Universitat Pompeu Fabra Barcelona

Click to show/hide PDF Layer

Lab 2

Pablo Arias*, Ricard Borrell Pol†, Sergi Laut Turón‡, Pavel Pratyush§, Daniel Santos-Oliván#

Instructions

This Lab consists of four problems with several questions each. You must compile your answers to the exercises in a report, explaining how you solved the problem and answering the questions. This lab assignment is worth 15% of the total grade of the subject, and the deadline for it is May 12th at 23:59.

Each group must submit a compressed file named lab2_TxGyy.zip, where TxGyy is your group identifier. A .tar or a .tgz file is also accepted (e.g., lab2_T2G1.zip or lab2_T2G21.zip). The compressed file **must** contains three folders (1_communications, 2_def_integral, 3_indef_integral, 4_integral2D) and **all** the requested files with the following structure. Additional files will not be considered and will not be penalized.

```
lab2_TxGyy.zip ─┬─ 1_communication ─┬─ mpi_comms.c
                │                   ├─ Makefile
                │                   └─ job.sh
                ├─ 2_def_integral ──┬─ def_integral_seq.c
                │                   ├─ def_integral_par.c
                │                   ├─ Makefile
                │                   └─ job.sh
                ├─ 3_indef_integral ┬─ indef_integral_seq.c
                │                   ├─ indef_integral_par.c
                │                   ├─ Makefile
                │                   └─ job.sh
                └─ 4_integral2D ────┬─ integral2D_par.c
                                    ├─ Makefile
                                    └─ job.sh
```

A sample file named lab2_TxGyy.zip containing the reference codes has been published in Aula Global. You need to create your job scripts to perform your tests according to what is asked. The Makefiles should not be modified unless the exercise says so. Focus on the code and the work requested for each exercise.

If you have any questions, please post them on the lab class forum in Aula Global. However, do not post your code in the Forum. If you have other questions regarding the assignment that you consider cannot be posted in the Forum (e.g., personal matters or code), please contact the lab's responsible person at daniel.santos@upf.edu.

* pablo.arias@upf.edu.
† ricard.borrell@upf.edu.
‡ sergi.laut@upf.edu.
§ pratyush@upf.edu.
# daniel.santos@upf.edu.

Criteria

The codes will be tested and evaluated on the same cluster where you work. The maximum grade on each part will only be given to these exercises that solve in the most specific way and tackle all the functionalities and work requested. All the following criteria will be applied while reviewing your labs in the cluster.

Exercises that will not be evaluated:

- A code that does not compile.
- A code giving wrong results.
- A code that does not adhere to all the input/output requests.
- lab2_TxGyy.zip delivered files not structured or named as described previously.

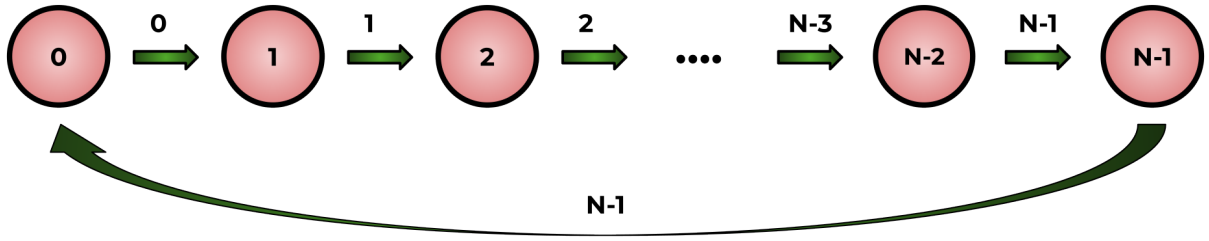
Exercises with penalty:

- A code with warnings in the compilation.

1. Communication basics

In this exercise, we are going to practice the basics of MPI communications in its different types.

To do this, each of the processors will communicate its rank number to the following processor like this:



We will test this using synchronous and asynchronous point-to-point communication and compare the performance between the two in a scenario in which we have added a specific sleep that models the case in which some processors need to perform some task after sending the information.

