Assignment 6: Flat World Climate model

Practical information

Deadline: Saturday 16/3 23.59

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

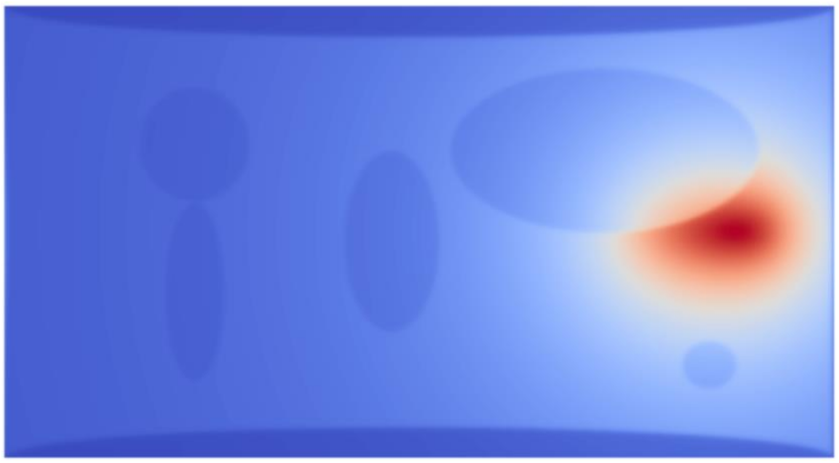
Introduction

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 must 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:

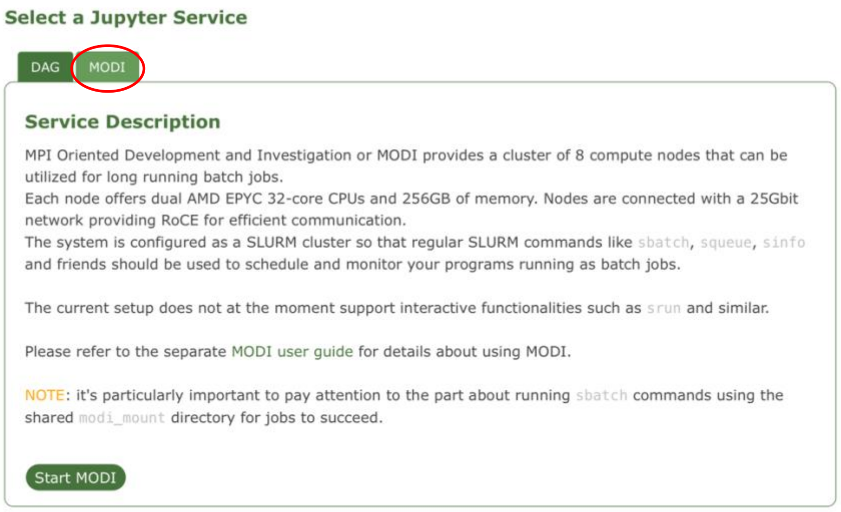
- _**Radiation**_: Calculate the warming of the world by the Sun.
- _**Energy Emission**_: Calculate the cooling of the world.
- _**Diffusion**_: Calculate heat diffusion in the world.



Temperature in the medium resolution model after 100 iterations. The albedos of Americas, Europe-Africa, Asia, and Australia leave an imprint in the temperature.

MODI

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



Service Description

MPI Oriented Development and Investigation or MODI provides a cluster of 8 compute nodes that can be utilized for long running batch jobs.
Each node offers dual AMD EPYC 32-core CPUs and 256GB of memory. Nodes are connected with a 25Gbit network providing RoCE for efficient communication.
The system is configured as a SLURM cluster so that regular SLURM commands like `sbatch`, `squeue`, `sinfo` and friends should be used to schedule and monitor your programs running as batch jobs.

