# Assignment 1

Introduction to Parallel Programming

due **Monday 13 September 2021, 23:59** (hard deadline)

## Instructions

Although it is allowed to work alone, we suggest that you try to find another student to work together and register in a group for Assignment 1 in Studium. (You will be allowed to change groups, if you want, in the next assignments.) In case of trouble, contact the assistant (clement.poncelet@it.uu.se).

Submission checklist:

- Submissions must clearly show your name(s).
- Submit a **single** PDF report, as well as all source code related to the exercises in Studium.
- Solutions must be in C++11 (or newer, but you should mention it on your report).
- Use appropriate synchronization mechanisms from the C++ standard library http://en.cppreference.com/w/cpp/thread. Do **not** rely on `semaphore.h`, `sem_t`, etc.
- All source code must compile and run on the Linux lab machines.
- **Provide instructions for compilation and running.**
- No source code modifications must be required for reproducing your results.
- Your report must describe the theoretical concepts used as well as all relevant details of your solution.

Recall that you need to score half the points (5 in this case) or more to pass an assignment. In case you do not reach a fully working solution for some exercise, describe your partial solution, the main challenges you did not manage to overcome, and proposals to address them, for partial credit. Please keep your answers short and concise, but clear and complete.

## Exercise 0: "Hello, World!" in C++ (0 points; no need to submit)

**a)** Write a simple "Hello, World!" program (i.e., a program that simply prints "Hello, World!") on the console in C++. If you're new to C++, you can find many good tutorials on the web: for instance, one of them is http://www.cplusplus.com/doc/tutorial/.

**b)** Use `g++` to compile your program. Run the resulting executable.

## Exercise 1: Concurrency and Non-Determinism (1 point)

The `non-determinism.cpp` program, available on Studium, creates a number of independent threads. Each thread simply prints that it is running, and then prints that it is terminating.
Compile the program with: `g++ -std=c++11 -Wall -pthread non-determinism.cpp -o non-determinism`
Run the program several times. Discuss and explain the observed output. Are there other possible outputs that you did not (yet) observe?

## Exercise 2: Shared-Memory Concurrency (1 point)

The `shared-variable.cpp` program, available on Studium, creates three threads that operate on a shared integer variable.

- The first thread repeatedly increments the variable by 1.
- The second thread repeatedly decrements the variable by 1.
- The third thread repeatedly prints the value of the variable.

All threads terminate after approximately one second.
Compile the program using: `g++ -std=c++11 -Wall -pthread` and run it several times. Discuss and explain the observed output.

## Exercise 3: Race Conditions vs. Data Races (2 points)

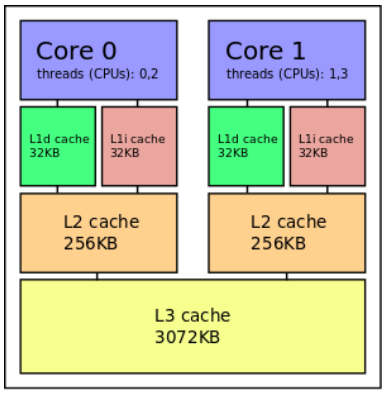
Study the source code of `non-determinism.cpp` and `shared-variable.cpp`. Does either program contain a race condition? Does either program contain a data race? Justify your answers.

## Exercise 4: Multicore Architectures (2 points)

If you're not already working on that server, log in to the server `gullviva.it.uu.se` by using the command `ssh login123@gullviva.it.uu.se` (substitute your user name for `login123`).

**a)** Use the `lscpu` command to obtain some information on the CPU architecture of your lab machine. How many (logical) CPUs does your machine have? How many (physical) processors (i.e., sockets) and processor cores? How many hardware threads are running on each core?

**b)** The diagram below shows the CPU architecture and cache layout of an Intel I5-450M processor. However, `gullviva` has a processor from AMD, not from Intel. Use the information gained from `lscpu` and `lscpu -p` to draw (either by hand or with a drawing program) a similar diagram for `gullviva.it.uu.se`. What are the main differences compared to an Intel-based processor?




(Image source: http://diego.assencio.com/images/linux/cache-hierarchy-intel-i5-m450.png)

## Exercise 5: Performance Measurements (2 points)

Download the `performance.cpp` program from Studium and compile it with `g++ -std=c++11 -Wall -pthread`. The program allocates and initializes an array of $N$ MB. It then creates $T$ independent threads (where $T$ and $N$ are given as the command-line arguments). Each thread performs some work on a separate part of the array, of size (approximately) $N/T$ MB. When all threads are finished, the array is deallocated.

**a)** Using a (departmental) machine with at least 16 physical cores, run the program for *each combination* of $T \in 1, 2, 4, 8, 16, 32$ and $N \in 1, 2, 4, 8, 16, 32$. (You can either do this manually or write a simple script to automate it.) Note the reported run times.

**b)** Visualize your measurements in a chart (graph), e.g., in a line chart or three-dimensional graph. You are encouraged to use suitable software (such as LibreOffice Calc, Excel, etc.) to produce the chart.

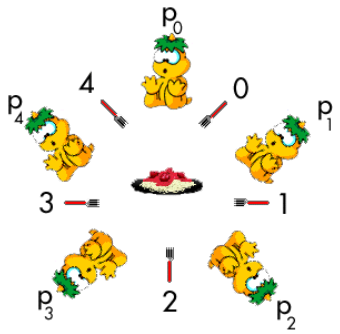
**c)** Discuss and attempt to explain your measurements.

## Exercise 6: Dining Philosophers (2 points)

Once upon a time, some philosophers were sitting around a table. On the table was a big bowl of spaghetti, and between each pair of adjacent philosophers was a fork. The philosophers spent most of their time thinking. But every so often, one of them would get hungry. To eat some spaghetti, a philosopher needs to pick up *both* the fork to her left and the fork to her right. After eating, she puts both forks down again, so that they may be used by her neighbors.

Download the `dining.cpp` program and compile it with `g++ -std=c++11 -Wall -pthread`. The program simulates $N$ dining philosophers (where $N$ is given as a command-line argument). Each philosopher behaves as follows:

1. think until the left fork is available, pick it up;
2. wait until the right fork is available, pick it up;
3. when both forks are held, eat for a fixed amount of time;
4. then, put the right fork down;
5. then, put the left fork down;
6. repeat from step 1.




(Image source: http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/randomized-algorithms/examples/distributed.html)

Your tasks are the following:

**a)** Run the program several times for different values of $N$ ($2 \leq N \leq 10$). What output do you observe? (You can use Ctrl+C to interrupt the program.) Discuss and explain (as far as you can) the observed output.

**b)** Devise and implement a solution to the dining philosophers that does not suffer from the problem(s) you observed in part **a)**.

For further information about this problem and some pointers, read the Wikipedia page on the dining philosophers: http://en.wikipedia.org/wiki/Dining_philosophers_problem.
Hint: Use Thread Sanitizer https://github.com/google/sanitizers/wiki/ThreadSanitizerCppManual to better understand what is going on and what is the problem. (Compile using the flag `-fsanitize=thread`).

**Good luck!**