Practice 1
Study and proposal for the parallelization of an application

Objectives:

- Learn to find problems and applications derived from **any branch of knowledge**, whose **computational load justifies** undertaking a parallelization process → When is it justified to undertake a parallelization process?
- Find or **design** a **sequential** application suitable for parallelization on centralized (i.e., multithreaded applications) or distributed (multithreaded applications) memory parallel machines.
- Justify by means of metrics **already known in other subjects** why the application in question is suitable for parallelization.
- Propose suitable architectures for the parallelization of your application (multi-core systems, GP-GPU, *clusters*, heterogeneous systems, big.LITTLE, multiprocessors, ...).
- Study how compilation parameters can affect application performance on a parallel machine.
- Apply methods and techniques specific to this subject to estimate the **maximum gains** and **efficiency** of the parallelization process.

Development:

Task 1: Search or development of a good candidate application for parallelization

In this first practice the students will have to face a process of **search or design of a proble**, of **any nature and branch of knowledge** (engineering, mathematics, physics, chemistry, psychology, etc.), whose solution is given in terms of a software application of high computational cost. The search for information should be done through the usual channels: scientific journals, books, Internet, tutorials, Sarteco Conference website, Zenodo repository, search in public repositories such as github/gitlab, etc.

✓ **Note 1.1**: It is not necessary to formally understand the specific problem (since it could belong to the field of a discipline other than Computer Science), but it is necessary to understand the **structure of its implementation** and its **operation**. The sequential program must be written in **C or C++** and **must not be interactive**, i.e. it will take some parameters at the beginning of its execution (command line or a text file) and after a (sufficiently long) time it will deliver the results. In case of opting for an interactive program (e.g. chess game), two instances of the same program must be confronted in order to make the set non-interactive and to be able to apply the usual metrics.

✓ **Note 1.2**: This task should be completed in **2-3 hours** of practice time. During this time they should communicate the problem they are working on to the practical teacher so that he/she can approve it and continue with the rest of the tasks.

✓ **Note 1.3** At the end of the second practice session each group should report:

- The title and a brief description of the problem you are going to study.
- Final list of all members of the group.

This information **will be made public** on the Virtual Campus (Moodle).

✓ **Note 1.4:** We recommend that you review the titles of the contributions to the Parallelism Workshop or the Embedded and Reconfigurable Computing Workshop to find parallelization proposals that you can adapt to this practice:

http://www.jornadassarteco.org/programa/?anyo=2024
http://www.jornadassarteco.org/programa/?anyo=2023
http://www.jornadassarteco.org/programa/?anyo=2022
http://www.jornadassarteco.org/programa/?anyo=2021
http://www.jornadassarteco.org/programa/?anyo=2019

http://www.jornadassarteco.org/programa/?anyo=2018
http://www.jornadassarteco.org/programa/?anyo=2017

You can download some conference papers via the ZENODO website: https://zenodo.org → search for papers related to Communities (Communities) js2019, js2021, js2022, js2023 etc.

Recommendations to Task 1:

- A problem that can be parallelized must "take" a sufficiently long time for it to make sense to parallelize it. This is because the mechanisms that you will introduce in your code (in the following practices) of creation/destruction of threads or processes, and of communication and synchronization, will suppose a *time overhead* that may not be negligible, making it not make sense to undertake its parallelization.
- Matrix multiplication is a good example of a fully parallelizable problem, however, this type of problem, where 100% of the algorithm is parallelizable (100% data parallelism) is not of interest to us in this practice. Look for a problem where, when parallelizing, data parallelism and other types of parallelism are balanced (review the slides of unit 3 in the Virtual Campus).
- **Remember:** It is allowed to develop synthetic programs (designed by each group).
- The coding of the application can be original for each group or taken/adapted from a bibliographic resource, always **duly referenced**.

Task 2: Study of the problem prior to its parallelization

It involves **studying the sequential coding of your program (previous task)**, describing in maximum detail the main features of the code.

Help: In the defense of this part, it should be demonstrated why the chosen code is a good candidate for parallelization.

Task 2.1 Perform the **Inter-Task Dependency Graph** of your sequential code. Does the graph reveal "chunks"/tasks of the application whose execution can be implemented in parallel (i.e., tasks whose sequential execution order we can mess up and leave the application output invariant)?

