PROJECT HPC : Parallel Resolution of an Equivalent 2D Laplacian Problem

BE Part II : In Part 1 we have practiced some basics MPI routines, useful to deploy a parallel program and to exchange data (namely 'messages') between the processes involved. We're now willing to perform the parallel resolution of a discretized (2D) Laplacian Problem.

The BE is organized as follow :
1/ description of the continuous problem and the numerical method that will be led to reach the approximate numerical solution of the continuous problem. The parallel algorithm is then describe, including the manner how the global domain is split on each processes, and the consequence on the communication pattern between them.
2/ description of the sources of the program. **Few modifications will have to be made in the source file (only for one file)**. After modifications are done, several run will be led with small size domain. Solution on each domain (according to the number of processes involved) will be display. Then some performances analysis will be done.

Download from CALMIP website :
https://www.calmip.univ-toulouse.fr/spip.php?article387
`MP2Physique_SUJET_PROJET_HPC_2018_2019.pdf`
#connect to azteca
`ssh –X votrelogin@azteca.univ-tlse3.fr`
#copy archive in your account on Azteca
cp `/users/calmip/rnn3209a/PROJECT_MP2_PhysFonda_2018_2019.tar .`
`scp PROJECT_MP2_PhysFonda_2017_2018.tar votrelogin@azteca.univ-tlse3.fr:~/.`
#expand archive
`tar xvf PROJECT_MP2_PhysFonda_2018_2019.tar`
Go in the proper location with command
`cd PROJECT_MP2_PhysFonda_2018_2019`
environment
`source env.sh`

In the following, sentences after character '#' describe the following command that have to be done.
Remark : all files are written in FORTRAN language. When 'uncomment the line' is asked, that means that you have to erase the 2 first character of the line involved : '!!'

1 The problem

1.1 Continuous problem

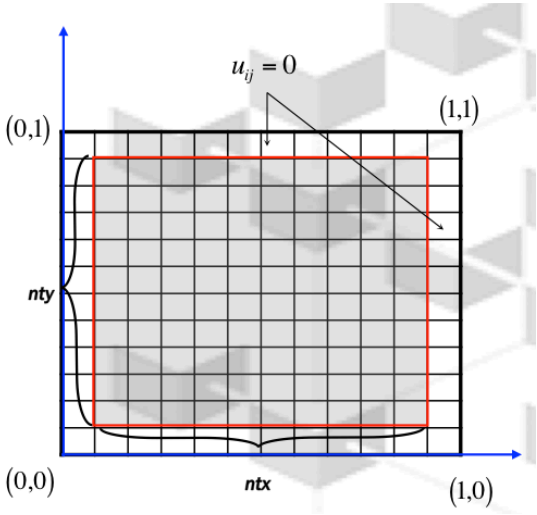
We're dealing with the following problem (Poisson problem) :

$$\Delta u(x,y) = f(x,y) \text{ in } \Omega = [0,1]\times[0,1]$$
$$u(x,y) = 0 \text{ on } \partial\Omega$$
$$f(x,y) = 2\cdot\left(x^2 - x + y^2 - y\right)$$
$$u_{exact} = xy(x-1)(y-1)$$

Notice that this is not a time evolution problem. Time variable is not involved.

1.2 Domain discretization and Numerical method

The domain is discretized as follow :



Components of vector (u_{ij}) are the unknowns of the discretized problem, where

$x_i = ih \quad ; \quad i = 0,....,ntx+1$

$y_i = ih \quad ; \quad i = 0,....,nty+1$

$ntx = nty$: number of inner points

$$h = \frac{1}{(ntx+1)}$$

To solve the discretized problem we will perform an Iterative Method (Jacobi Method, $o(h^2)$) , that is, we will build a sequence of vectors $\left(u_{ij}^n\right)_{n\in IN}$ that will converge to the solution of the numerically approximate problem. We present below the Jacobi Iteration that build $\left(u_{ij}^{n+1}\right)$ from $\left(u_{ij}^n\right)$ the previous computed term of the sequence :

