Parallel Computing and Algorithms Spring 2016

Distributed HPC programming with MPI/OpenMP

Distributed Matrix Multiplication

Prof. Francois Kilchoer

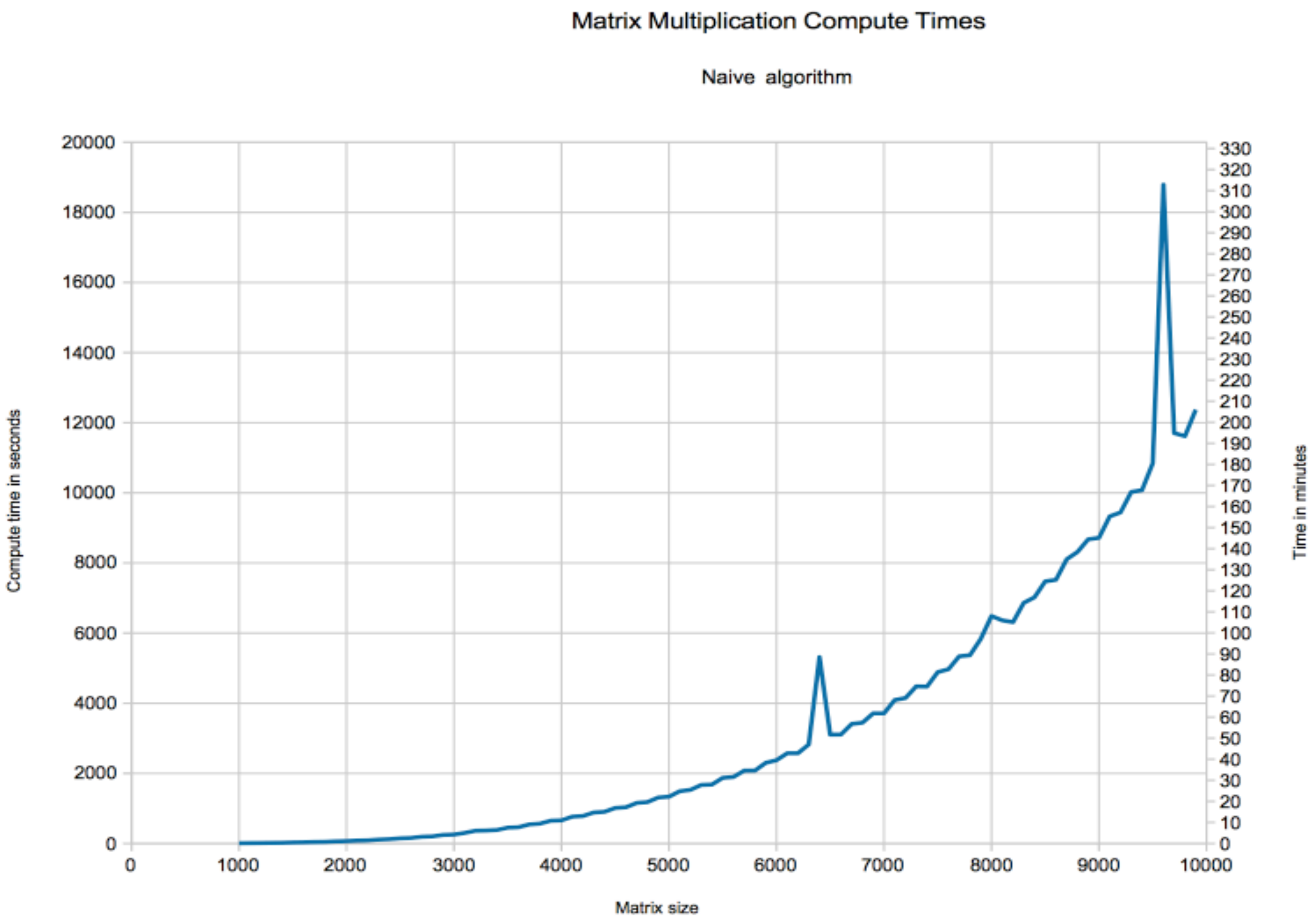
1. Performance analysis of a distributed matrix multiplication program

The objective of this exercise is to execute and analyse the performance of a parallel square matrix multiplication program written in MPI/OpenMP (combining both libraries). This program computes the following product:

$$\mathbf{A} \times \mathbf{B} = \mathbf{R}$$

where **A**, **B** and **R** are **NxN** square matrices.

