Introduction

Objective

This project aims to solve a specific problem by means of MPI and OpenMP programming models. Specifically, the main aim is to understand the given real problem, to provide different parallel solutions and analyse the scalability of the provided solutions and the effects of the design decisions.

Material to Deliver

You should deliver the source code of the different activities of the project: "*OpenMP program*" and the "*Hybrid program: MPI + OpenMP*"; according to the deadlines given below. The code will be delivered using a *repository*. The code should contain explanations about the structure, the main decisions and the library functions used to implement the solution. For each delivery, you should provide a brief PDF report (about 4 pages) with at minimum the following items:

I. **Scalability of the Program**: You must show the Execution Time, together with the *Speedup and Efficiency* for different number of processes and problem size.
II. **Obtained Results:** You must give an explanation of the results for the main metrics (*Speedup and Efficiency*) and the reasons for the obtained results.
III. **Conclusions:** Critical analysis of your solution and results, also including a discussion about the obtained results on previous implementations.

Software and Instructions

In order to do the project, you need to install in your local computer a *ssh* client and a *user account* in the EPS teaching cluster (moore.udl.cat). In the cluster, you will have all the software that you need to do the project.

Likewise, in order to develop and test the code on your local computer, you will need to install any software that implements OpenMP (version 2) and MPI. In relation to MPI, you can use whatever you want, but we recommend the open software MPICH2, which is portable and very popular. You can download and find information about its installation and its use at the following address:

http://www.mcs.anl.gov/research/projects/mpich2/

To run the serial code for this practice, there are no special requirements, such as a graphical server (You can view the output with a text editor or any other application that reads files with ppm extension such as the gimp application). You can download the serial code from "*Project" folder in the Resources Section* on Sakai.
You should deliver reports throughout the Sakai virtual campus, by means of the corresponding *Assignment*. The link to the **repository** with the source code

must be included in the corresponding report. **The activities can be done by groups of two students maximum.**

Deadlines for the deliveries and Percentage of the final mark:

- Delivery 1 (OpenMP Program): 23th of April (25%)
- Delivery 2 (MPI Program): 28th of May (25%)
- Delivery 3 (Hybrid OMP-MPI) 11th of June (15%)
- Delivery 3 (Comparison of the results): 18th of June (15%). Remember that you can choose to do the "Comparison of the results" or the "Supercomputer report" recently published. Whatever you choose, you will have to expose in a public session scheduled at the end of the course.

Note:

In order to make the report understandable and easily evaluated or reviewed by any reader, you must follow the rules of good practices to prepare a report. The report must contain the Title of the activity and the authors. Use the editing techniques properly: tables, figures, cross-references, table of contents, etc. The report should be complete, justifying the design decisions, making a proper analysis of the performance and the comparison of the effects of different alternatives to reach the more appropriate decision. You can insert some code pieces for helping to explain your solutions. The figures must be made correctly. You must explain/justify the basic structure of your implementation.

Some common mistakes that you should avoid:

- Vague explanations without justification.
- Explanations of the implementation without justify the structure and the decisions you took.
- Lots of tables with data. You have to identify the most important data and summarize your explanations. Be concrete. Add value with your contributions.
- The figures are not properly created, bad axis ranges, no labels, blurry, or added as screenshot.
- The figures are few and not representative, lots of figures that can be summarized, etc.

Description of the Problem

We want to implement a parallel version for a popular program, the *Mandelbrot set*. The *Mandelbrot set* is a geometric figure of infinite complexity (fractal nature) obtained from a mathematical formula and a small recursive algorithm, as shown in the following figures:

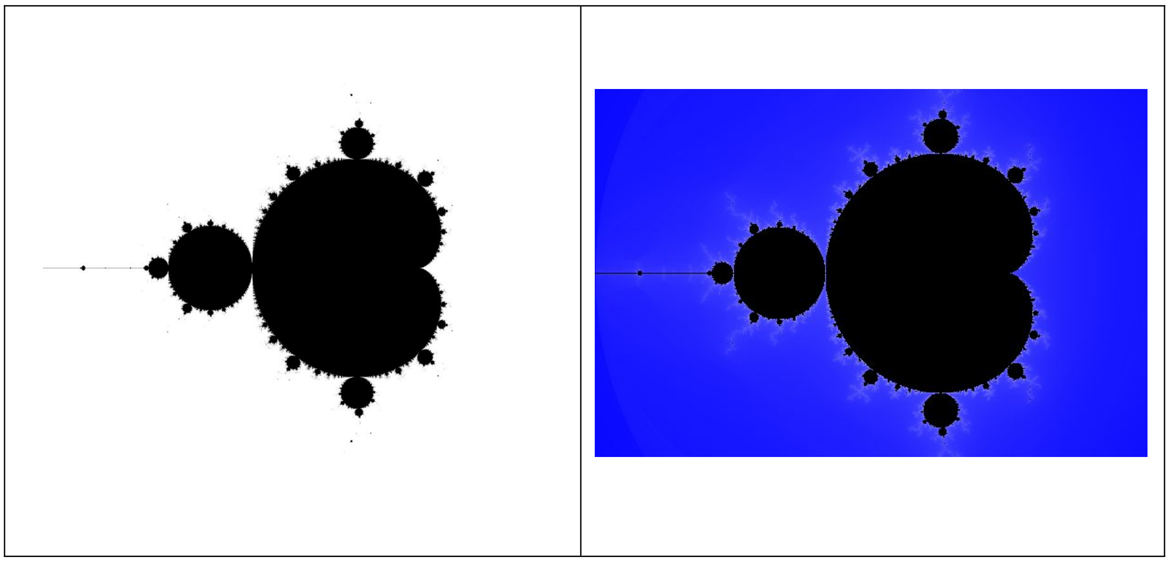

Figure 1 Mandelbrot set. (a) Geometric figure. (b) Output result by the delivered source code.