To compare synchronous and asynchronous communications, we will measure the time spent by each processor during this operation and compute its average, minimum, and maximum time. These simple statistics will be done using collective communication among all processors.

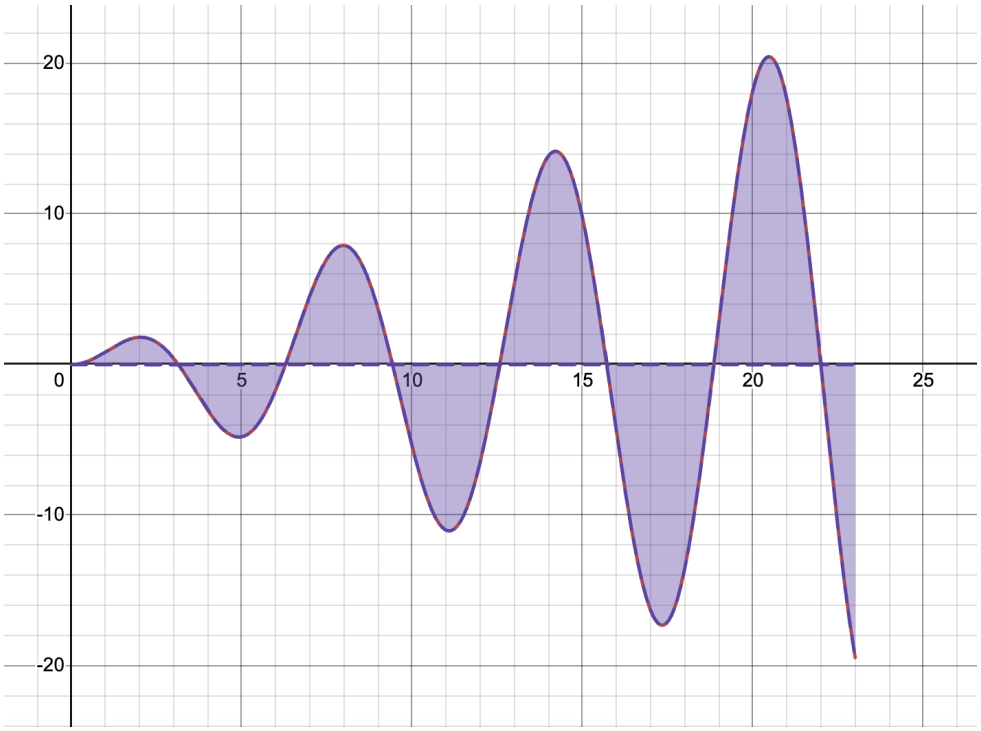
A file called **mpi_comms.c** is provided with the main skeleton of the exercise. The parts marked with a **TODO** need to be fulfilled to complete the exercise.

Report Questions 1 — Communication (15%)

1. Complete the Send/Receive blocks in the **test_synchronous** function and the correct computation of the ranks to whom send/receive.
2. Use collective communication routines to compute the statistics required for the computing time.
3. Complete the Send/Receive blocks in the **test_asynchronous** function. Analyze and comment on the differences you observe between this and the first question.

2. Numerical integration: definite

In this exercise, we will compute a definite integral numerically as it was done in the OpenMP lab. A definite integral can represent the area between a curve $f(x)$ and the coordinate axis as can be seen in the following image:



In this case the image is representing $f(x) = x \cdot \sin(x)$ and the shaded area is just:

$$\int_0^L x \cdot \sin(x) = \sin(L) - L\cos(L)$$

There are several algorithms to compute definite integrals and we are going to use the simplest one, the midpoint rule, in which the x domain is divided into N domains (let's say from 1 to N). In this sense, we label the separation between domains from 0 to N so the domain i will be the one between the nodes: $[x_{i-1}, x_i]$. For simplicity, we assume that all of the domains are equal $[x_{i-1}, x_i] = \Delta x \quad \forall i$. With all of this, we can write the midpoint rule as:

$$\int_0^L f(x) = \sum_{i=1}^{N} f\left(\frac{x_i - x_{i-1}}{2}\right) \Delta x$$

In the sample files, there is a version **def_integral_seq.c** that provides a sequential version of the algorithm. This program takes two arguments, the total number of domains in which we divide x and the value of L.