A naïve program to compute this product is provided on the course web site. Times measured for this algorithm for various matrix sizes on the computing grid in Fribourg are summarized in the following chart:



You will notice 3 "spikes" in the measurements, where the algorithm takes abnormally longer times (for matrix sizes of 6400, 8000, and 9600). For the time being, I have no explanation for this (bonus for the first person who finds out why there are such peaks).

Your task will be to improve the running times for matrix multiplication using openMP and MPI. The goal is to reduce running times to less than 20% of the simple sequential program running time.

The idea of the algorithm you are to implement is as follows. The master divides matrix **A** into several blocks of lines and matrix **B** into several blocks of columns. Each worker receives one block of lines of matrix **A** and one block of columns of matrix **B**. Using this data each worker can compute one block of the result matrix **R**. This process is illustrated in the figure 1.

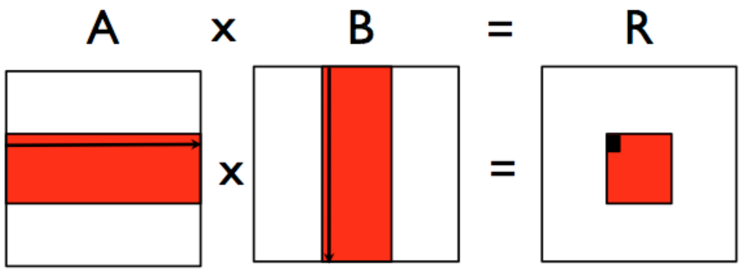


Figure 1: Distribution of the matrices

As mentioned above, each worker computes a block of the resulting matrix. To do this, each worker can use one core (only one thread of execution) or several cores (several threads of execution). The MPI/OpenMP view is that MPI distributes 1 task per physical machine, and lets OpenMP create the necessary threads on each machine (typically one per core).

You are free to split matrices A and B however you like in order to achieve the best performance.

In the rest of this document we will use the following naming convention:

- **N**: The size of matrices (square)
- **P**: The number of MPI processes
- **T**: The number of openMP threads per MPI process

These values are the independent parameters we can use to influence the computing time of the program.

Your task is to play with these values and to record computing times. Then you need to analyse and explain the obtained results.

2. Computations to run

For varying matrix sizes, you are to measure the performance of your solution using MPI point-to-point communication using in one test **buffered** mode communication, and in the other test **immediate** mode communication. For each matrix size, you need to run the test 3 times minimum to make sure that local conditions don't interfere with your measurements. The tests on the grid will be done by groups of students. For each group some values for these parameters are imposed. You must use ***at least*** these values but, of course, you can play with other values as long as you respect the constraints mentioned above. The provided programs record the following time values:

- The ***initialisation time*** = the time used by the master to create the matrices
- The ***sending time*** = the time used by the master to send data to all workers.
- The ***computing time*** = the time spent on the Master to obtain all results from the worker and to reconstruct the result matrix **R**

The times presented above are illustrated on figure 2 below.

