ARQUITECTURAS Y PROGRAMACIÓN PARALELA

Practico segundo

Andrei Shumak

March 2018

Contents

1 Introduction

1.1 About

This document represents second practical task (2/4) 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

There are 3 cases of matrices multiplication:

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

With a usage of OpenMP parallel code. 3 various types of parallelism:

```
a = Matrix;
b = Matrix;
c = Matrix;
// p1
// #pragma omp parallel for shared(a,b,c)
// p1-2
// #pragma omp parallel for shared(a,b,c) collapse(2)
for (int i = 0;i < c.rows;i++)
    // p2
    // #pragma omp parallel for shared(a,b,c)
    for (int j = 0;j< c.columns;j++)
      {
       long long rtemp = 0;
       for (int k = 0;k<a.columns;i++)
           rtemp += a[i][k] + b[k][j];
        c[i][j] = rtemp;
        }
```

Compare the results and find the best optimization. Then compare by time best found optimization mode with mode without parallelism but with -O3 in multiplication of matrices [100*100][100*100], [200*200][200*200] ... [2000*2000]*[2000*2000].

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 Test results

Ver2 Ofast	(p1)			(p2)			(p1-2)		
	2	4	8	2	4	8	2	4	8
A	0.197695	0.129064	0.119859	6.31513	7.43275	9.83482	0.232939	0.132087	0.127119
B	1.64595	0.966132	1.66842	1.54578	0.888284	0.642319	1.76337	0.958156	1.25229
C	0.519145	0.243983	1.00561	0.510157	0.250347	1.05968	0.505038	0.244016	1.01196

