Analysis of Speed-UP Matrix Multiplication using MPI

Imad Eddine TIBERMACINE

Abstract:

Matrix multiplication is a concept used in engineering fields such as: image processing, signal processing, graphs….etc. The complexity of matrix multiplication is O(n^3), because of that, huge matrices require a huge computation time. We , as problem solvers, our principal role is to optimise the execution time of those algorithms using different sequential and parallel algorithms. In this research, we used the open MPI method of parallel computing to evaluate the execution time under different cases.

1. Introduction:

MPI provides the user with a programming model where processes communicate with other processes by calling library routines to send and receive messages. The advantage of the MPI programming model is that the user has complete control over data distribution and process synchronization, permitting the optimization data locality and workflow distribution. The disadvantage is that existing sequential applications require a fair amount of restructuring for a parallelization based on MPI.

2. IBN-BADIS Cluster Hardware:

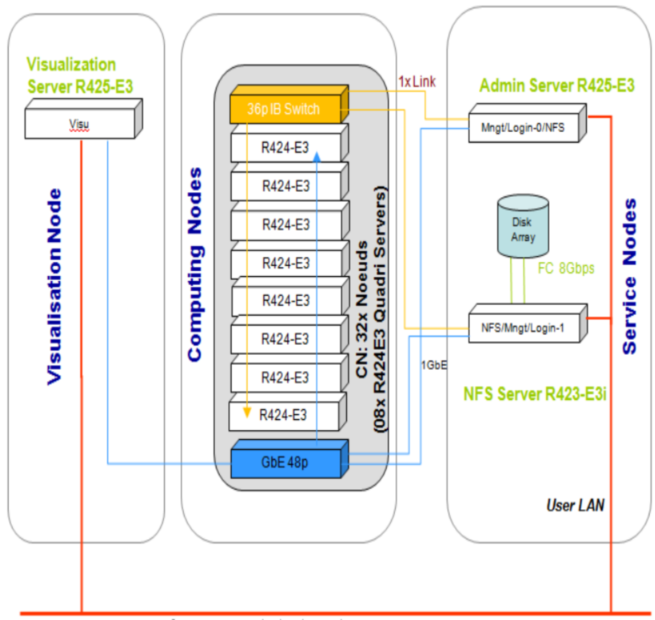
The experiments were done on Cluster of IBN-Badis (HPC of the CERIST researches center), its composed from 32 nodes, each node contains x2 processors Intel(R) Xeon(R) CPU E5-2650 2.00GHz, each processor contains 8 cores which makes 512 cores in total. The theoretical power of the cluster is around 8TFLOPS.

3. Cluster Architecture:

IBNBADIS consists of an ibnbadis0 administration node, an ibm badis10 viewer node and 32 ibnbadis11-ibnbadis42 computing nodes.

The ibnbadis10 visualization node is equipped with an Nvidia Quadro 4000 GPU (6GB, 448 cores) which can be used for calculations. Its equipped with:

- SLURM for job management
- C/C++, Fortran
- MPI and MP



4. Testing Nodes:

According to the following figure, the only nodes 38 and 39 were available at testing time, so all the tests in this sheet have been computed on the node 38 of the cluster.

```
[atibermacine@ibnbadis0 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
visu         up   infinite      1  alloc ibnbadis10
r424*        up   infinite      6 drain* ibnbadis[12,16,19,22,37,40]
r424*        up   infinite      7  down* ibnbadis[13,18,24-25,31,35,42]
r424*        up   infinite     17  alloc ibnbadis[11,14-15,17,20-21,23,26-30,32-34,36,41]
r424*        up   infinite      2   idle ibnbadis[38-39]
[atibermacine@ibnbadis0 ~]$
```

5. Steps of the analysis:

In this test, we used different sizes of matrices from 100*100 to 10000*10000, and the algorithm implemented is as follows:

- Split the first matrix row wise to split to the different processors, this is performed by the master processor.
- Broadcast the second matrix to all processors.
- Each processor performs multiplication of the partial of the first matrix and the second matrix.
- Each processor sends back the partial product to the master processor.