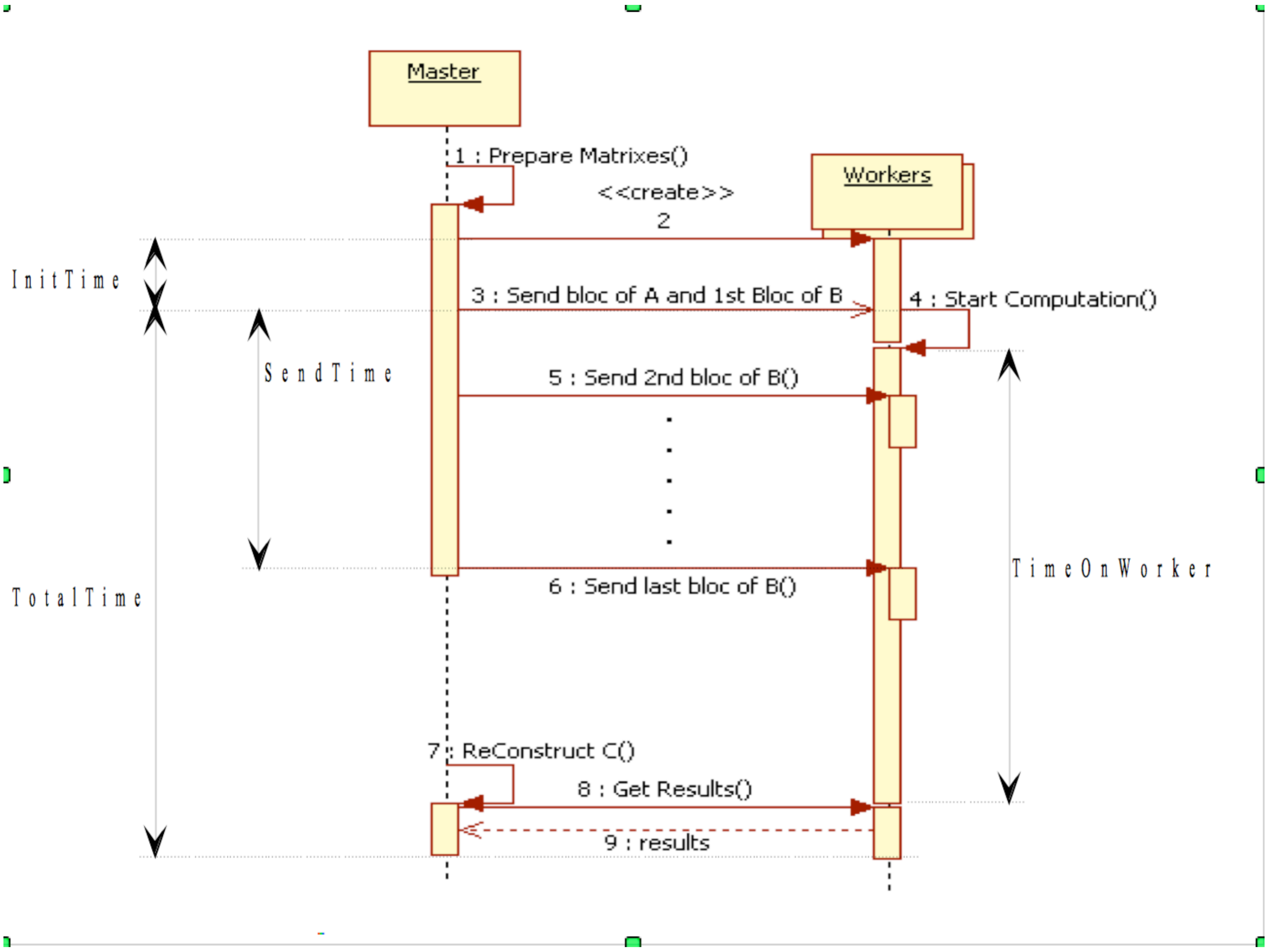


Figure 2: Sequence diagram of the execution of the POP-C++ matrix multiplication program

On Figure 2 we see that the Master sends blocks of data to a worker who, once data is received, can start computing. Ideally, when the master has finished sending data to the last worker, it will not have to wait to receive the results from the first worker.

3. Work to do

Each group will provide the results indicated below for their MPI/OpenMP version. The naïve algorithm version is given. The MPI/OpenMP version should allow the user to specify on the command line the number of openMP threads to be created by each MPI process for local processing.

For comparison purposes, the sequential compute times are provided. You also have the results of the matrix multiplication for various matrix sizes as computed by the sample program.

3.1. Overall length of computation

Compare the computation time spent by the simple algorithm with each of your two MPI/openMP programs. For this, you will use at least 10 different matrix sizes, one matrix size in each 1000-range. For example, you might choose **N** as 1900, 2800, 3700, 4600, 3500, 4600, 5500, 6400, 7300, 8200, 9100.

3.2. Computation of sending times

For each of your solutions, compare the time spent

- sending data to workers
- computing time (essentially the time spent receiving all results, as a proxy)
- total time

Fro the comparisons to do above, provide Make the following plots (for all plots X-axis=size **N** of the matrix)

- plot 1, sequential vs parallel algorithm comparison: Y-axis = compute time
- plot 2, comparison of buffered / immediate sending: Y-axis = sending time
- plot 3, comparison of buffered / immediate sending : Y-axis = total computing time

With each plot also give a table containing the measured values.