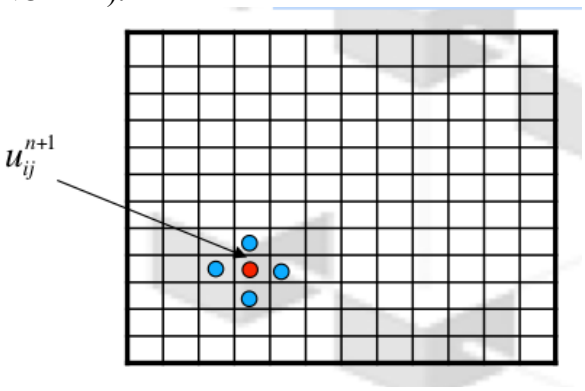
$$u_{ij}^{n+1} = c_0\left(c_1\left(u_{i+1\,j}^{n} + u_{i-1\,j}^{n}\right) + c_2\left(u_{ij+1}^{n} + u_{ij-1}^{n}\right) - f_{ij}\right)$$

where

$$c_0 = \frac{h^2}{4}$$

$$c_1 = c_2 = \frac{1}{h^2}$$

This jacobi Method is based on a five point stencil (neighbors EAST,WEST, SOUTH and NORTH):



1.3 Parallelisation

Each process 'p' will lead the jacobi iteration method on a sub domain Ω_p. This parallelisation strategy is called SPMD, Single Program Multiple Data. It allows to achieve a equilibrium in the amount of computation between all the processes. In the example below, global discretized domain Ω is splitted in 4 parts, namely sub-domains. The size of these domain is equal to $\left(\frac{nty}{NP}\right)\times\left(ntx\right)$, NP : number of processes.

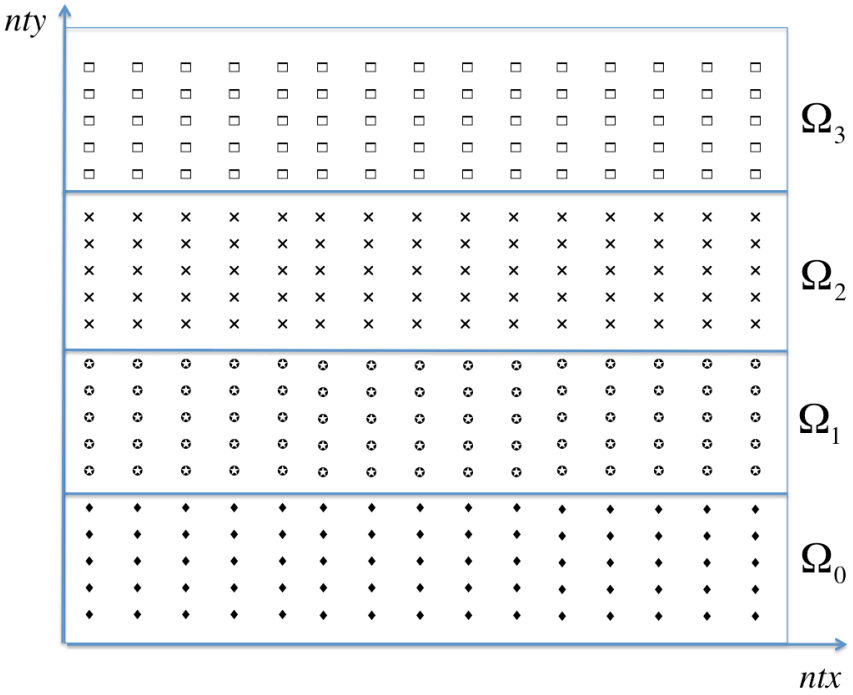


Figure 1 : Decomposition in 4 sub-domain

We describe below the algorithm. In practice we define a threshold (a very small scalar). Above this threshold we consider that convergency haven't been reached yet. For a very small value, the number of iterations to converge may be very high and the method can be very time consuming.

Parallel algorithm:

For each processes:

While global error > threshold
    For discretized point (i,j) of sub-domain Ω_p
        Send data to neighbor
        Recieve data to neighbor
        Perform jacobi iteration
    End for all points (i,j) of sub-domain Ω_p
    Compute new global error
    n=n+1
End While test

1.4 Communication pattern between processes

for a given process p :

