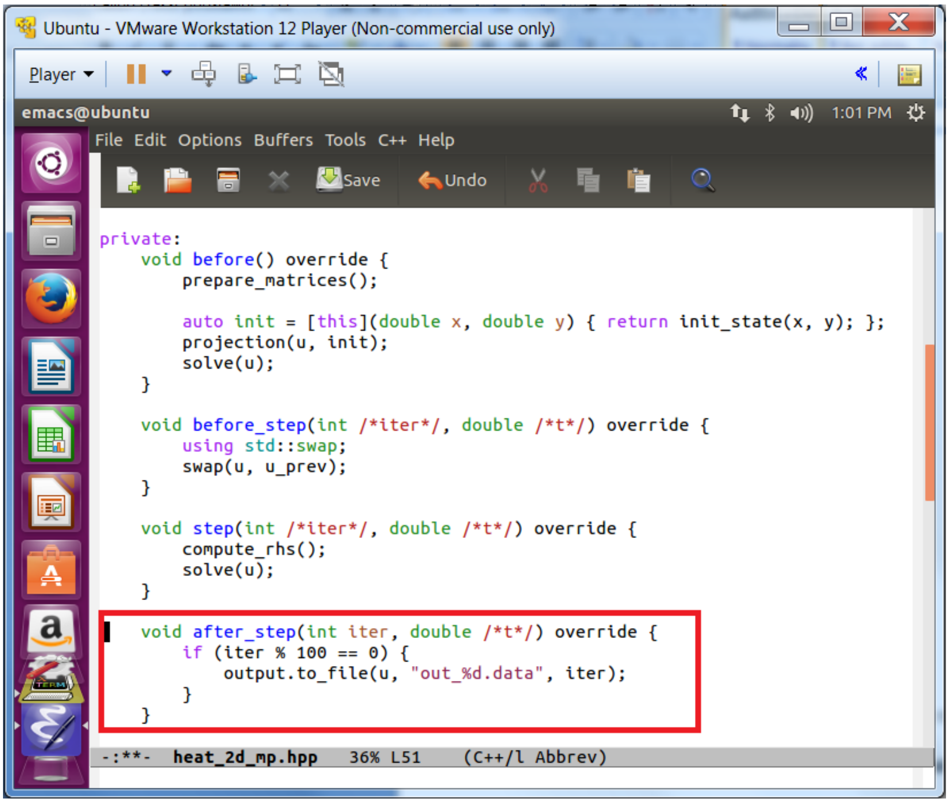
we need to add dumping out the initial state at the routine called at the beginning of the simulation

output.to_file(u, "init.data");



we need to add the routine called at the end of each time step, to dump out state every 100 steps

```
void after_step(int iter, double /*t*/) override {

  if (iter % 100 == 0) {

    output.to_file(u, "out_%d.data", iter);

  }

}
```