Remarks: Always indicate the units of the measured values. Use linear X and Y axis, and use the same units for all graphs.

For each plot comment, explain and justify the obtained results.
Hint:

- All results must be an average of several executions (min. 3) in order to decrease the influence of local conditions.
- Times measured by the program are LT (local time= wall clock time on the machine).

4. Technical information:

All code source is available on Moodle (`MATRIX.zip`). Do the following to install and to test the execution of the sequential matrix multiplication program:

1. Create a directory to run your tests, for example: `mkdir project`
2. Go to this directory: `cd project`
3. Copy two files to your project directory: `Makefile MStandard.c`
6. Compile the program: `make`
7. Make a test to check that everything is working: `make run`

   You should obtain something similar to the following:

```
./MStandard 100 debug
debug is now on.

Start sequential standard algorithm (size=100)...
Created Matrices A, B and C of size 100x100
A[100x100]:
1.0    1.0    1.0    1.0...
2.0    2.0    2.0    2.0...
3.0    3.0    3.0    3.0...
 .      .      .
 .      .      .
 .      .      .

B[100x100]:
1.0    1.0    1.0    1.0...
2.0    2.0    2.0    2.0...
3.0    3.0    3.0    3.0...
 .      .      .
 .      .      .
 .      .      .
-----------------------
Times (init and computing) = 0, 0.004 sec

size=100   initTime=0  computeTime=0.004 (0 min, 0 sec)
C[100x100]=A*B:
5050.0 5050.0 5050.0 ...
10100.0 10100.0 10100.0 ...
15150.0 15150.0 15150.0 ...
   .       .       .
   .       .       .
   .       .       .

....................
```

5. Information on the usage of `crontab`

To ease the automation of execution, `cron` is your friend, as it allows you to launch scripts without being logged on the machine. To do this, you will use the `crontab` command (see `man crontab`):

- `crontab -l` : show the current programmed crontab
- `crontab -e` : edit crontab

Perhaps confusingly, crontab refers both to the command, and to the file that specifies which program(s) to run at what time(s).

The `crontab` file format is:

```
# This is a comment;  next line sets the appropriate shell
SHELL=/bin/bash
min hour day month * program-to-run with-possibly-some-arguments >output-goes-here
```

EXAMPLE:

```
SHELL=/bin/bash
19 15 21 10 * source /home/project/MATRIX/MPI/scriptParallel.sh > /home/project/MATRIX/MPI/log.log
```

The effect is that at 15h19, October 21st, the script

    /home/project/MATRIX/MPI/scriptParallel.sh

will be launched, and the standard output will be written to the file `log.log`

Note:

- You must indicate the **full path name** of the script file you want to execute.
- The environment that cron will run your script in **is not the same as your logged in environment.** To fix this, you need to, at a minimum, ensure that the PATH environment variable is set up, as shown below.
- The line `SHELL=/bin/bash` is mandatory to be sure that the correct shell (`bash`) will be used.

The command `crontab -e` allows you to edit the `crontab` file and the command `crontab -l` to see the current content of the file.
For more information on the `crontab` command type: `man crontab`
In the script you use to run your programs, set your PATH environment variable so that it looks like this:

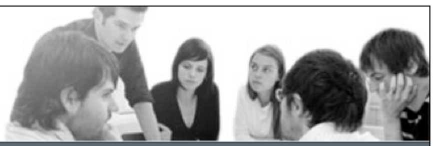
```
PATH="$HOME/bin:/opt/mpi/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:/usr/local/popc/bin:/usr/local/popj/bin:/common/mpi/bin"
```

6. Report

Each group must provide the following documents on Moodle:

1. A «pdf» version of your report (max. 20 pages!). This report should contain all your interesting results (as required in this statement), as well as a pertinent remarks and comments.
   On the first page of your report, indicate after your names, the login username(s) you have used to run your tests (see template on the next page).
2. A «zip» file containing all the result files (text format) produced by the program that you have used to draw plots of your report, as well as the program(s) you have used to implement your solution.

MASTER OF SCIENCE IN ENGINEERING



Parallel Computing and Algorithms - Spring 2016

Distributed HPC programming with MPI and OpenMP

Distributed Matrix Multiplication

Prof. Francois Kilchoer

Student Names

login name

1. Title of section 1 of your report

Section 1 text....

2. Title of section 2 of your report

Section 2 text....