Introduction

Objective
This project aims to solve a specific problem by means of MPI and OpenMP programming models. Specifically, the main aim is to parallelize the given problem and analyse the scalability of the proposed solutions and the effects of design decisions.

Material to Deliver
You should deliver the source code of the different sections of the project: "*OpenMP program*", "*MPI program*" and the "*Hybrid program: OpenMP and MPI*"; according to the deadlines given below. The code will be delivered using a *github repository*. The code should contain explanations of the main decisions and library functions used to do the parallel implementation. For each delivery, you should also do a brief PDF report (maximum 4 pages) with the following items:

I. **Scalability of the Program**: You should show the Execution Time, together with the *Speedup* for different number of processes and problem size.
II. **Obtained Results:** You should give an explanation of the results for the main metrics (Speedup and Efficiency) and the reasons for the obtained results.

We will analyse the delivered solutions and will publish one of them as reference.

Finally, you can produce a new report with a comparison between the reference solution and your proposal. You must analyse differences on performance, justify the reasons and describe your proposals for future improvements in both solutions. The aim of this activity is to promote critical thinking and help to identify the strengths and weaknesses of your own work. **Note that this last activity is optional. You can choose between this last activity or analyse, in detail, one of the top twenty Supercomputers listed in the Top500: https://www.top500.org/lists/top500/2020/11/**

Software and delivery 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 and 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 find the source code in the front-end of the *moore.udl.cat* cluster, folder "*/share/apps/files/convolution/code*" and also in the **Project** folder that you can find in Resources section of the virtual campus.

You should deliver reports throughout the Sakai virtual campus, by means of the corresponding *Activity Section*. A ***github*** reference to the code must be included in the corresponding report. **All the activities can be done by groups of two students.**

Deadlines for the deliveries and Percentage of the final mark:

- Section 1 (OpenMP Program): 9th of May (25%)
- Section 2 (MPI Program): 30th of May (25%)
- Section 3 (Hybrid Program): 6th of June (15%)
- Section 4 (Comparison of the results): 18th of June (15%)

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 correctly formatted considering aspects such as titles, axis labels, legend, type of lines, symbols, colours, etc. The text in the figures, labels and axis must be legible. The axis must be correctly dimensioned. The type of chart must be selected correctly depending on that you want to explain. Figures must be accompanied by a brief explanation.

Description of the Problem

Convolution is a *mathematical* operation on two functions **f** and **g** used to produce a third function that is interpreted as a modified version of one of the original functions. Basically, it gives the area overlapped between both functions as a function of the amount that one of the original functions is translated, as it is shown in Figure 1.

Convolution is used on probability, statistics, computer vision, natural language processing, image and signal processing, engineering, and differential equations.

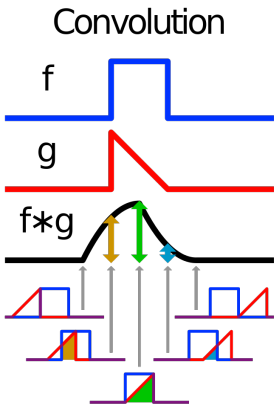


Figure 1 Convolution process

In order to clarify the objectives of the convolution process, it will be presented applied in the field of image processing. In this field, the convolution process is widely used to change the intensity of each individual pixel to reflect the intensities of the surrounding area.

A common use of convolution is to create image filters as could be blur, sharpen, and edge detection. Figure 2 shows the results after applying the emboss filter.

Figure 2 Emboss effect

The convolution process in the field of image processing is based on applying a matrix named as “convolution kernel”. This matrix consists of an *n*×*n* matrix identifying the surrounding area of the evaluated pixel. Each cell of the kernel contains a pixel weight in respect of the pixel.

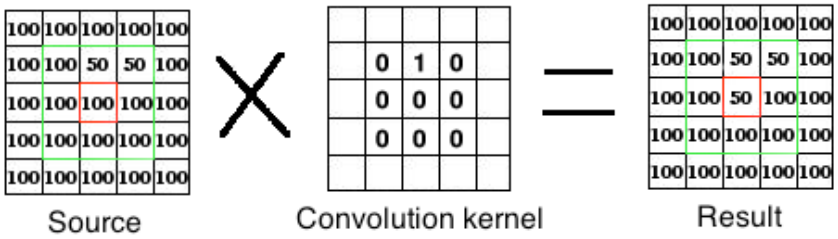


Figure 3 3x3 Convolution Kernel. The red cell represents the evaluated pixel.