6. Sequential algorithm results before using MPI:

Before using the MPI, we used the sequential naive algorithm to compare the results with the parallel method later, the naive sequential algorithms is as follows:

```
for i=1 to n
        for j=1 to n
                c(i,j)=0
                for k=1 to n
                        c(i,j)=c(i,j)+a(i,k)*b(k,j)
                end
        end
end
```

I implemented this algorithm using C language and i run it on the cluster with different matrix sizes, the results of this experiments are in the following table:

Matrix Size	Execution Time in seconds
50*50	0.000001
100*100	0.000001
300*300	0.248432
1000*1000	1.965423
2000*2000	15.00390
5000*5000	205.858632
10000*10000	1711.94133

Tab.1: Execution time of the sequential algorithm for matrix multiplication with dynamic size

I tried to design the plot that defines the relation between different matrix sizes and the execution time:

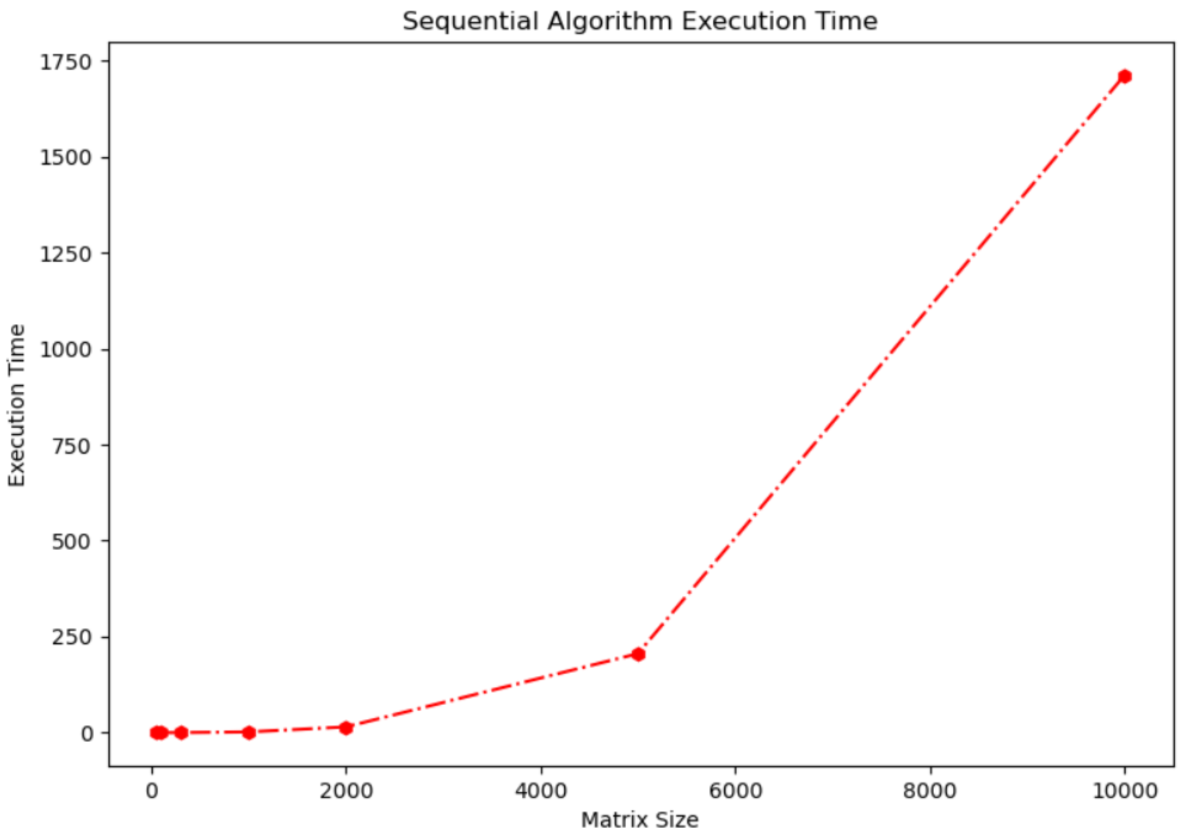


Fig.1: Execution time of different matrix sizes using sequential algorithm

7. MPI parallel algorithm results:

The following result show the execution time of different matrices sizes with different number of processors:

Processos	1000*1000	2000*2000	5000*5000	10000*10000
1	1.965423	15.00390	205.858632	1711.94133
2	0.705548	7.949866	135.201684	1076.00828
3	0.398119	4.180119	69.160852	553.358637
4	0.332966	2.975796	48.084834	400.158163
5	0.299756	2.378483	37.492173	315.647855
8	0.251118	1.931557	28.399764	221.031034
10	0.197534	1.472169	21.796179	162.124374
12	0.171343	1.247648	18.234817	142.718125
14	0.180424	1.171759	16.097543	133.333384

Tab.2: Execution time of the parallel MPI algorithm for matrix multiplication using different sizes with different number of processors

The figure below defines the graph of the previous results:

