High Performance Computing for Science and Engineering I

P. Koumoutsakos, W. Petersen, P. Hadjidoukas
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2017

Set 1 - Matrix Multiplication

Issued: September 29, 2017

Question 1: Getting started on Euler

Euler is the new computing cluster of ETH Zurich, an evolution of the Brutus cluster. The cluster works with a queueing system: you submit your program with its parameters (called job) to a queue and wait for it to be finished. Your first task consists of the following steps:

a) Euler accounts are created automatically when a user logs in for the first time. You will need to enter your nethz username and password. Euler is only accessible within the ETH Network.[1]

b) Login from within the ETH network on Euler via ssh:
`>ssh username@euler.ethz.ch`
and insert your password when asked to.

Congratulations! You are now on a login node of the Euler cluster. In this environment you can write code, compile and run small tests. Keep in mind that there are other people working on the same nodes, so be mindful on how you use them!

c) The Euler environment is organized in modules, which are conceptually packages of settings that can be loaded and unloaded as needed. The basic commands to use the module system are:
`>module load <modulename>`: this command sets the environment variables related to the specified module.
`>module unload <modulename>`: this commands unsets the environment variables related to <modulename>.
`>module list`: lists all the modules currently loaded.
`>module avail`: outputs a list of all the modules that can be loaded.

If, for example, we need to compile a program with the GNU compiler (gcc or g++), we first load its module with
`>module load gcc`
and then proceed with the compilation of our program:
`>g++ -O2 main.cpp -o program_name`

d) Performance measurements and long computations should not be performed on the login nodes but rather they should be submitted to the queue. Moreover, since your simulations might involve a lot of I/O (input/output), you must never run your simulations in your `$HOME` directory, but setup the runs in your `$SCRATCH` space. The disks associated with this space are

[1] You may use VPN `https://sslvpn.ethz.ch` to connect to the ETH Network from home.

designed for heavy loads. Lastly, your quota in `$HOME` is much smaller compared to `$SCRATCH`. Follow the links below for more information. To submit a simple job to the queue, you can use the following command from the folder where your program is stored (use `cd $SCRATCH` to change to your scratch directory):

```
>bsub -n 24 -W 08:00 -o output_file ./program_name program_args
```

This command will submit a job for your executable `program_name` with arguments `program_args` by requesting 24 cores from a single node and a wall-clock time of 8 hours, after which, if the job is not already finished running, it will be terminated. The report of the job, along the information that would usually appear on the terminal, will be appended in the file `output_file`, in the folder from where the job started.

While your job is running you can always use the command:

```
>bjobs
```

to get the status of your jobs.

In order to terminate a job you can use the command:

```
>bkill <jobid>
```

Additional information on the Euler cluster, its instruments and on how to use it can be found at:

- `https://scicomp.ethz.ch/wiki/Euler`
- `https://scicomp.ethz.ch/wiki/User_documentation`

Question 2: Matrix Multiplication

General matrix multiplication (GEMM) occurs in various applications in scientific computing. As a result, highly optimized functions for matrix multiplications have been developed, that try to utilize as much computing capabilities as possible.

In this exercise we will take a look at three naive implementations of matrix multiplication. The first one calculates one element of a target matrix at a time. The second one calculates gradually row by row, and the third column by column.

Finish the skeleton code and analyze the execution times.

a) Finish the matrix multiplication functions.

b) Profile all the variants of the algorithm. Run each matrix multiplication function 10 times and report on the fastest execution time (see `benchmark` function). Why do you not get the same numbers? How many times is the fastest variant faster than the slowest?

   Run the code with and without optimizations (to enable optimizations, add `-O3` flag) and compare the results. Depending on the CPU you are running on, additional speed-up can be gained by telling the compiler to explicitly optimize for that architecture (`-march=native` flag).

c) Try to think of possible optimizations to improve your results.

d) (Optional). Try to install OpenBLAS library or Eigen library. Compare their execution time for matrix multiplication with yours.