ARQUITECTURAS Y PROGRAMACIÓN PARALELA
Practico cuatro

Andrei Shumak

May 2018

Contents

1 Introduction

1.1 About

This document represents forth practical task (4/4) from Architecture and Parallel Programming taught by Ricardo Javier Pérez García.

1.2 Main problem

Create an algorithm, which will multiply matrices using various technologies and compare efficiency of this technologies in terms of time. Technologies:

- Sequential: compiler and code optimizations
- Parallel: OpenMP, Pthread, MPI

1.3 Current problem

1.3.1 First part

There are 3 cases of matrices multiplication:

- A: [8000000*8] * [8*8]
- B: [8*8000000] * [8000000*8]
- C: [800*800] * [800*800]

Use MPI. Make this multiplications with 3 optimization cases:

- p1 - parallel first loop;
- p2 - parallel second loop;
- p1-2 parallel both loops;

Compare this results.

1.3.2 Second part

Then test optimization p1 for multiplications of matrices:

- [100*100][100*100]
- [200*200][200*200]
- [300*300][300*300]
- ...
- [1900*1900][1900*1900]
- [2000*2000][2000*2000]

Compare this results with sequential one. Explain, why do they differ in a such case.

2 Work

2.1 Project structure

Due to some problems with my computer a new code and new environment were used. For compiling and running `mpicc` and `mpirun` were used. The whole system was tested with Google Cloud Platform.
Operative system: Debian 9.
Processor specification

CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4
Socket(s)	1
Model name	Intel(R) Xeon(R) CPU @ 2.00GHz
Stepping	3
CPU MHz	2000.166
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	56320K

2.2 Test results

Ver2 Ofast	(p1)			(p2)			(p1-2)		
	2	4	8	2	4	8	2	4	8
A	1.549482	0.834543	0.798195	10.12342	12.43232	13.83453	1.523423	0.81234	0.752134
A	3.390034	2.234174	1.950232	3.132132	1.934234	1.723423	3.512344	2.31232	1.923423
A	1.903621	1.113111	1.019913	1.823232	1.023423	0.823412	1.923123	1.23233	0.98234

