ARQUITECTURAS Y PROGRAMACIÓN PARALELA
Practico tercero

Andrei Shumak

March 2018

Contents

1 Introduction

1.1 About

This document represents second practical task (3/4) 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 Pthread library. 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

After comparing the results, find the best optimization case. Explain, why it's the best. Then test this optimization 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

Whole work has been made in C++11 using GCC 7.3.
Processor specification

CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4
Socket(s)	1
Model name	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
Stepping	3
CPU MHz	2704.268
CPU max MHz	3500.0000
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	6144K

2.2 Paralleling realization

```
// p1 paralleling / p1-2 paralleling
for (int i=0;i<n;i++)
    // p2 paralleling / p1-2 paralleling
    for (int j=0;j<0;j++) {
        rtemp = 0;
        for (int k=0;k<A.m;k++)
            rtemp += A[i][k] * B[k][j];
        C[i][j] = rtemp;
    }
// in reality this paralleling looks much more complicated. So, check out the source code.
```

2.3 Test results

Ver2 Ofast	(p1)			(p2)			(p1-2)		
	2	4	8	2	4	8	2	4	8
A	0.18626	0.125167	0.119043	196.616	322.639	923.81	102.39	164.21	151.491
B	1.60719	0.966278	1.29216	1.56806	0.8606	0.680956	1.55992	0.87448	0.997212
C	0.481157	0.241532	0.999372	0.628195	0.358321	1.26064	0.738031	0.657019	0.743734