In the example shown in Figure 3, the convolution process corresponds to the individual multiplication of every paired cells/pixels, from the source and the kernel. Finally, it obtains the sum of all of them, which corresponds to the resulting pixel value, as it is presented in the Figure 4.

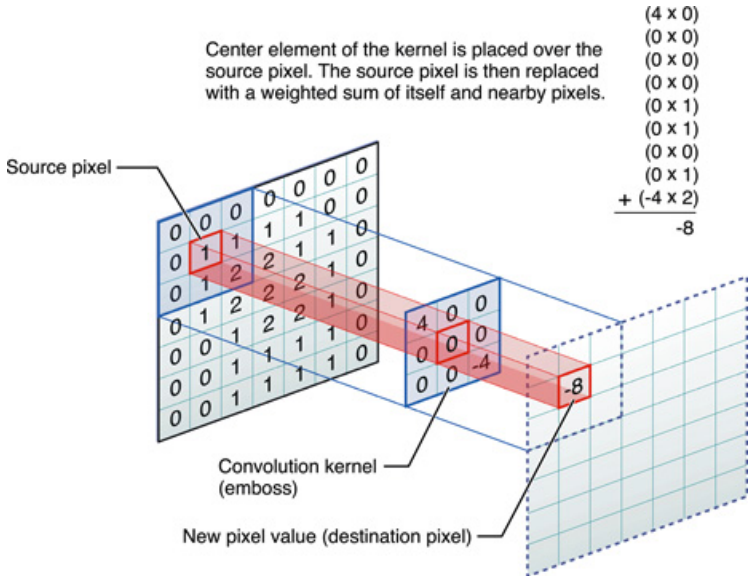


Figure 4 Convolution process

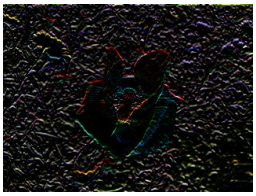
Varying the composition of the kernel, it is possible to obtain different results. Figure 5 shows different image convolution results.

Original picture

Blur

Motion blur

Finding edges



Sharpen

Figure 5 Convolution operations with different kernel matrix.

It is also possible to have more complex functions than the simple *add* in order to achieve different results.

The convolution process complexity depends on the original matrix size, the kernel dimension and also on the mathematical function applied to the cells. By this reason, its parallelization is an important technique to take into account in further implementations.

Project materials

To develop this project, you can create your own test-bed. However, in order to compare the performance and the results obtained from different groups solutions, we will provide the test-bed that you have to pass for providing your results and elaborate the reports. The material you have for this project is as follows:

- **Serial code**: Source code for a serial implementation. This serial implementation allows you to process small and large images. For large images (>500MB) you will have problems allocating enough memory for processing convolution. If you want to test for images bigger than the images provided for us, then you would need to split the original images into partitions. In such a case, please ask us for a serial code, which allows to do partitions of original images.
- **PPM images**: In folder "*/share/apps/files/convolution/images*" you can find the set of images to test. Notice that some of these images are so large please do not copy those images to your $HOME folder. Disk space on the front-end is scarce and will be exhausted very soon.
- **Kernel matrix**: In folder "*/share/apps/files/convolution/kernel*" you can find the set of kernel matrices. Kernel matrices are described using text files. Some of the kernel provided (3x3 Edge, 5x5 Sharpen) are very useful in image processing area; the big ones (25x25, 49x49, 99x99) are randomly generated for testing the scalability of your proposals. You can copy them to your $HOME folder.
- The serial code generates a text file with the time spent in different execution phases and a PPM file with the resulting image. Again, you can store the resulting images in your $HOME but your disk quota will be exceeded quickly. By this, we strongly recommend saving the resulting images to the execution node. The resulting images can be stored locally to the execution node using the folder: "*/state/partition1/<imagename.ppm>*". **Remember to delete them**, once you finish or transfer this to your laptop by means of *sftp* command.

Source code compilation and execution

For the source code compilation follow the next instructions:

```
$gcc convolution.c –o serialconv
```

Remember that is **totally forbidden to execute the code in the front-end** of the cluster. 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 (only serial processes) and execute it follow the next instructions:

```
$qlogin -q high.q -pe smp 4
user@compute-0-X$
```

Notice that your prompt has changed to the assigned machine.

```
user@compute-0-X$ cd <hpc_project_folder>
user@compute-0-X hpc_project_folder]$ ./serialconv
/share/apps/files/convolution/images/im03.ppm
 /share/apps/files/convolution/kernel/kernel3x3_Edge.txt
/state/partition1/prova.ppm >
$HOME/hpc_project_folder/results/textfilename.txt
```

For executing the OpenMP and MPI implementation, you must use the script provided into the Benchmarking activity.