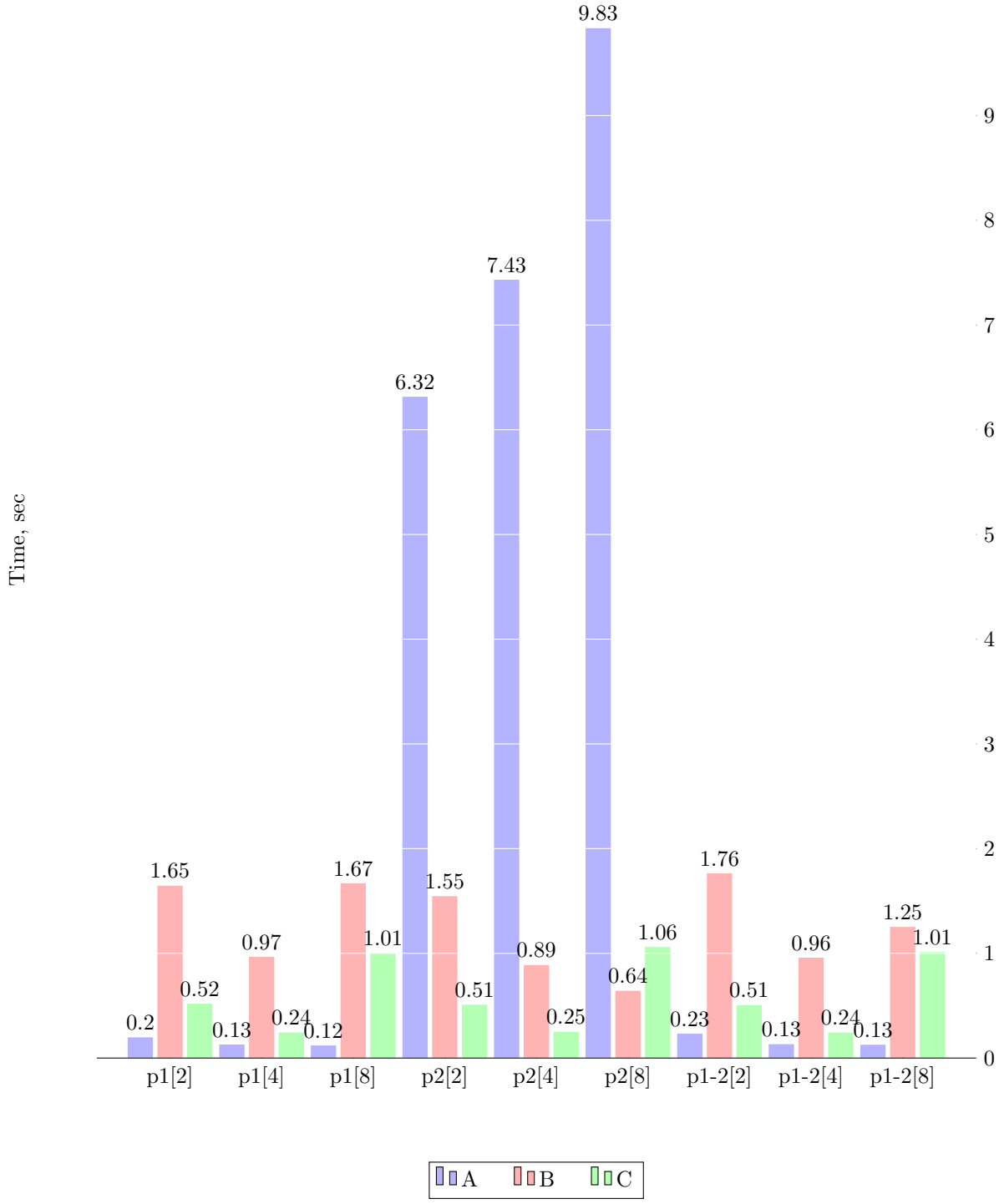

Table 1: Three different types of paralleling.

	2 threads	4 threads	8 threads	Sequential
100	0.000359684	0.000201792	0.000209434	0.000664618
200	0.0033987	0.00173079	0.00168639	0.00694997
300	0.0115897	0.00626077	0.00718302	0.0234747
400	0.0297665	0.0163515	0.0163389	0.0568424
500	0.0700867	0.050693	0.0402516	0.140952
600	0.122435	0.0748994	0.10624	0.248233
700	0.253131	0.137676	0.67711	0.611602
800	0.48812	0.24231	1.01659	1.31977
900	0.960322	0.471842	1.463	2.10859
1000	1.81641	0.915272	2.01474	4.1982
1100	3.68893	1.84725	2.7318	7.63495
1200	6.46311	3.26538	3.56235	12.7478
1300	7.14515	3.57895	4.85347	14.2194
1400	10.959	5.59556	5.82831	20.5534
1500	14.5075	7.36882	8.03367	26.9261
1600	17.881	9.09748	9.33654	32.527
1700	19.1924	9.98215	11.3708	35.689
1800	26.1325	13.4464	13.9006	47.8608
1900	31.405	16.1072	16.9868	57.1945
2000	38.0737	18.8906	20.693	68.331

