ETH zürich

High Performance Computing for Science and Engineering I

P. Koumoutsakos, M. Troyer
ETH Zentrum, CLT F 12
CH-8092 Zürich

Fall semester 2014

Set 1 - Numerical Integration and Multithreading

Issued: September 26, 2014
Hand in: October 3, 2014, 8:00am

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.

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) Similarly to Brutus, 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), 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. To submit a simple job to the queue, you can use the following command from the folder where your program is stored:
`>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.

Additional information on the Euler and the Brutus clusters, its instruments and on how to use it can be found at `http://www.brutuswiki.ethz.ch/brutus/Introducing_EULER` and `www.brutuswiki.ethz.ch/brutus/Brutus_wiki`.

Question 2: Parallel Numerical Integration

The value of an integral $\int_a^b f(x)dx$ can be approximated by computing its Riemann sum:

$$\mathsf{S} = \sum_{i=1}^{n} f(x_i^*)\Delta x$$

where $\Delta x = x_i - x_{i-1} = (b-a)/n$, x_i^* some point in the interval $[x_{i-1}, x_i]$, $x_0 = a$ and $x_n = b$. The *midpoint approximation* uses, in the Riemann sum, the middle point $\bar{x}_i = \frac{(x_{i-1}+x_i)}{2}$ of each interval $[x_{i-1}, x_i]$. In this question you will implement and parallelize this method.

a) Write a (serial) program that calculates the integral $\int_1^4 f(x)dx$, where $f(x) = \sqrt{x} \cdot ln(x)$, using Riemann sum with the midpoint approximation.

The code can be found in the file `integral_seq.cpp`. A Makefile allows for straightforward compilation of both the serial and parallel codes by typing: `make`.

b) Parallelize your code by starting several threads to compute the Riemann sum. Make sure you do not introduce race conditions and verify your implementation by comparing the final result with that computed by the serial program. Note that each thread should handle a different interval of the integral.

The parallel code can be found in the file `integral_mt.cpp`.
The number of threads can be specified as a runtime argument to the executable, e.g. the command: `./integral_mt 4`, runs the parallel code with 4 threads.

c) Choose an appropriate number (n) of intervals and check how the wall-clock time for computing the integral decreases with respect to the number of threads. Plot the time versus the number of threads and report your observations.

Notes:

- You can optionally use the `timer` class, implemented in the `timer.hpp` file, for your time measurements.
- As your Brutus account may not be available, the time measurements can be performed on any computer system of your preference (e.g. personal laptop). In this case, you should report the hardware/software configuration of that system (i.e. number and type of cores, operating system and compiler).

The measurements were conducted on the Euler cluster, allocating a single compute node (`bsub -n 24`) and running all the experiments within a single batch script. Each experiment was repeated multiple times and the lowest time was taken. In Figure 1, we observe that the execution time decreases as the number of threads increases. However, the code fails to scale on more than 20 threads, a behavior which is better depicted in the strong scaling plot of Figure 2. The observed performance degradation is attributed to overheads introduced by the thread scheduler of the operating system. This issue can be resolved by applying thread pinning, i.e. binding each thread to a specific processor core. This technique is automatically supported by OpenMP and will be discussed in the related lectures and exercises.

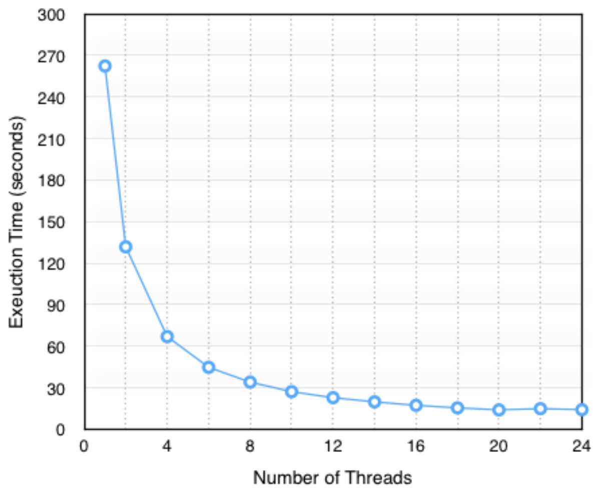

Figure 1: Execution time vs Number of threads.

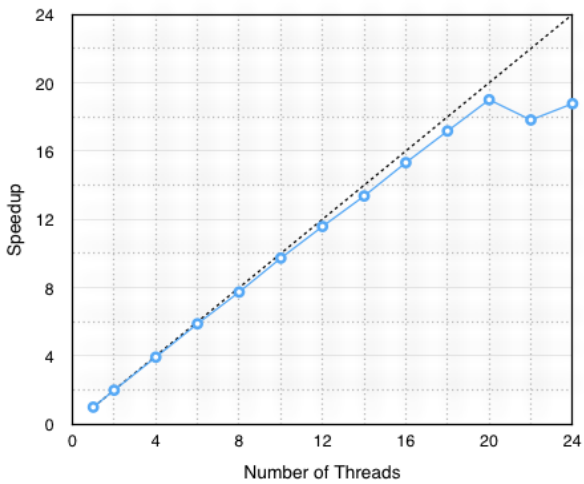

Figure 2: Strong scaling plot.

Summary

Summarize your answers, results and plots into a PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.