For this exercise, you need to create another file called **def_integral_par.c** that from the input/output works exactly as the sequential one but it is parallelized with MPI.

Report Questions 2

Definite integral 15%

1. Analyze the sequential version and explain how can be parallelized.
2. Create a parallel version of the program in a file called def_integral_par.c. Notice that you need to respect both the input and the output of the program.
3. Analyze how the error in the integration varies for large values of the computational domain L $>$ 1000 and the need for a large number of points.
4. Analyze the strong scaling for N=$\mathbf{10^9}$ and L=1000.

3. Numerical integration: indefinite

In this exercise, we upgrade the program that we have done in the previous section to incorporate indefinite integrals. This can be seen as the functions whose derivative is the original function, the reason why they are sometimes called antiderivative. We can define them as:

$$F(x) = \int_0^x f(t)dt$$

For example, in the case that we did for the previous case we can write:

$$F(x) = \int_0^x t \cdot \sin(t)dt = \sin(x) - x\cos(x)$$

To solve this numerically, we will suppose that our function F(x) is discretized in our nodes x_i with $i = [0, N]$. Using the midpoint rule, for each node, the indefinite integral will be given by:

$$F(x_j) = \int_0^{x_j} f(t)dt = \sum_{i=1}^{j} f\left(\frac{x_i - x_{i-1}}{2}\right)\Delta x$$

.

In the sample files, there is a version **indef_integral_seq.c** that provides a sequential version of the algorithm. The execution of this file generates output files, one called **indef_integral_seq.info** and**indef_integral_seq.txt**. The first has information about the run and the second has the solution data in binary format. A Python script to visualize the solution generated has been included and can be called as: `python plot_indef_integral.py indef_integral_seq.info`

The goal of this exercise is to generate a parallel version of this algorithm and output using the MPI output methods. To achieve that, we need to consider the following points:

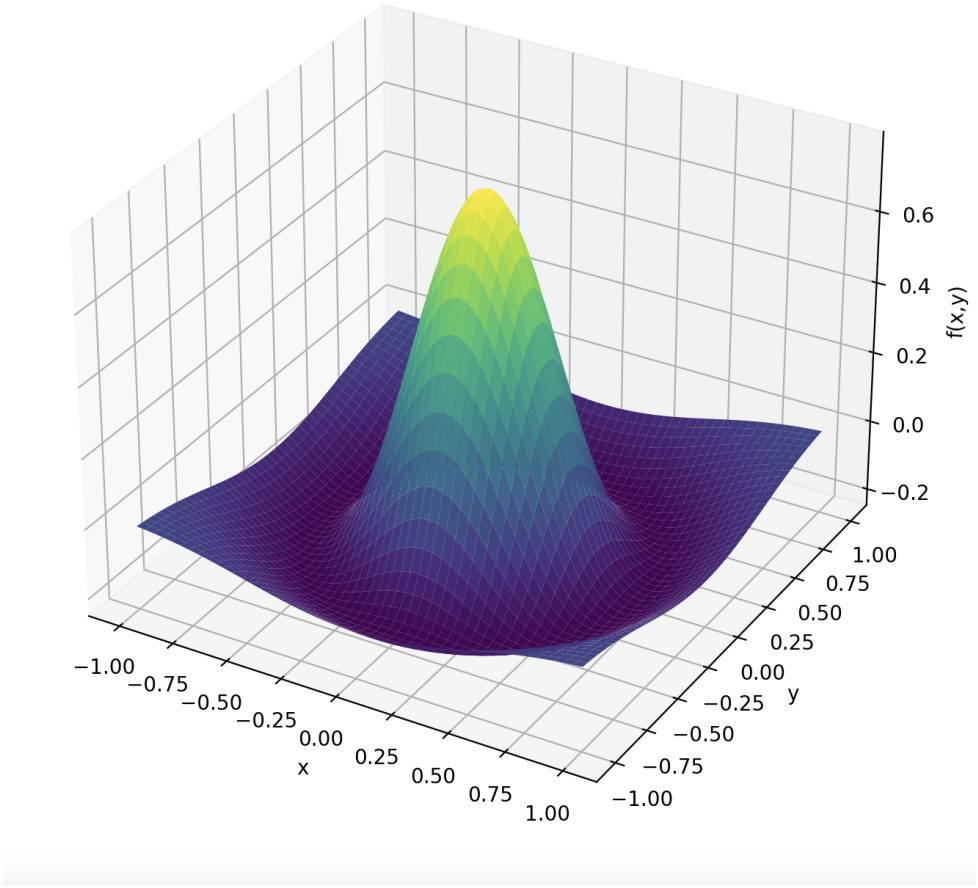
- The array $F(x_i)$ needs to be properly distributed among all the ranks.
- According to the expression given for $F(x_j)$, the value for a given index j depends on all the previous so parallelism seems impossible, study how to break this dependency.
- The output generated by the program using the MPI output needs to be compatible with the python program included. The files generated need to be called **indef_integral_mpi.info** and **indef_integral_mpi.dat**.