The code provided in this project is a serial implementation (in C) of the Mandelbrot set that is shown in the Figure 1 (Right), where the intensity of the background colour indicates the proximity of each point to an element of the set (*Black*: element belongs to the set, *White*: it is close to the set and *Blue*: far from the set). To compile and run the code, you should follow the next instructions:

```
$gcc mandelbrot.c –o mandelbrot -lm
$./mandelbrot > sortida.ppm
```

-lm is used for compiling the math library, not using this flag will result in a compilation error.

Remember that it is totally forbidden to execute the serial/parallel versions of the program on the cluster front-end. Doing this you can collapse the server or produce a server downfall. By this, it is important to use the front-end queuing system even for serial programs.

For queuing **serial processes** and execute them, follow the next instructions:

```
$qlogin -pe smp 4   #This command is requesting for an exclusive machine
user@compute-0-X$. #Notice that the prompt changed to the assigned machine
user@compute-0-X$ cd <hpc_project_folder>
user@compute-0-X hpc$ ./ mandelbrot > sortida.ppm
```

Project Statement

Delivery 1: OpenMP Implementation

In order to obtain the best performance of an application implemented with a parallel programming model, we should start by optimizing the application at node-level. According to this, the first activity will be focused on study the operation of the sequential version, analysing the pieces of code candidate to be parallelized following a work and data decomposition model. Next, we will apply the OpenMP directives to parallelize the corresponding code according to the hardware and the suitable work decomposition model at node level.

In this section you should elaborate a report discussing about the following points:

- Justify the parallel design: hotspots, bottlenecks, parallel inhibitors…
- Justify the selected partitioning pattern of the source matrix: size of the partitioning, data location, loops parallelization configuration, and so on.
- Test the performance obtained using different thread scheduling policies (static, dynamic, guided) and discuss about the results.
- Analyze scalability and speedup in relation to the number of threads and size problem. Consider on your discussion the effects of the synchronization, sharing and private data, mutual exclusion, etc., in your solution.
- (It will be highly assessed if you Show the results of the roofline model obtained from your openMP solution using the Intel System Studio tool and discuss the limits of your proposal and possibilities for performance improvements.

For the evaluation, even more important than the quality of the solution is the correct explanations of the obtained results and your design decisions.

Delivery 2: Implementation with MPI

In this activity, you should propose a new implementation combining both programming models, MPI and OpenMP. You should compare the performance of this hybrid solution in relation to the previous delivery for different problem and processes sizes.

MPI implementation is providing us the possibility to apply a data decomposition model. For this end, we can consider different options to implement the task mapping:

a) Static mapping of tasks

It consists in dividing the region into a fixed number of fragments, which are processed in parallel by different processors/cores. Note that the allocation is done at the beginning of the execution of the program and cannot be modified. Can be classified into quadrants, rows, columns, etc. Figure 2 shows an example.

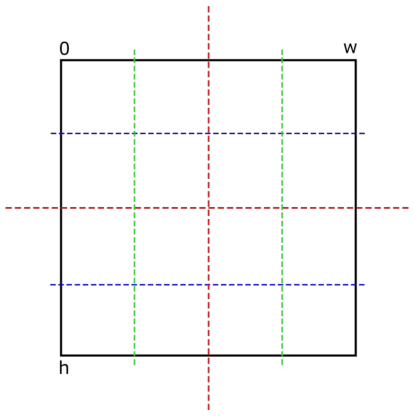

Figura 2 Static problem partitioning

b) Dynamic mapping of tasks

In this case, the fragments will be mapped to different processors/cores as soon as they finish with the previous calculations.

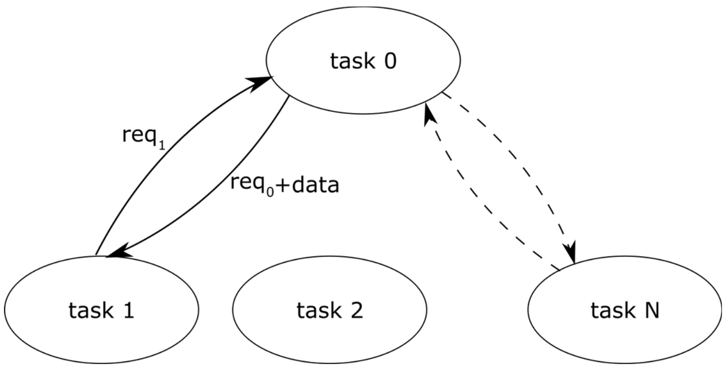

Figura 3 Dynamic problem partitioning

You must analyse the mapping tasks options from the point of:

- The Speedup obtained for different number of cores and problem size.
- The Load balancing.

Explain the strengths and weaknesses of each implementation and discuss the scenarios where you think it would be more convenient to use each one and and why. Consider on the discussion the effects of heterogeneity on the execution nodes.

In order to measure the time used by MPI programs, we recommend the use of the *MPI_Wtime* function, as the following example shows:

```
float x, y, z;
double start, elapsed;
start = MPI_Wtime();
z=x*y;
elapsed = MPI_Wtime() - start;
```

Delivery 3: And the best solutions is…?

This activity is optional. You can choose between this activity or the writing work about “Supercomputers”.

This activity consists in the analysis of different approaches, identifying the best solution based on the obtained performance and justify why, and also whether is it possible to further improvements and how. The aim of this activity is to promote critical thinking and help to identify the strengths and weaknesses of the own work.