Ω_p : is sub-domain associated to process p

Ω_{p+1} : is sub-domain associated to process $p+1, north-neighbor$

Ω_{p-1} : is sub-domain associated to process $p-1, south-neighbor$

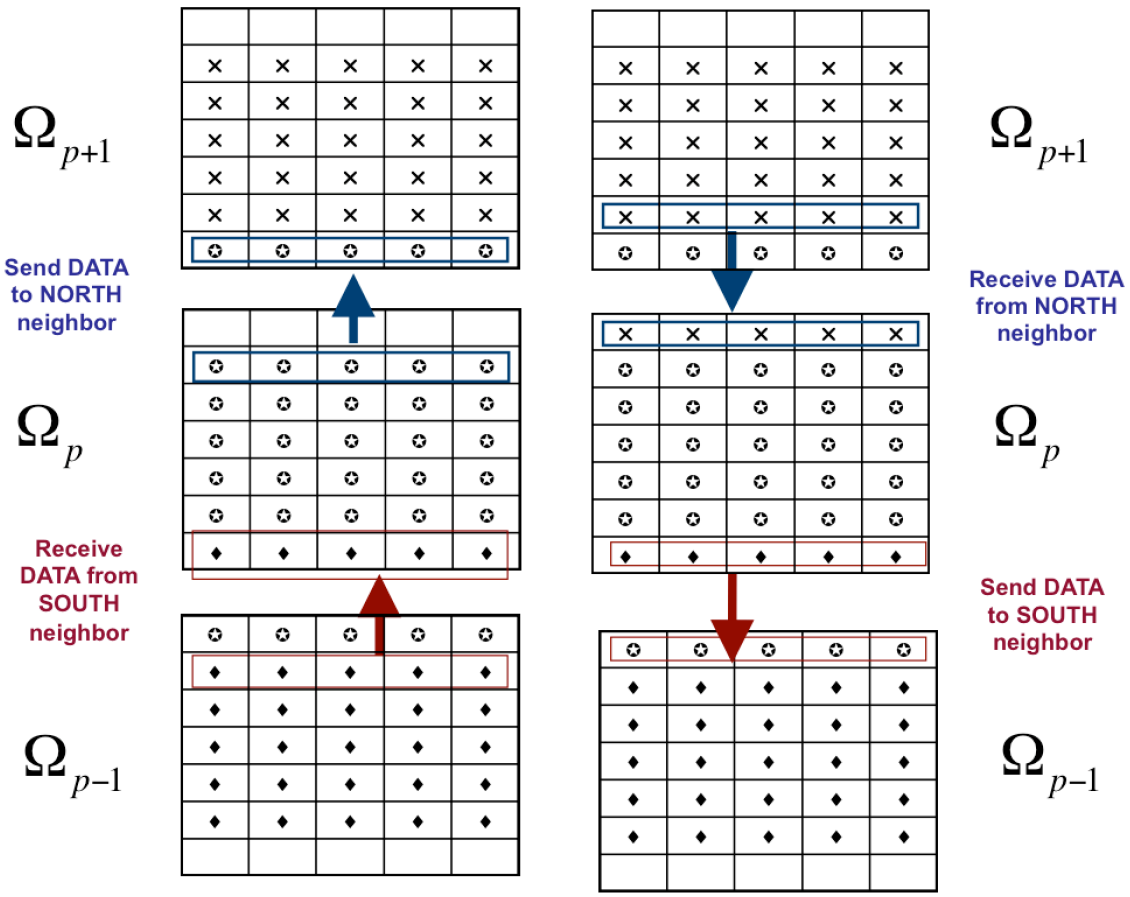


Figure 2 : Communication pattern

Remark: with 4 processes for instance, note that process 3 has no NORTH neighbor, and process 0 has no SOUTH neighbor. This will be automatically be handled by MPI routine that will be used (MPI_CART_CREATE), which will compute properly each neighbor list for each processes.

For the sake of simplicity in terms of programming, ‘ghost’ point are added to data structures (2D arrays) that store the value of the unknowns corresponding to a given domain, of a given process p. You can see in the figure above ‘blank’ lines (left), which will receive data from North or South neighbor.