Example:
Network of dependencies between tasks: $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow ... T_n$, where
T_1 : Program initialization.
T_2 : Calculation of the convolution of matrix A and matrix B ...
T_3 : ...

Task 2.2 Regarding the variables your program uses:

- Study how is the **read/write access to each one** and how are the variables used (example: large data arrays, rarely accessed data, many automatic variables, ...).
- Answer: Can the program flow be **structured** more appropriately for more efficient execution on a parallel machine?
- Can we anticipate any problems/improvements with the **cache** by modifying the program in some way?

Help: You will need to locate which are the hot variables (the ones that are accessed the most) and those whose access is punctual.

Task 2.3 Computational load variation study:

Study how **the load** of your problem may **vary** and its impact on the performance of your problem. This type of dependencies are easily represented by graphs, **include them**. Example: How does the *speed-up* change if we vary a certain parameter X of our problem.

Generate graphs of the type:

Speed-up or efficiency ***versus*** `parameter_that_scales_our_problem`.

Help: The speed gain (*speed-up*) of your application may strongly depend on the **dimensions** of a certain matrix, the **accuracy** of some output, etc.

Task 2.4 "Tuning" the compilation parameters

The implementation will have to run under the Linux operating system (Ubuntu 20.04 from EPS) and the GCC compiler (gcc/g++). Study and reflect in a table some of the parameters and compilation options that you think can benefit the parallel performance of the application the most. Explain briefly what each selected parameter does.

They shall:

- Present evidence of how certain parameters affect the performance of your application (e.g. changes in execution times). Remember that we are not yet parallelizing, just **fine-tuning** the compilation process.
- Present evidence of autovectorization in some parts of your program. That is, show that the compiler uses SIMD instructions to perform certain tasks.
  Help: Use `"gcc -S program.c"` or `"objdump -j .text -D program.elf"` to explore what the compiler does.

Help: Check what `"gcc -help=target"`, and `"gcc --help=optimizers"` (ditto for g++ if using C++) do at https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html.

Task 2.5 Analyze and demonstrate (by graphing) what combination of compiler parameters and options "squeeze" or improve the output produced by your compiler. Some speed gain can be achieved if we learn to use these parameters correctly. On some problems this effect will be more apparent than on others. In your case, what does this effect look like, and what do you think it is due to?
Help: you should argue with concepts such as *cache*, more efficient use of memory, instruction generation specific to your processor, etc.

Task 2.6 Analyze and study the performance of the application. Try to explain what the variations in *speed-up* are due to as a consequence of varying the parameters that scale the problem.

Task 4: Warm-up for the following practices:

Solve the following problems:

- One tap takes 4 hours to fill a certain water tank and another tap takes 20 hours to fill the same tank. If we use both taps to fill the tank, which is initially empty, how long will it take? What will be the gain in speed? And efficiency?
- Suppose you now have 2 taps that take 4 hours to fill the tank. Same questions as above.
- And now suppose you have 2 taps that take 20 hours. Proceed with the calculations as well.
- You now have 3 taps: 2 of those that take 20 hours and 1 of those that take 4. What would happen now?

Task 5: Generation of deliverables:

Structured report of the internship with **ALL** the previous sections duly completed (follow the recommendations of your internship teacher).

It is mandatory to include: objectives, presentation of the proposed problem, detailed study of the proposed application, performance analysis, defense of the program, why it is a good candidate for parallelization, parallel architectures suitable for the execution of the properly parallelized program, bibliography.

Implement at least the following rules in a *makefile*:

`make clean` (clean working directory)
`make` (build the project)
`make run` (run the program with default parameters).

Note 3.2: The practical should be handed in by the method of your practical teacher's choice before the practical session the week of October 7-11. Therefore, 4 sessions (8 hours) of laboratory practice will be available.

Note:

The theoretical/practical work must be original. The detection of copying or plagiarism will result in a grade of "0" in the corresponding test. The direction of the Department and the EPS will be informed about this incidence. Repeated misconduct in this or any other subject will lead to the notification of the corresponding vice-rectorate of the faults committed so that they can study the case and sanction according to the legislation.