Assignment 6: Flat World Climate model

Practical information

Deadline: Sunday 20/3 12.00

Resources:
- ERDA for file storage
- Jupyter for the Terminal to access MODI
- Benchmarks through SLURM on MODI

Handin:
- Total assignment: a report of up to 3 pages in length (excluding the code)
- Use the template on Absalon to include your code in the report

Introduction

Climate modelling is a very important research area and is hindered by the fact that simulations need to cover ten-of-thousands of years to be of any value, and at the same time the spatial resolution needs not be very high. Therefore, the area needs faster computers, not bigger computers.

In this assignment we will work on a simple 2D system and use only few time-steps. The climate model is hugely oversimplified; to avoid working with spherical coordinates, we will model the world as flat and replace the sun with a light bulb that oscillates back and forth over the world. Think of it as a computer simulation of a lab-experiment one could perform, except for the periodic boundaries. Real simulations have to run for millions of time-steps but since runtime is linear with the number of steps, we will only run up to a thousand time-steps in the exercise. How much sunlight is absorbed depends on the albedo.

The integration of each time-step consists of three steps:
- *Radiation*: Calculate the warming of the world by the sun.
- *Energy Emission*: Calculate the cooling of the world.
- *Diffusion*: Calculate heat diffusion in the world.
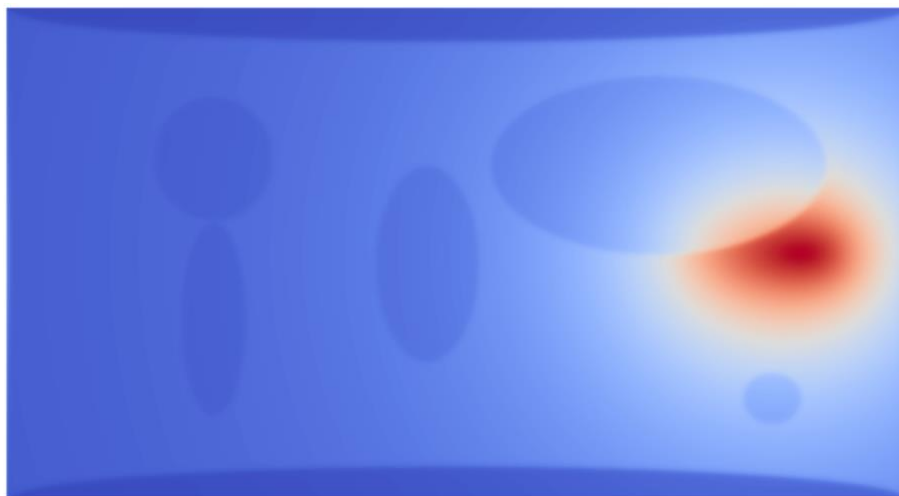
Temperature in the medium model after running for 100 iterations

MODI

For this assignment we need g++ and MPI to compile and MODI for running the benchmarks.

You can read more about MODI in the user guide: https://erda.dk/public/MODI-user-guide.pdf

Spin up a Jupyter session on MODI selecting the "HPC notebook" notebook image. In the terminal (or the folder view on the right side) you can see a number of folders.
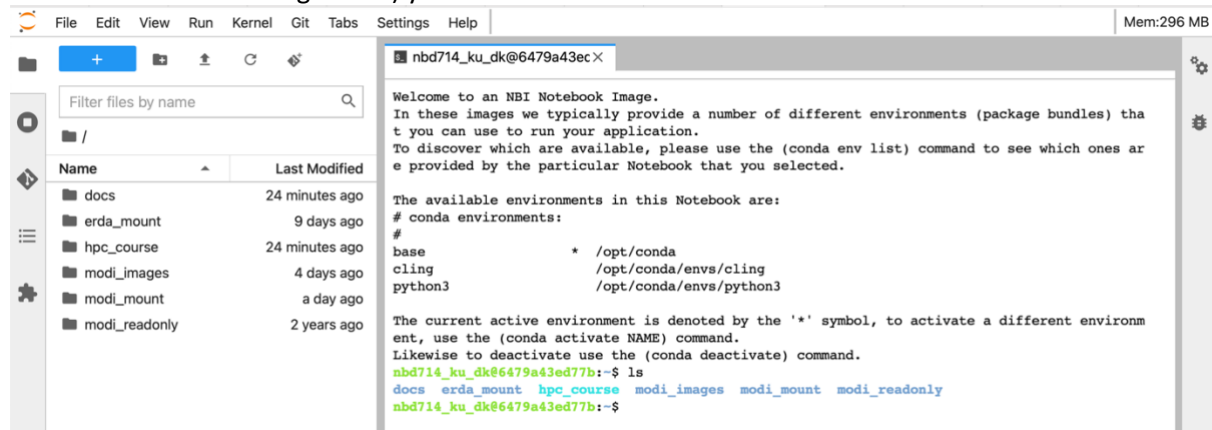
The different folders contain:

`erda_mount`: your own files.