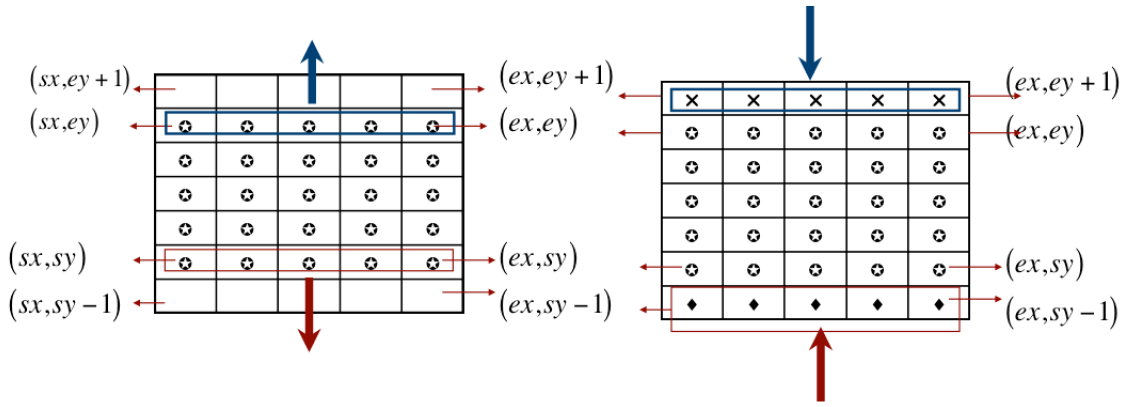


Figure 3 : Data structure. Definition of the bounds of the 2D-array which stores unknowns of the problem. *start x : ex, start y : sy ; end x : ex, end y : ey*. Bounds sy and ey will change according to the rank of the MPI process (see poisson.f90), in a manner similar to we have seen in the practice 'compute_pi'.

These 'ghost' or 'blank' locations are very useful to receive and store the data sent by neighbors (only south and west in this case).

2 Parallel computation

In the section we describe all runs that will have to be done and we ask some questions. All the answers will be put in your report.

2.1 Computer configuration

How many cores are available in the computer you are currently using ? (use command : numactl --hardware)

How many processors (or 'socket' or 'node')? (use command : numactl --hardware)

What is operating system currently running ? (Windows, MacOSX, UNIX, Linux)

[OPTIONAL]
What is the interconnection topology between 'nodes' ? (use command : numactl –hardware, look at 'node distance' matrix) choose between : ring, hyper-cube, all-to-all
What is the performance in Gflop/s of machine Azteca ? [to help : the frequency is 2 Ghz ; each core hold 2 FPU, vector capacity of each FPU is 2 (in double precision namely 8 bytes or 64 bit)]

2.2 Description of the different source files

poisson.f90 : main program
initialisation.f90 : initialize arrays (data structure) for vectors *u, u_new, u_exact* and source term *f*
communication.f90 : perform the communication between processes
neighbor.f90 : define for a given process its North and South neighbor [including 'NULL' neighbor, if SOUTH or North doesn't exist]
compute.f90 : compute a jacobi iteration
error.f90 : compute the error

All this files will be compiled and linked to give one single executable to be run.

2.3 tests with small Mesh and questions

Run tests in parallel :
Run different tests for different values of NTX=16 and NTX=32 (in poisson.f90) and different number of processes (=1, 2 and 4)

To run tests :

- edit and modify files poisson.f90 (Read carefully all comments and apply modification when keyword 'TO DO' is find):
    - gedit poisson.f90 &
- to compile:
    - make poisson
- to run (on 4 processes for example):
    - mpirun –np 4 poisson
- to visualize results

- gnuplot graph4

Sum-up of the commands to be performed:
#Read carefully all comments and apply modification when keyword 'TO DO' is find.
gedit poisson.f90 &
#Compile MPI executable
make clean
make poisson
 # Run in parallel for 1 process
mpirun –np 1 poisson
 # visualisation of the solution
gnuplot graph1
 # Run in parallel for 2 processes
mpirun –np 2 poisson
 # visualisation of the solution
gnuplot graph2
 # Run in parallel for: 4 processes:
mpirun –np 4 poisson
 # visualisation of the solution
gnuplot graph4

Questions about the tests :

- Verify convergency is achieved for each cases and write down the error and the number of iterations necessary in each cases (read on screen « Global error ») for different values of NTX, and different number of processes (1, 2 and 4).
- Give some comments

2.4 Performance tests and questions

In this part, time to reach solution can be very long, so ***we focus on the time spent for 10 iterations***, assuming that eventually it will converge.

Run performance tests in parallel and Questions :

Run tests with different values of NTX (in poisson.f90) : 1000, 5000, 10000, and 15000

For each value of NTX, compute and analyze speed-up for 1, 5, 10 and 20 cores [Optional 40].
What can you say about speedup evolution (scalability)? Give some comments.

To run tests :
(for each runs with couple (NTX, number of processes) make 2 or 3 tests; take time information on screen)
#edit poisson.f90 to modify NTX
gedit poisson.f90 &
#Run poisson performance.
make poisson
 # Run in parallel (for instance 5 processes)
mpirun –np 5 poisson

3 How to make the report

Hereafter follows some indications for the report you will have to send by e-mail : nicolas.renon@univ-tlse3.fr

The report should be short (3 or 4 pages as a maximum).

You will create a file and put your results in.

This file should be named as follow:

Name_SurName_HPC

The version or format of the file that you will send has to be a PDF file (but you can create it in any format you like).

The report must contain your answers and computing experiments results for question §2.1, §2.3 and §2.4.

Thank you.