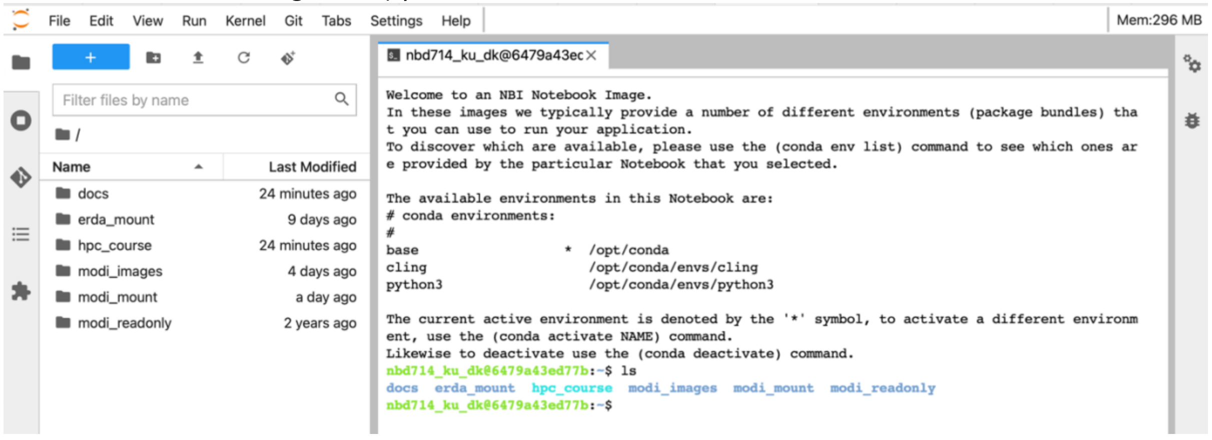
The current setup does not at the moment support interactive functionalities such as `srun` and similar.

Please refer to the separate MODI user guide for details about using MODI.

NOTE: it's particularly important to pay attention to the part about running `sbatch` commands using the shared `modi_mount` directory for jobs to succeed.

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.



Name	Last Modified
docs	24 minutes ago
erda_mount	9 days ago
hpc_course	24 minutes ago
modi_images	4 days ago
modi_mount	a day ago
modi_readonly	2 years ago

```
Welcome to an NBI Notebook Image.
In these images we typically provide a number of different environments (package bundles) tha
t you can use to run your application.
To discover which are available, please use the (conda env list) command to see which ones ar
e provided by the particular Notebook that you selected.

The available environments in this Notebook are:
# conda environments:
#
base                  *  /opt/conda
cling                    /opt/conda/envs/cling
python3                  /opt/conda/envs/python3

The current active environment is denoted by the '*' symbol, to activate a different environm
ent, use the (conda activate NAME) command.
Likewise to deactivate use the (conda deactivate) command.
nbd714_ku_dk@6479a43ed77b:~$ ls
docs  erda_mount  hpc_course  modi_images  modi_mount  modi_readonly
nbd714_ku_dk@6479a43ed77b:~$
```

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. Notice that from the start, the two codes are slightly different because we already include the startup procedures for MPI in `fwc_parallel.cpp`. To run the sequential 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 data format 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. For the parallel code you have to submit it as a job to SLURM

```
$ sbatch job.sh
```

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. In the main loop we always prepare a copy of the global temperature distribution on rank 0, even if it is not written to a file. This is intentional, to test how well the code is able to gather data, and should be kept, even if in a real code this would only be done when needed.

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 and show that you get the same checksum and stats running on different number of cores. The 7 points are distributed as follows:

- Working parallel checksum and stats routines (up to 1 point)
- One dimensional slab domain decomposition of the integration loop (up to 2 points)
- Functioning I/O (up to 2 points)
- Two-dimensional domain decomposition of the problem (up to 2 points)

There can be extra points for elegant solutions or going beyond the minimum (e.g. looking at hybrid OpenMP + MPI).

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 1 core on one node is included. The “exclusive” flag means that there will only be one user on the nodes. You should not use “exclusive” while developing, only when doing benchmarks. “modi_HPPC” indicates the queue. Each node has 64 cores, so using exclusive is a highly wasteful way to run, but it is a good way to get reliable benchmarks. The apptainer 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, ..., 128 MPI processes*. *Do the strong scaling for all three resolutions.* You may also use a higher number of cores, if you can get access to more than two nodes. The results should be presented as an easy-to-read graph, which shows the strong scaling of the parallel code measured as the efficiency. E.g. the wall clock time it takes to run the code with one core divided by the wall clock time it takes to run the code on N cores times the number of cores used as a function of the number of cores:

$$\text{Efficiency}(N_{core}) = \frac{T_{1core}}{T_{Ncore} \times N_{core}}$$

Fit an Amdahl law to the three different models. Choose a large enough number of iterations to get meaningful results. Also, very important, *remember to turn off I/O when benchmarking by using an empty output filename*.

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. Interpret and discuss your scaling results. You should *interpret the results considering the code, the different workloads, and the distributed memory architecture of ERDA*. You can consider adding extra timers to benchmark individual parts of the code to e.g. identify bottlenecks for serial scaling.