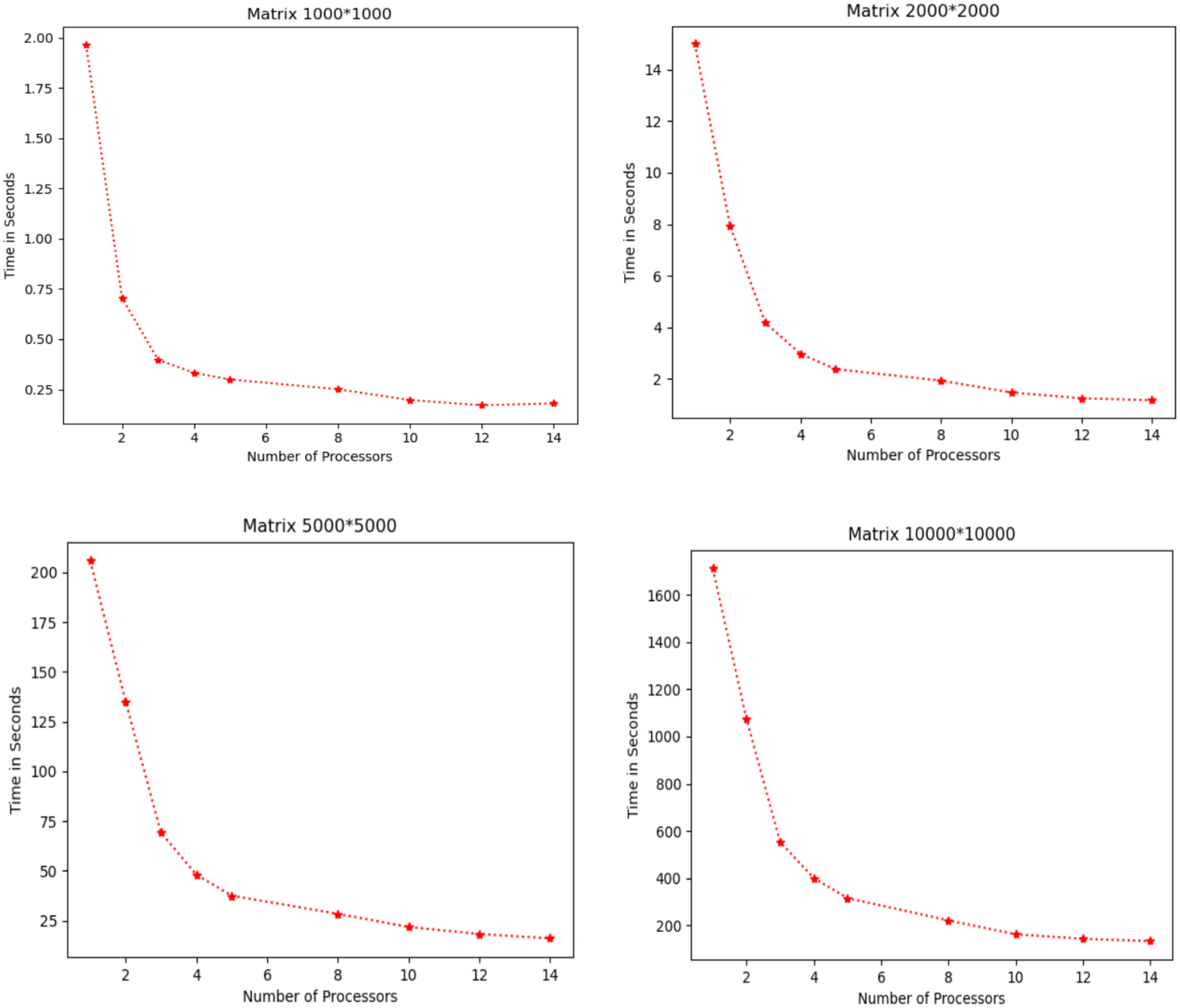


Fig.2: The execution time of different matrix sizes according to the number of processors

8. Speed UP and Efficiency:

Using the previous results, in this section I tried to calculate both of speed up and efficiency, and plot their graph. We can calculate the speed up and Efficiency using the following rules:

SpeedUP (s)= Execution time with 1 processor / Parallel Time

Efficiency = SpeedUP / Number of Processors

Processors	SpeedUP 1000*1000	Efficiency	SpeedUP 2000*2000	Efficiency
1	1	1	1	1
2	2.785668	1.392834	1.887314	0.943657
3	4.936772	1.645590	3.589347	1.196449
4	5.902773	1.475693	5.041978	1.260494
5	6.556742	1.311348	6.308180	1.261636
8	7.826691	0.978336	7.767774	0.970971
10	9.949795	0.994979	10.191696	1.019169
12	11.470693	0.955891	12.025747	1.002145
14	10.893356	0.778096	12.804595	0.914613

