Preparing your environment for the practical

January 11, 2023

Contents

1 Login to the cluster

- It may be useful to open the documentation of Smilei in the web browser:
  `http://www.maisondelasimulation.fr/smilei`
  In particular, the documentation explaining the entries of the input namelist (but normally you should find all the necessary explanations in the handouts):
  `https://smileipic.github.io/Smilei/namelist.html`
- Open a shell/Terminal window to work on command line
- Login to the the Ruche cluster and then enter your password, which will not appear on your screen (see the access credentials you have received):
  `ssh -XY username@...`

  Once you are connected you will be in your home space, whose path is also referenced with the shortcut `$HOME`. It is highly recommended to use this space only to compile the code. As explained in the following, run your simulations in the space called `$WORKDIR`.

2 Compile the code Smilei

- Download Smilei:
  `git clone https://github.com/SmileiPIC/Smilei.git`
  and then enter the newly created folder `Smilei`:
  `cd Smilei`
- Clean potentially incomplete build files:
  `make clean`
- Compile the code from the code folder:
  `make -j 10 machine=ruche`
  This should create the files called `smilei` and `smilei_test`. If you see the line `Linking`

`smilei_test for test mode` and no errors are displayed, everything should have worked well. It is normal to see messages like `In file included from ...`

- Compile the postprocessing library `happi`:
  `make happi`

- To know the location of your executable file, just use:
  `pwd`
  This command will display the path to your current working directory, for example `path/to/executable`. This path will be used later. Now your executables `smilei` and `smilei_test` should be found in your folder `path/to/executable`.

3 Prepare your simulation

- Enter your working space:
  `cd $WORKDIR`

- Create a new simulation folder, for example called `sim`, where you will run your simulation:
  `mkdir sim`
  `cd sim`
  Each time you do it, choose a convenient name of the folder to remember which simulation it contains. It is recommended to create a new simulation folder for each simulation in order to avoid overwriting data.

- Inside the simulation folder, create a link to the executables:
  `ln -s path/to/executable/smilei`
  `ln -s path/to/executable/smilei_test`
  The expression `path/to/executable` is just an example: you will need to insert the actual path where your files `smilei` and `smilei_test` are. In the case of the cluster Ruche, the code should be in `$HOME/Smilei`, so the command should be:
  `ln -s $HOME/Smilei/smilei`
  `ln -s $HOME/Smilei/smilei_test`

- Inside the simulation folder, you will need a file to submit a simulation job to the job scheduler, e.g. `submission_script.sh`. You can transfer the file you already have through `scp` or just copy and paste it in a new file inside your simulation folder. A copy of the `submission_script.sh` should be in `cd $WORKDIR/TP-M2-GI` of Ruche, so if you are already inside your simulation folder, you can copy the `submission_script.sh` with this command:
  `cp $WORKDIR/TP-M2-GI/submission_script.sh .`

- Inside the simulation folder, you will need also the input file of your simulation `InputNamelist.py`. A copy of the `InputNamelist.py` should be in `cd $WORKDIR/TP-M2-GI` of Ruche, so if you are already inside your simulation folder, you can copy the `InputNamelist.py` with this command:
  `cp $WORKDIR/TP-M2-GI/InputNamelist.py .`

  Once you have all these files in your simulation folder (executables, submission script, input namelist) you are ready to run your simulation. If you change the name of your namelist, remember that it must be a `.py` file and it must appear a the end of the `submission_script.sh`.

4 Run your simulation

- IMPORTANT WARNING: do NOT launch a simulation directly, always use a simulation job submission script as described below. You are now connected in the login nodes of the cluster, made to transfer files and compile codes, and shared among the connected users. If you launch a simulation directly it will be run on this shared space where all the machine users can connect, slowing down or blocking their operations. Imagine to have a very slow home wifi connection, sufficient only to send some e-mails to work, shared among you and many house-mates. In this analogy running a simulation directly on the login node is equivalent to start a long video-call, blocking everyone elses' attempt to send e-mails and work properly.

  Instead, launching a simulation with a job submission script as described in the following will make the simulation run on the compute nodes, where the necessary resources are safely distributed among the machine users. Science is also learning to work together and to respect each other's space.

- Check if you have all the required files (executables, submission script, input namelist) through the command:
  `ls`

- To check that your namelist does not contain syntax errors, use the **`smilei_test`** executable on the namelist (you'll need to load the same libraries used for the code compilation):
  `./smilei_test InputNamelist.py`
  If you see the line `END TEST MODE`, the namelist does not contain syntax errors and can be run.

- Launch your simulation job:
  `sbatch submission_script.sh`

- To check the status (running/queueing etc) of yout job:
  `squeue -u $USER`
  This should also give you the number `JobId` of your job, necessary for the next command.

- To delete your job from the queue:
  `scancel JobId`

- To read the end of the log file and let it refresh (if you want to watch your simulation execute for example):
  `tail -f smilei.log`
  ctrl+C will allow you to stop watching the file refresh.

- If you want to change the time you want for your simulation, change the corresponding line in the file **`submission_script.sh`** (here 20 minutes):
  `#SBATCH -time=00:20:00`
  The longest simulation of the session runs approximately for 3 minutes with 10 MPI processes and 2 OpenMP threads. These parameters are already set in the submission script.

- If you want to change the number of OpenMP threads in your simulation, change the corresponding line in the file **`submission_script.sh`** (here 2 threads):
  `export OMP_NUM_THREADS=2`

- If you want to change the number of MPI process in your simulation, change the corresponding line in the file **`submission_script.sh`** (here 10 processes):
  `#SBATCH -ntasks=10`

5 Postprocess your simulation results

- Open `IPython` (before, you will need to load the Python modules and define variables like how you did to compile the code, and be sure you have compiled `happi`):
  `ipython`

- Import the libraries you need:
  ```
  import happi
  import numpy as np
  import matplotlib.pyplot as plt
  ```
  The output files have the extension `.h5` and can be opened with the postprocessing library `happi`. You will need also the file `smilei.py`, generated at the start of your simulation.

- Open your simulation:
  `S = happi.Open("path/to/my/results")`
  again, `"path/to/my/results"` is an example, you will need to put the path of your simulation. If you use simply `S = happi.Open()`, the library `happi` will open the results inside the current working directory.

- Now you can use the commands in the section **Crash course on the `happi` postprocessing library** in the handouts.

6 Command line cheatsheet

- `pwd` : shows the path of the current working directory.
- `cd path` : go to `path`
- `ls` : see the content of the current directory.
- `ls path` : see the content in `path`.
- `rm file` : removes `file`. To remove a folder, you will need an additional flag: `rm -r folder`.
- `cp source_file destination_path` : copies `source_file` to the `destination_path`.
- `scp source_file destination_path` : same as `cp`, but you can also transfer folders and files to a different machine, e.g. from the cluster to your computer and vice versa. You will be asked to provide your username, the server address and your password, e.g. `scp source_file username@server:/destination_path/`. This command can be used to transfer output files from the cluster to your computer for later postprocessing if so you prefer (of course larger data files will need more time to transfer).
- `mv source destination` : move `source` (can be a file or directory) to a `destination`. If the `destination` does not specify a path, the command renames `source` with the name `destination`.
- `ipython` : opens `Ipython`, where also the previous commands can be used. To run a Python script inside this interface, use `%run script_name.py`.