Parallel and Distributed Computing

Parallel programming/computing

MPI Project Raport

Filip Mazur - 226018

Applied Computer Science
Department of Computer Science and Management
Wroclaw University of Science and Technology
2020-06-15

Contents

1. Introduction

This project concerns the preparation of a program which will use Message Passing Interface (called MPI henceforth) in order to perform multiplication via Fox Algorithm of matrices of varying sizes distributed over varying number of parallel processes and then performing various time measurements described in paragraphs below.

1.1 Project description

The project is written in C++ under Linux OS compiled using mpic++ and using Open MPI version 3.1 [1] installed via AUR. To make time measurements chrono library was used.
Due to the nature of Fox Algorithm several constraints had to be put on the project:

- Allowed number of processes for mpirun "p" is such that square root of "p" must be an integer in order to allow Fox Algorithm to distribute the matrix between nodes by forming a special "checkboard-like" grid
- Matrix size "n" must be dividable by square root of "p" in order to rule out unfair work division between processes

Some further details about the project are that data is loaded into matrices from two csv files which were generated to have 1000 by 1000 random integers between 0 and 9 and whenever we need a matrix of size lesser or equal to 1000 we just load the part that we need from the file. Similarly output of the Fox Algorithm is also saved in an output csv file and both calculation and total running times are appended into a separate csv file.
In order too run the program type into terminal:
mpirun –hostfile hostfile -np N totallynotavirus
where N is desired number of processes not greater than the number of slots present in hostfile

2. Results

2.1 Measurement method description

There are two types of time measurements being made of:

- Fox Algorithm's calculations
- Total running time which includes loading the data, calculations, verifying the results and saving the results

The calculation of difference between total running time and calculation's time was made as well. The measurements were run over varying matrices' sizes and over varying number of nodes performing the calculations. Matrices' sizes:

- 50 (maximum number of nodes = 4 due to constraints described above!)
- 100
- 200
- 500
- 1000

Number of nodes:

- 1
- 4
- 16

Tests were performed on i5-5200U CPU @ 2.20GHz

2.2 Measurement results

Table 2.1: Measurements results

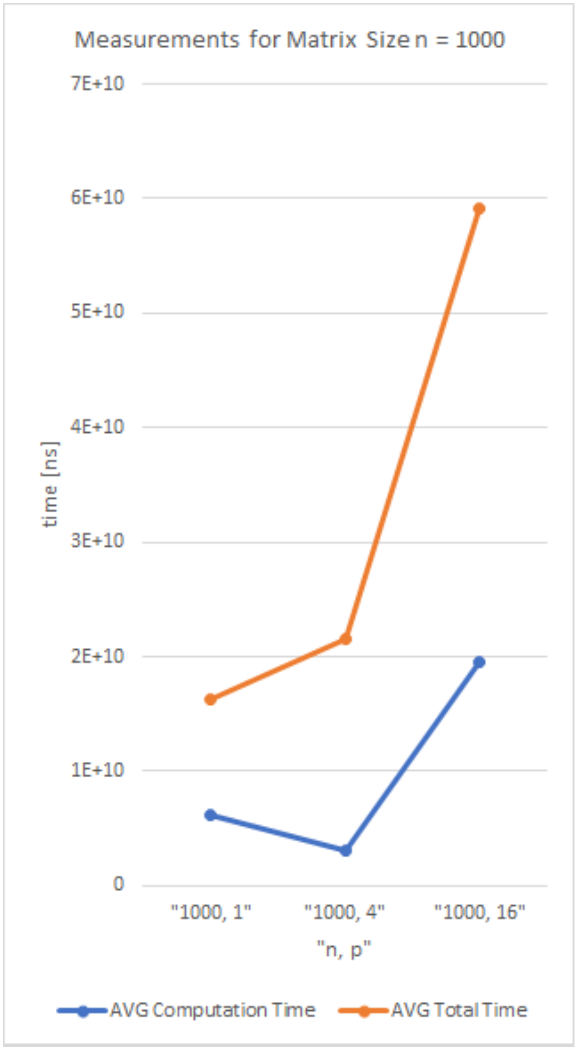
Matrix Size	umber of Nodes	*AVG Computation Time*	*AVG Total Time*	*AVG Total - AVG Computation Time*
50	1	666153.3	11263700	10597546.7
50	4	18868671.9	40278740	21410068.1
100	1	4153773	43620180	39466407
100	4	10206197	167663780	157457583
100	16	475025900	805502000	330476100
200	1	35524980	202398800	166873820
200	4	67680730	872357800	804677070
200	16	1141177000	2804238000	1663061000
500	1	539156900	1972587000	1433430100
500	4	317055500	3892990000	3575934500
500	16	5419873000	14038450000	8618577000
1000	1	6237843000	16262660000	10024817000
1000	4	3099597000	21559850000	18460253000
1000	16	19503330000	59072340000	39569010000

Disclaimer: on charts presented below "n" refers to matrix size, "p" refers to number of nodes.

.5 .5

Figure 2.1: Results in chart form p.1

Figure 2.2: Results in chart form p.2



.5

Figure 2.3: Results in chart form p.3

3. Conclusions

As we can see in the results the reduction in computation time wasn't immediately visible - it started to drop down from "n" = 500 and onward. That could mean several things, but most likely it's due to additional complexity that's derived from the necessity to support all the additional operations required to run MPI Grid. We can clearly see though that between "n" = 500 and "n" = 1000 average computation time is dropping and at a faster rate. From this we can deduce, that for small instances local multiplication is more efficient, Fox Algorithm at some point between "n" = 200 and "n" = 500 overtakes it, and then leaves it further and further behind, making it an ideal solution for really big instances.

Bibliography

[1] Multiple contributors. Open mpi.