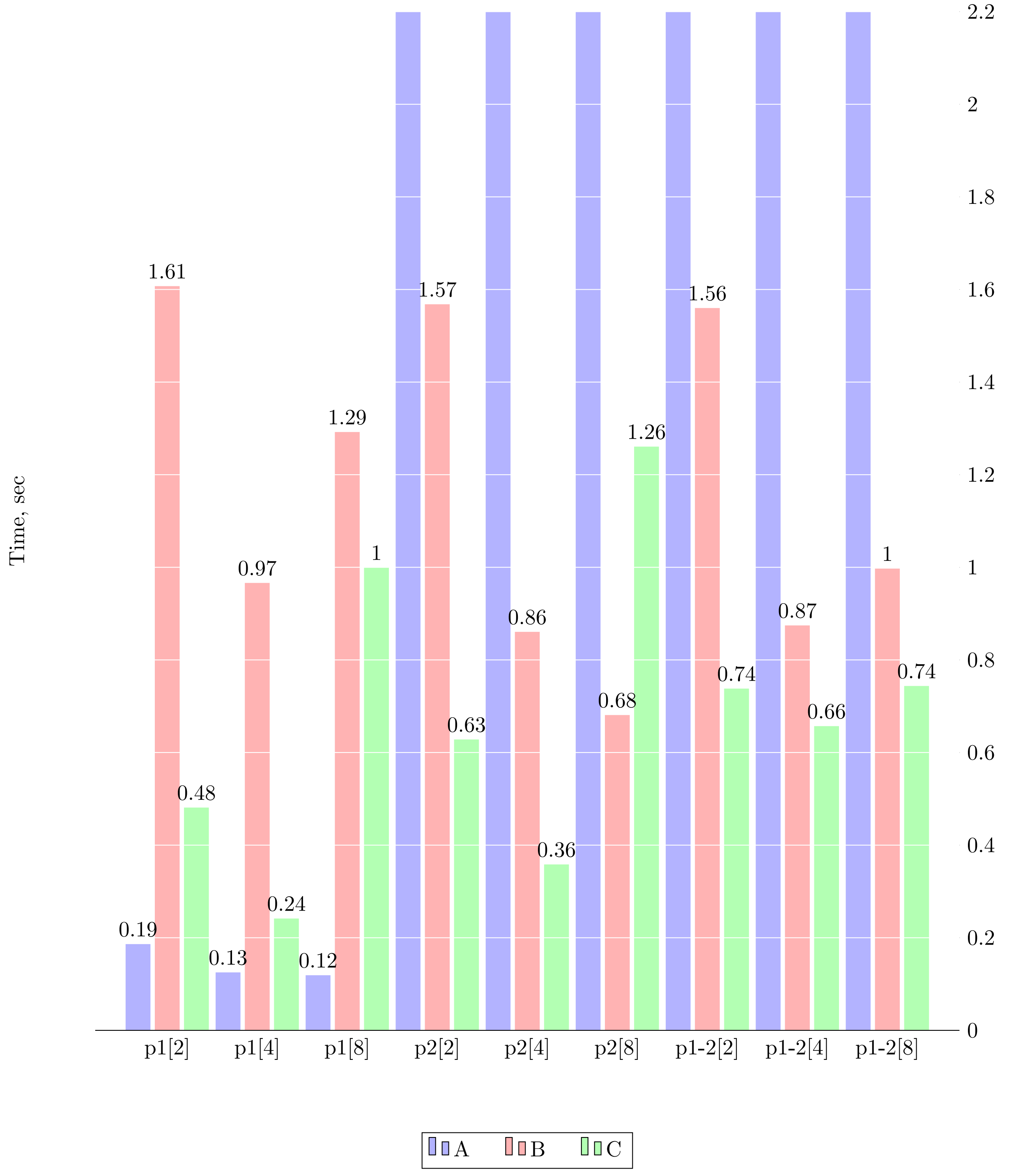

Table 1: Three different types of paralleling.

	Sequential	2 threads	4 threads	8 threads
100	0.00288479	0.000438418	0.000264458	0.000382979
200	0.00756037	0.00380557	0.00220997	0.00215456
300	0.0227632	0.0117452	0.00618789	0.0071733
400	0.0562806	0.0289096	0.0201086	0.0183031
500	0.137979	0.0695028	0.0384055	0.0390692
600	0.251417	0.124947	0.0659467	0.104885
700	0.556935	0.243365	0.131242	0.65465
800	0.951875	0.46521	0.228821	1.00674
900	1.91785	0.934513	0.465059	1.43914
1000	3.61267	1.79912	0.899405	1.99779
1100	7.27063	3.63176	1.82247	2.68657
1200	13.2686	6.46166	3.30083	3.55059
1300	14.0056	7.00627	3.5973	4.67281
1400	20.0237	10.6176	5.75171	6.06044
1500	27.3134	14.0973	7.28161	7.62823
1600	34.4836	17.08	8.93861	9.28366
1700	35.5341	18.9147	9.72411	11.7623
1800	53.9686	25.8968	13.0668	14.4938
1900	62.8329	31.6206	15.5105	17.6178
2000	68.8113	36.5647	18.7435	21.6152