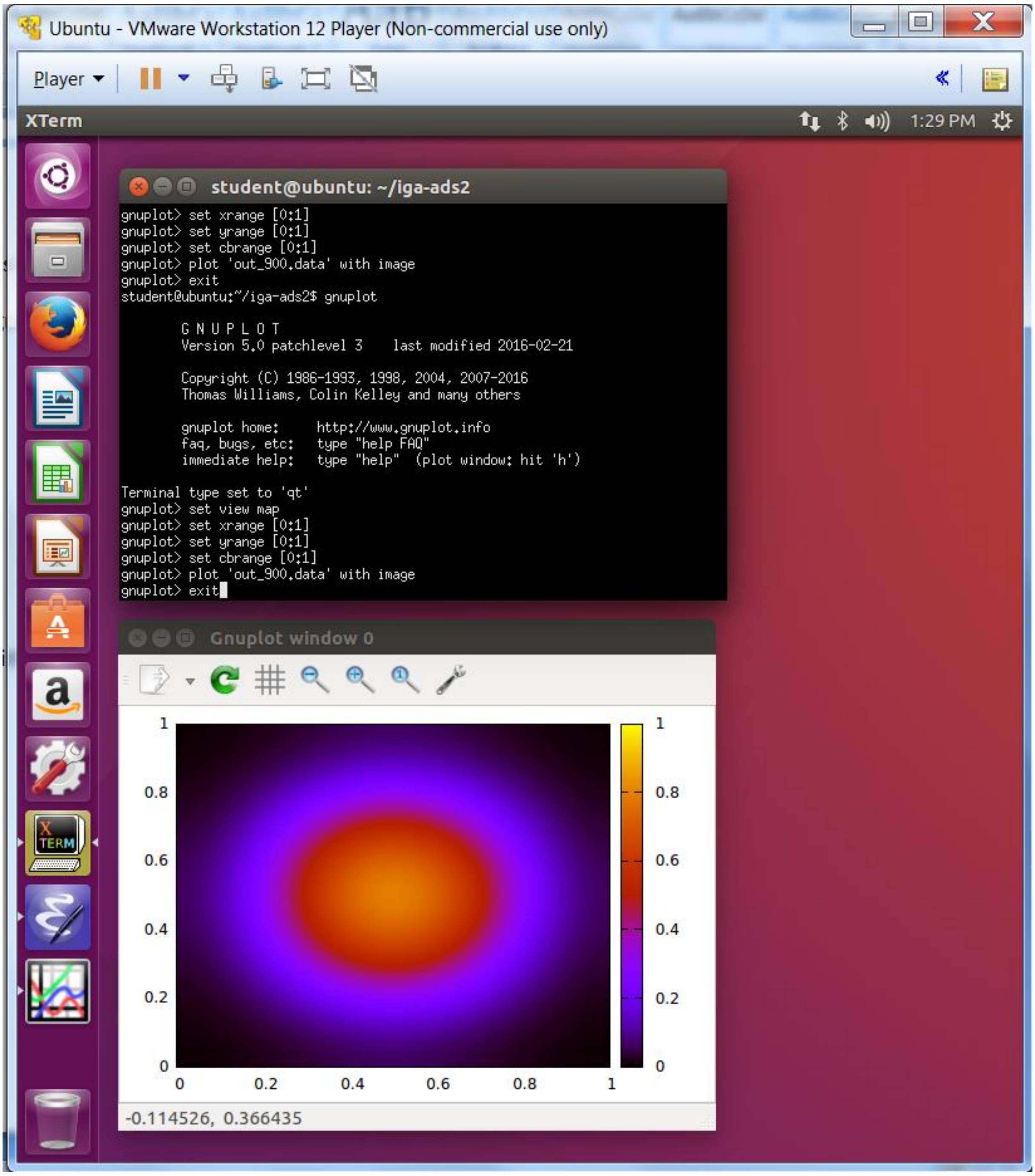


```
student@ubuntu:~/iga-ads2$ make

[ 62%] Built target ads

[ 75%] Built target heat_3d

[ 87%] Built target heat_1d

Scanning dependencies of target heat_2d_mp

[ 93%] Building CXX object
CMakeFiles/heat_2d_mp.dir/src/problems/heat/heat_2d_mp.cpp.o

[100%] Linking CXX executable heat_2d_mp

[100%] Built target heat_2d_mp

student@ubuntu:~/iga-ads2$ ./heat_2d_mp
```

```
student@ubuntu:~/iga-ads2$ ls out* -al

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_0.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_100.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_200.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_300.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_400.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_500.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_600.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_700.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_800.data

-rw-rw-r-- 1 student student 2222055 Jul 19 13:04 out_900.data

student@ubuntu:~/iga-ads2$
```

Now, we can plot the snapshots from the simulation

```
student@ubuntu:~/iga-ads2$ gnuplot
gnuplot> set view map
gnuplot> set xrange [0:1]
gnuplot> set yrange [0:1]
gnuplot> set cbrange [0:1]
gnuplot> plot 'out_900.data' with image
gnuplot> exit
```



We can make a movie

student@ubuntu:~/iga-ads2$ ffmpeg

The program 'ffmpeg' is currently not installed. You can install it by typing:

sudo apt install ffmpeg

student@ubuntu:~/iga-ads2$ sudo apt install ffmpeg

Create gnuplot_script file with

set view map

set xrange [0:1]

set yrange [0:1]

set cbrange [0:1]

plot 'out_0.data' with image

set term png

set output "out_001.png"

replot

plot 'out_100.data' with image

set output "out_002.png"

replot

(Repeat for every data file using 00X consequtive numbers)

student@ubuntu:~/iga-ads2$ chmod a+rwx gnuplot_script

student@ubuntu:~/iga-ads2$ gnuplot gnuplot_script

student@ubuntu:~/iga-ads2$ ffmpeg -framerate 24 -i ./out_%03d.png output.mp4