Report Questions 3 — Indefinite integral 30%

1. Analyze the sequential version and explain the strategy to parallelize it.
2. Create a parallel version of the program in a file called indef_integral_par.c. Notice that you need to respect both the input and the output of the program. Include plots generate with the python program for different input parameters.
3. Analyze the strong scaling for N=$\mathbf{10^8}$ and L=1000.
4. What differences do you observe with the definite integral exercise?

4. Hybrid programming: integration of a 2D surface

As in the previous exercises, we use the integral of $f(x)$ to compute the area between the function and the axis, we equivalently use the integral of $f(x, y)$ to compute the volume between the surface defined by $f(x, y)$ and the XY plane.



In this case, we will perform an integral in two dimensions and then, parallelize one dimension using MPI and the other with OpenMP in a hybrid approach.

The data to integrate, the function $f(x, y)$ is given through a data file that needs to be loaded in parallel using the MPI parallel I/O. The data is given in a list of double numbers of total size $Nx \times Ny$. Think carefully about how are going to distribute all the data in the different processors. The data is in the range [-1,1] for both x and y and it is computed assuming that we have a homogenous square grid and the given points are in the center of the squares. The binary file containing the data needs to be read by multiple processors using the MPI parallel I/O. Each processor will read a segment of the file corresponding to its part of the computational domain.

In two dimensions, using the midpoint rule, the integral of $f(x, y)$ over a domain $[-1, 1] \times [-1, 1]$ is approximated by dividing the domain into $Nx \times Ny$ subrectangles, each of size $\Delta x \times \Delta y$. The function $f(x, y)$ is evaluated at the midpoint of each subrectangle (x_i, y_j), where x_i and y_j are the midpoints of the intervals in the

x and y directions, respectively. The approximate integral is then given by:

$$Volume \approx \sum_{i=0}^{N_x-1} \sum_{j=0}^{N_y-1} f(x_i, y_j) \cdot \Delta x \cdot \Delta y$$

The calculation of the integral is parallelized using MPI in conjunction with OpenMP to leverage both inter-node and intra-node parallelism. MPI is used to distribute the data in one of the dimensions and distribute computations to multiple computing nodes, where each node handles a portion of the 2D grid. Inside each node, OpenMP is utilized to further parallelize the computation among multiple cores or threads. This dual-level parallelism significantly speeds up the integration process and allows for efficient scaling on high-performance computing systems.

A file called **integral2D_par.c** has been provided with the skeleton of the program that you need to complete in which the key parts to solve are marked. Multiple data files are provided with different grid sizes. A tar file with all the data files is stored in the cluster folder `/tmp/IPPD/Seminars/lab2_data_ex4.tar.gz`. Remember that this can be extracted as: `tar xvf lab2_data_ex4.tar.gz`.

Report Questions 2 — Definite integral 30%

1. Start with the MPI parallelization. Explain what is the strategy to parallelize, especially how the data is going to be shared among ranks.
2. Complete the parallelization using OpenMP to parallelize the other dimension. Explain how you have proceeded.
3. Analyze and comment on the scaling for different values of MPI tasks and with/without OpenMP.