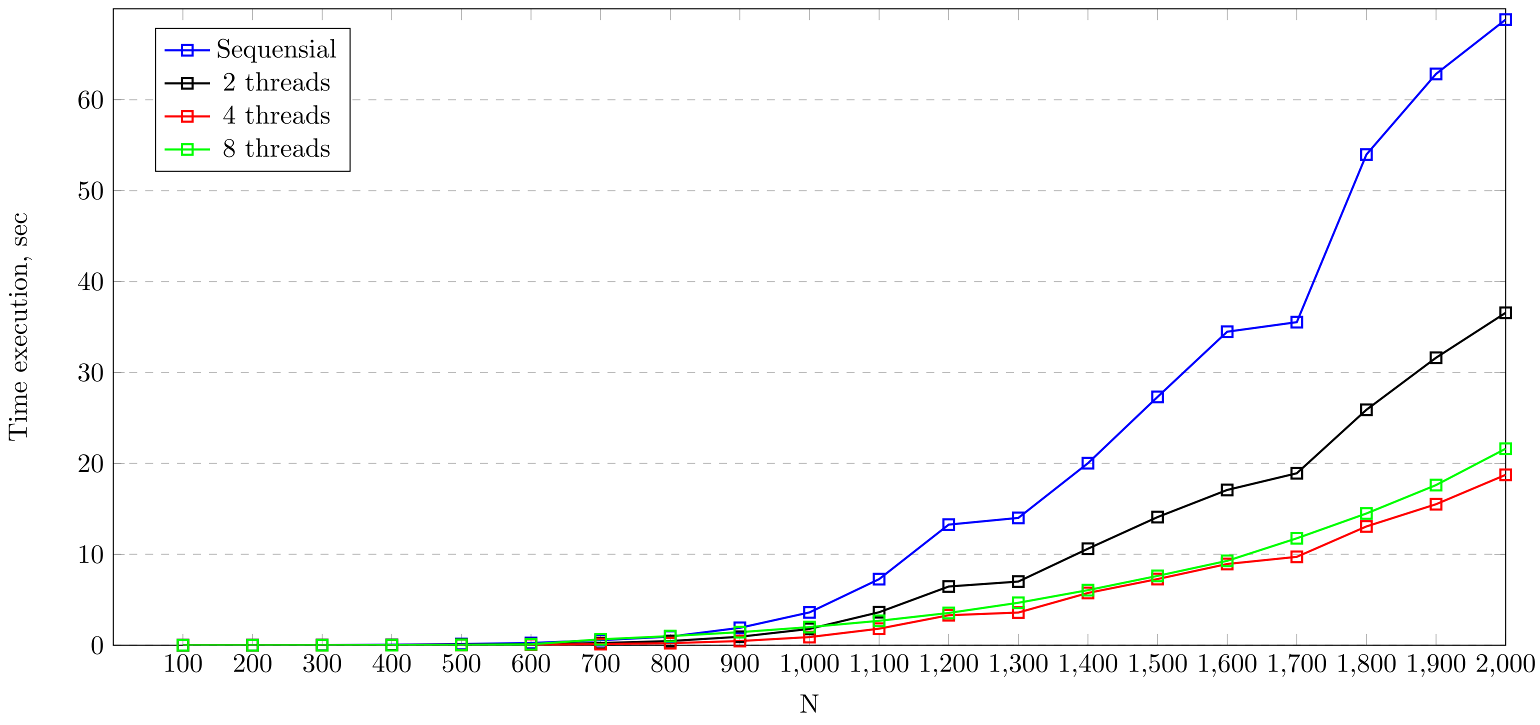

Table 2: Multiplication of matrices [N*N]*[N*N], where N equal 100,200, ... ,2000. With p1 optimization

3 Results analysis

3.1 First part

3.1.1 First loop paralleling (p1)

- A: We have to make 8.000.000*8*8 operations. First loop paralleling with 2 threads makes 2 threads, each thread should make 4.000.000*8*8 operations. In case with 4 threads our result is 1.5 times better. It's not 2 times better, because we spend a little more time for making 2 more threads, but that not a main reason. Our L3 cache is only 6MB, when matrix `8.000.000*8*8 (long long)` needs about 488 MB. And every core uses one cache. So in the case with 8 threads we have almost the same result as with 4. The reason is HTT. Minimal difference.
- B: We have to make 8*8*8.000.000 operations. First loop paralleling with 2 threads makes 4 threads, each thread should make 4*8*8.000.000 operations. It is slower than p1[2] becaues of cache work. 4 threads 1.6 times better. 8 threads work worse that 4. Probably the reason is L3 cache. very often we are waiting processor to ask values from RAM, when our cache is full of work of other thrads.
- C: The most interesting case with p1 paralleling. 8 thread work worse then 4 thread in 4 times, even 2 timers worse than 2 threads. We have such difference in time because of cache. If we will look at future square multiplication, we will see, that with bigger N cache influence becomes less and less noticeable for 8 threads.

3.1.2 Second loop paralleling (p2)

- A: This case is the most interesting. With greater amount of threads time increases in very special way : 196 sec, 322 sec, 920 sec. So, why? We make 8.000.000 iterations for first loop and then we make `n` threads. So, in case with 2 threads we will create 16.000.000 threads, with 4: 32.000.000 threads, and with 8: 64.000.000. Also for 8 threads we will have a big problems with HTT. 1 core will try to work with 2 threads, but every time we will need to load values from RAM to cache etc.
- B: Also interesting case. Here we can see, that 8 threads will be better than 4!!! reason of such behaviour is next. 8 threads case requires each line of Matrix we will do 8.000.000*8 operations, when 4 threads case 2*8.000.000*8. It easier to load to cache : line is stored in one block of RAM.
- C: Once again 8 thread not showing itself in the best way. Reasons: cache and big amount of lines(of not big length).

3.1.3 Third loop paralleling (p1-2)

- A: So, here we met problem from previous case. It's a little faster because of dividing lines of Matrix by threads. And here it helps 8 threads be better than 4.
- B: 4 threads as always faster than 2 in 1.7 times. For such combination of methods, 8 threads have a middle result.
- C: almost equal result. Nice.

So, looking at plots, I've chosen p1 as a method for Second part. It has best results in case C among all.

3.2 Second part

Ok, from our result it's easy to notice, that with bigger and bigger values of N, correlation between cases sequential, 2 threads and 4 threads becoming 2:2. Sequential 2 times lower than 2 threads, and 2 threads are 2 times lower than 4 threads. 8 threads is close to 4 threads, but it doesn't show any interesting results. Just in the begging it was loosing even 2 threads between the values 700-1000. Reasons: my code isn't ideal, cache, only 4 real cores. Some local peaks are probably because of usage of other computer applications of computer resources.

4 Conclusion

What to say in the conclusion. If you have a machine with 4 cores, it's better to use 4 threads if you want to multiply matrices and work with a huge amounts of data. Most Optimal case is usage first loop paralleling. During all calculation we will create a little number of threads.

5 References

Here you can check out about HTT.
Here you can find code sources.