Project Statement

Section 1: Implementation with OpenMP (Deadline 9th of May)

In order to obtain the best performance of an application implemented with a parallel programming model, firstly, we should start by optimizing the application at node-level. According to this, the first section will focus on studying the operation of the sequential version, analysing the pieces of code liable 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 strategy used to parallelize the serial code.
- Check the performance obtained using different thread scheduling policies (static, dynamic, guided) and chunk sizes and discuss about the results. It is not needed to test for all the test-bed of images.
- Analyze scalability and speedup in relation to the number of threads and size problem. Consider on your discussion the effects of the dependences between iterations, in case it was needed, in your solution.

Section 2: Implementation with MPI (Deadline 30th of May)

In this section, we will focus on the implementation using the MPI programming model. You can choose to develop one of the following two different versions, which work as follows:

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.

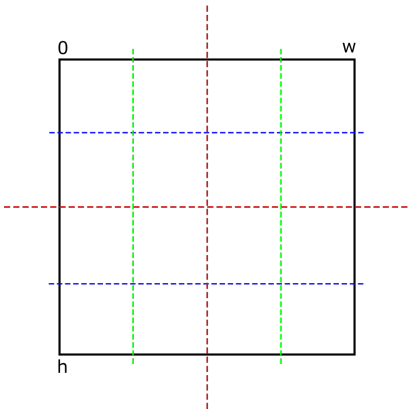


Figura 2 Static problem partitioning

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.

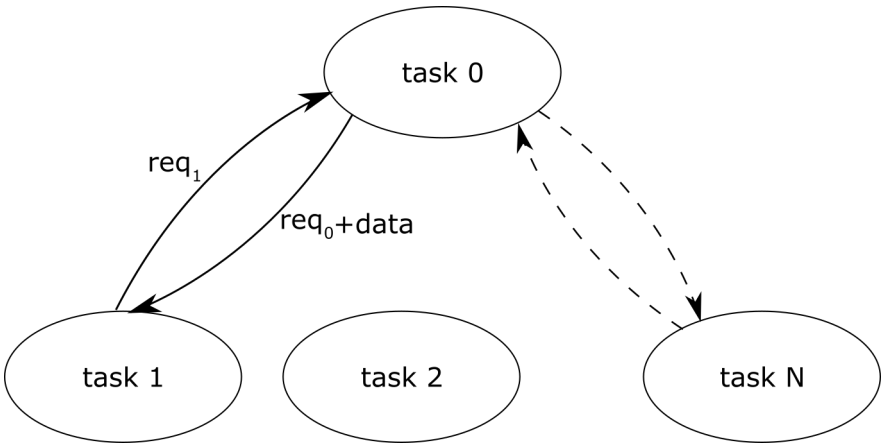


Figura 3 Dynamic problem partitioning

For the chosen strategy, you must to analyse the performance of each implementation at the following points:

- Speedup compared to the number of cores.
- Identify if there is imbalanced.
- Quantify the additional cost (overhead) of parallelization.

You can optionally develop both strategies. Obviously, we will assess it positively. In such a case, explain the strengths and weaknesses of each implementation and discuss some scenarios where you think it would be more convenient to use and why. Consider on discussion the effects of heterogeneity in the execution nodes.

In order to measure the time in the MPI programs, we recommend using the *MPI_Wtime* function, as the flowing example shows:

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

Section 3: Hybrid Implementation MPI+OpenMP (Deadline 6th of June)

In this Section, you should propose the final implementation, which combines both programming models, MPI and OpenMP.

You should compare the performance of the hybrid solution in relation to the solution implemented in the previous Sections for different problem and processes sizes.

Section 4: And the winner is…? (Deadline 18th of June)

In this section, you have to assess the quality of your proposals with respect a reference solution made by other classroom group.

For this study, we will provide the solutions proposed by your classmates. You will analyse the performance of the new solution regarding the points indicated in previous Sections. After this, you will identify the best solution and justify why. Indicate whether it is 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 your own work. Note that you can choose between to do this comparison or the supercomputer work.**