Tab.3.1: Speed up and efficiency calculated for the previous results

Processors	SpeedUP 5000*5000	Efficiency	SpeedUP 10000*10000	Efficiency
1	1	1	1	1
2	1.522604	0.761302	1.591011	0.795505
3	2.976519	0.992173	3.093728	1.013124
4	4.281155	1.070288	4.278161	1.062354
5	5.490709	1.098141	5.423579	1.084715
8	7.248603	0.906075	7.745253	0.968156
10	9.444711	0.944471	10.55943	1.055493
12	11.28931	0.940775	11.99526	0.999605
14	12.78820	0.913442	12.83955	0.917110

Tab.3.2: Speed up and efficiency calculated for the previous results

The figures below defines the graphs of the speedUP and Efficiency changes during the tests:

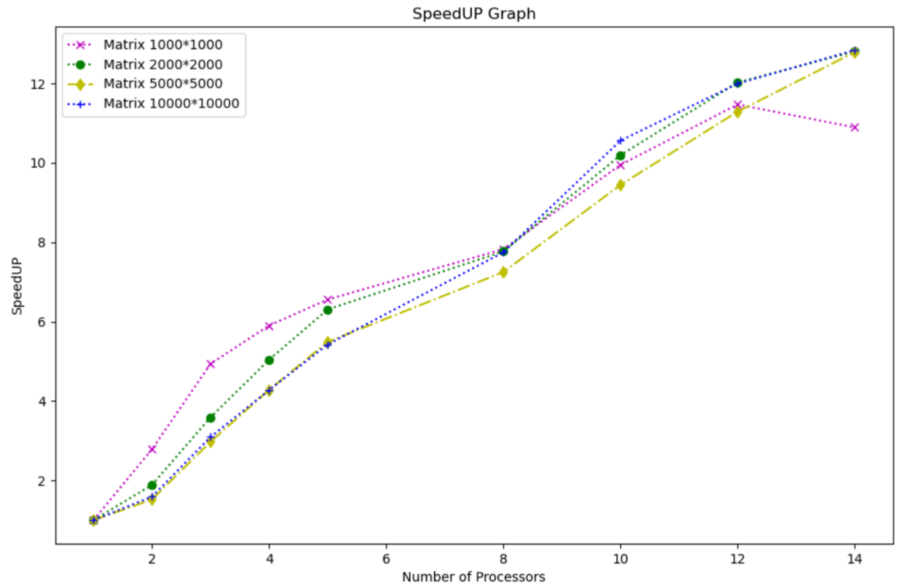


Fig.3: Graph of speedUP of the experiments

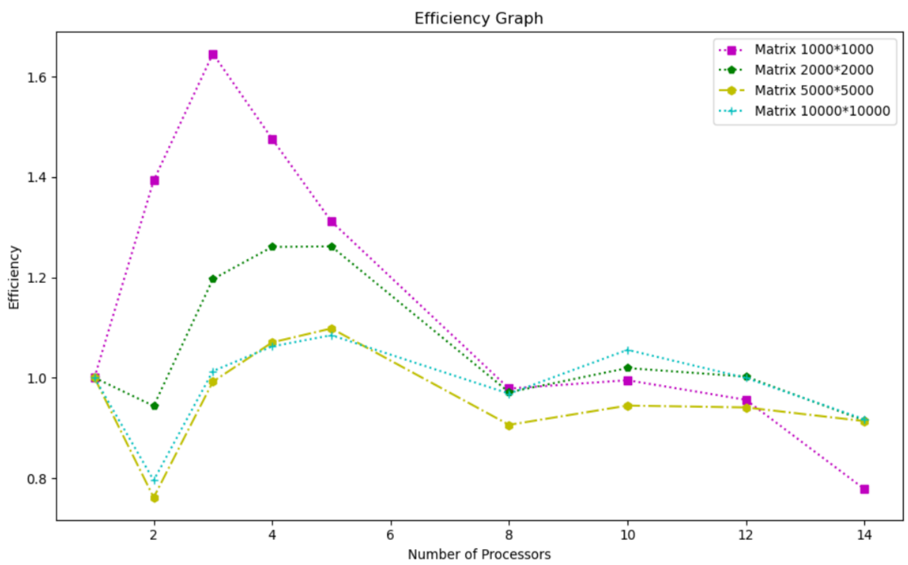


Fig.4: Graph of Efficiency of the experiments

9. Conclusion:

Based on the previous obtained results, and the plots shown above , the conclusion can be drawn is that MPI is a good method to use as an environment for parallel matrix multiplication with huge sizes, here we can increase the speedup but negatively affect the system efficiency. The second thing I can suggest is to use a hybrid parallel system to manipulate matrices of huge sizes.