`hpc_course`: course files.

`modi_images`: images of virtual machines that can be used when submitting jobs.

`modi_mount`: this folder is the only one that can be seen from the cluster nodes. You need to copy any executables you use (this is called *staging*) to this filesystem before submitting a job.

PREPARATIONS

Start by copying the exercise to your storage area and enter in to the folder. You can write 'ls' to get a file listing of the folder.

```
cd erda_mount/HPPC
cp -a ~/hpc_course/module6 .
cd module6
ls
```

To be able to edit the files for the exercise navigate to the same folder in the file view. Here you can see five files and a folder with data files:

- Makefile
- fwc_sequential.cpp
- fwc_parallel.cpp
- job.sh
- visual.ipynb
- models/

Before running the code, it needs to be compiled. This can be done by running make in the terminal. The compile line is slightly complicated, because we both need to include the MPI libraries and hdf5 support. Both can be obtained separately by calling the wrappers `h5c++` and `mpic++`. How to combine them? Fortunately, for OpenMPI one can obtain the explicit compile line generated by the wrapper by executing `mpic++ -showme:compile` and `mpic++ -showme:link`. This has been done and added to the Makefile, where we use h5c++ to get hdf5 included.

Two binaries are made: `fwc_sequential` and `fwc_parallel`. The `fwc_sequential.cpp` code is there to give you a reference sequential version of the program. To run the code on the login machine of the MODI you can do:

```
$ ./fwc_sequential --iter 100 --model models/small.hdf5 --out seq.hdf5
```

This will produce an output file in a dataformat called hdf5, which can be opened in python and used to produce a movie for the climate model stored as an animated gif that can be opened directly from the ERDA file explorer.

Correctness: You can check the correctness of the code by looking at min/max/mean and checksums.

Code structure:

The code has three parts: the three routines that implements the physics of the problem (`radiation`, `energy_emmision`, and `diffusion`), the main time loop, including I/O and startup (`simulate`), and the main routine that starts up MPI and reads in the command line parameters. In addition, there are three small routines for reading and writing data in the HDF5 format (`read_world_model`, `write_hdf5`, `write_hdf5_data`).

You can change the workload in the program by loading in different model sizes.

Task 1: MPI parallelise the program (points 7)

The calls to start up MPI have already been added to the parallel version of the program. Your task is to complete the parallelisation. This entails deciding on a domain decomposition, exchanging data between processes where appropriate, making sure that diagnostics used to check correctness are computed correctly, and collecting data on rank=0 when writing our I/O. Besides your implementation submitted as the code and in pdf, you also submit a report through Absalon. In the report, you should explain how you have parallelised the program. Remember to check that you get the same checksum.

Task 2: Strong scaling using SLURM (points 3)

When benchmarking the performance of your program, use the MODI servers also through Jupyter. However, in order to get exclusive access to a machine you need to submit your run through SLURM. An example of a SLURM script, job.sh, that runs the parallel program on 8 cores on one node is included. The "exclusive" flag means that there will only be one user on the nodes. "modi_devel" indicates the queue. Each node has 32 cores, so this is highly wasteful way to run, but it is a good way

to get reliable benchmarks. The singularity image is needed because each notebook image has a different set of software installed.

You should provide experimental results of running using 1, 2, 4, ..., 64 MPI processes. You may also use a higher number of cores. The results should be presented as an easy to read graphs, which shows the strong scaling of the parallel code measured as the wall clock time it takes to run the code as a function of the number of cores. Choose a larger enough number of iterations to get meaningful results. Also, very important, *remember to turn off I/O when benchmarking*.

Running the high-resolution model for many iterations can create large datafiles, so please clean up if you end up creating a lot of data. Do the strong scaling for all three resolutions. Interpret and discuss your scaling results. You can interpret them considering the code, the different workloads and in the context of the distributed memory architecture of ERDA. You can consider adding extra timers to benchmark individual parts of the code.