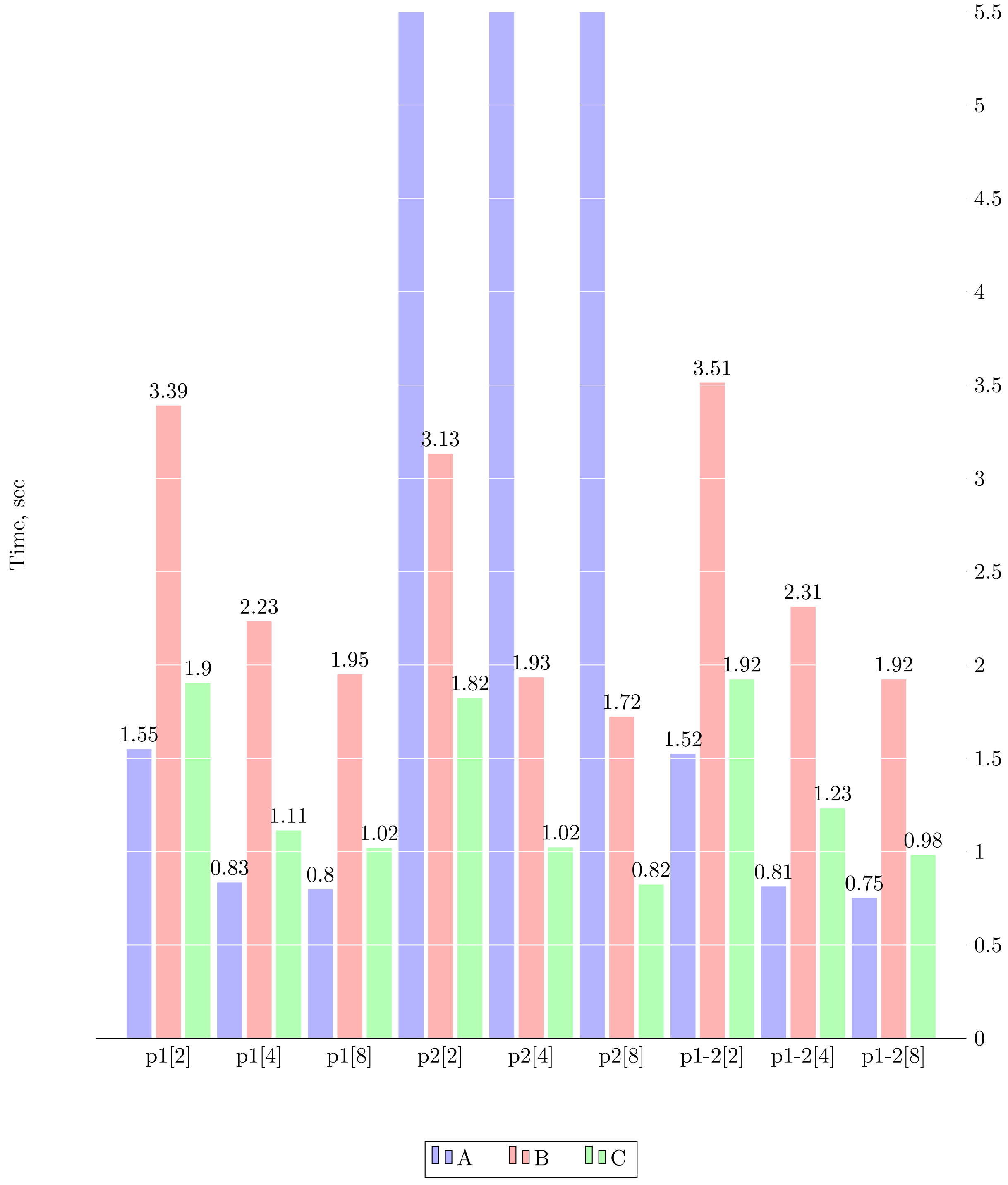

Table 1: Three different types of paralleling.

	Sequential	2 threads	4 threads	8 threads
100	0.005866	0.003407	0.002123	0.004860
200	0.048992	0.025787	0.013725	0.015675
300	0.172231	0.085344	0.044993	0.037792
400	0.404027	0.203603	0.104886	0.096556
500	0.794423	0.397927	0.207912	0.176564
600	1.391884	0.700220	0.365011	0.357141
700	2.383072	1.167209	0.703520	0.610619
800	3.545672	1.878283	1.132572	0.782668
900	6.278553	3.203405	1.822582	1.329168
1000	8.081593	4.125734	2.408056	2.095122
1100	12.240158	6.164183	3.456506	2.467258
1200	14.299724	7.297915	4.197993	3.462268
1300	20.510960	10.325189	5.950294	4.606251
1400	23.145634	11.544747	6.857423	6.264677
1500	31.716433	15.677286	9.267619	7.489619
1600	35.679971	17.707821	11.033760	10.192902
1700	47.705752	22.838477	14.026069	11.626754
1800	53.909055	26.568582	15.910368	13.985256
1900	73.711873	36.823495	21.503636	16.495144
2000	76.483606	38.491143	23.434931	20.209460

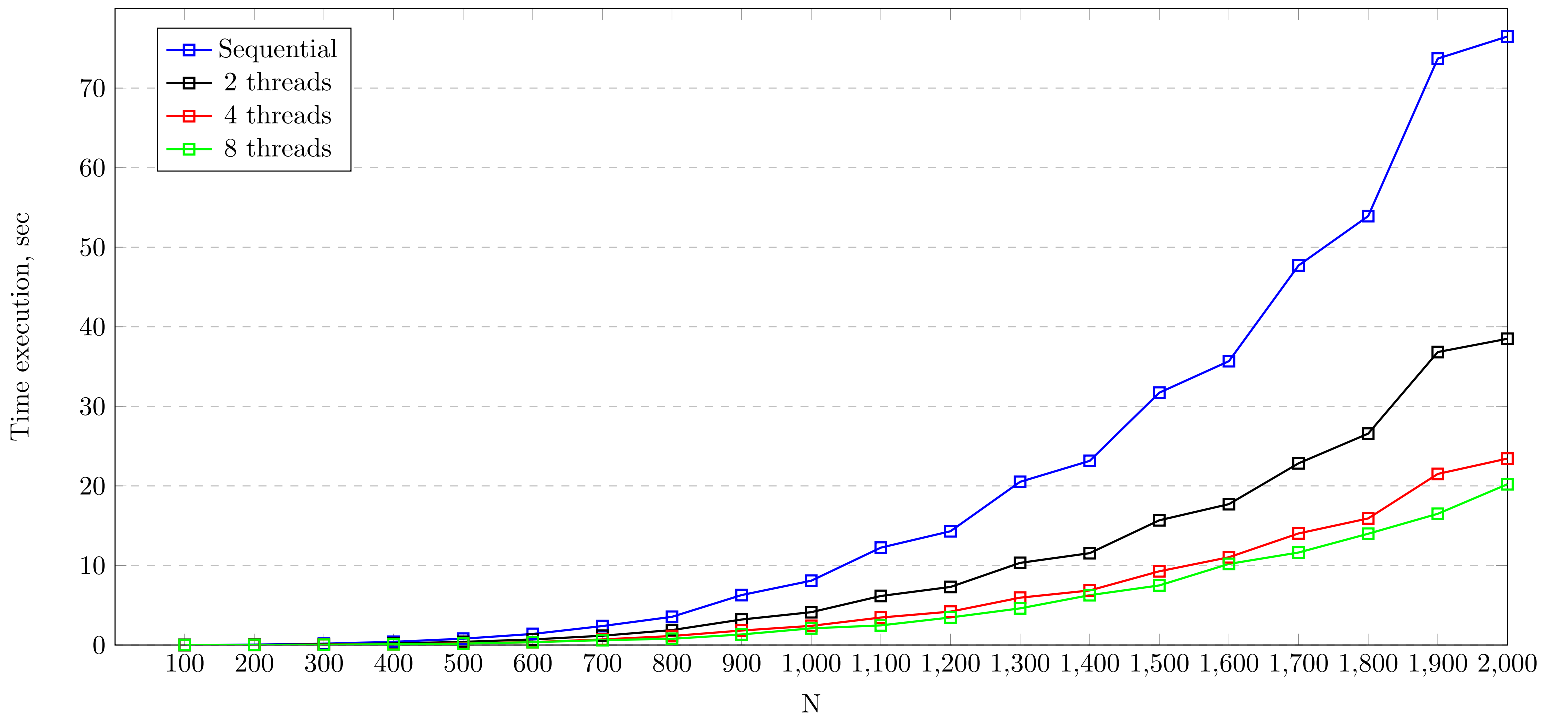

Table 2: Multiplication of matrices [N*N]*[N*N], where N equal 100,200, ... ,2000. With p1 optimization

3 Results analysis

3.1 First part

3.1.1 First loop paralleling (p1)

- A: Each process works only with its own data. There is no shared data in MPI. But we should not forget, that L3 cache is common for all cores. Before this, I made also a test for sequence on new machine. And result for A was 2.02. So we see that we have small optimization comparing with 2 execution processes. But as we need to send data for each process it's not 2 times better. Looking for execution with 4 and 8 processes we can see that 4 processes gave us almost double speed profit, but 8 cores gave us a small profit. I think that problem is in cache.
- B: For this case also the test was made. Result for case B was 3.62. The situation is similar to previous one. But lets not forget, that case B is the slowest one for p1. Because of making several processes working with big number of `j` 's. 8 processes gave us not very big, but profit, that is bigger that in A case.
- C: The code executed here was like code for A case. But sequential case had result = 1.73. So 2 processes gave us worst speed, but increasing number of processes we are given a nice profit for 4 processes and not so bad for 8.

3.1.2 Second loop paralleling (p2)

- A: Every time case A has problems with this optimization. And this time isn't an exception. Every time we need to jump between rows in this case. So it gives us such "delay". And biggest problem is, that bigger amount of processes create bigger delay in execution. L3 cache and just an "array"(we have to just on the rows) problem occur in this case.
- B: And comparing to all the paralleling techniques, this is the best for B case. Increase of processes also gives us smaller time execution. But it still isn't so big as we wanted.
- C: This paralleling makes C case work a little better than with p1 optimization. But difference isn't big. Also with increase of processes our program runs faster.

3.1.3 Third loop paralleling (p1-2)

- A: Results are almost the same as in p1. But they are a little better. That surprised me, specially for the case with 4 processes, and I have not idea, why it's faster than in p1 with 4 processes.
- B: Results are very similar to p1.
- C: Results are very similar to p1.

3.2 Second part

The most interesting thing is that 2 processes are faster in 2 times than sequential. This is first time, when we have such situation. Always it was 1.8 or 1.7 in previous technologies. 4 processes also show good result. And we should mention, that there is no HTT, so we can't see situation when we use only 1 core for 2 processes. That's why 8 processes have best results. Not so big (L3 is common by the way). But if we were working with remote machines using MPI, we would see, that increasing of processes gives us "speed" boost. But also matrices should not be so small. We could spend more time to send data and receive it.

4 Conclusion

What to say in the conclusion. It was interesting practice, because of using another environment. With had 8 separated cores (but using common cache). MPI is a good tool for remote paralleling. And if we would use openMP or Pthread technology on machines which receive data is would be a big profit.

5 References

Here you can check out about HTT.
Here you can find code sources.
Here you can read about Google Cloud Platform.
Here you can read information about MPI.