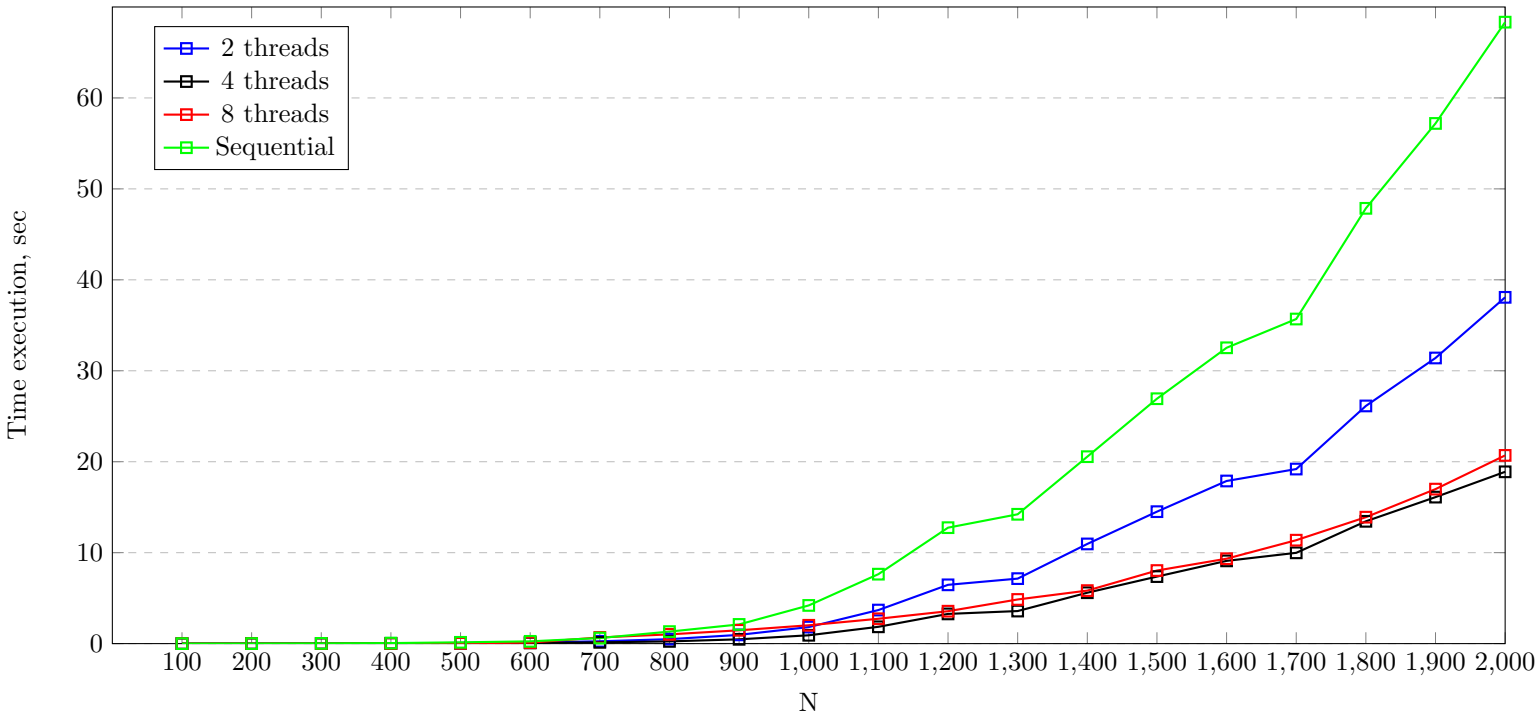

Table 2: Multiplication of matrices [N*N]*[N*N], where N equal 100,200, ... ,2000.

3 Results analysis

We have 3 cases of matrices multiplication: A,B,C. Our plots show us, that the time computer need to multiply is different.

3.1 Method of paralleling

- p1 : P1 gives us good efficiency. Comparing work on 2,4,8 threads we can notice, that 4 threads almost 2 times faster, than 2 threads, but we can't say the same thing about 4 and 8. The problem is in HTT (Hyper-threading Technology). This technology gives us increase in productivity up to 30%, but in our case sometimes it's doing worse than 4 threads. We can see it in the cases B and C.
- p2 : P2 gives worst results among of types of paralleling. Specially it's seen in the case A. Reason of it is that 8.000.000 times we create threads for 8 operations, then once again. But surprisingly 8 threads work better than 4 in case B (HTT doing well in this situation).
- p1-2 : This case is very similar to p1. In some cases it's better, in some worse.

3.2 Square matrices multiplication

For matrix square multiplication p1-2 method has been chosen. Case C for p1 and p1-2 works equally.
So here we see, that HTT in this case doesn't give 8 thread code any advantages towards 4 threads.

4 Conclusion

Making all the tests we can conclude, the best paralleling option for our program is p1 or p1-2, and not every time running 8 threads will be better, than running 4 threads (on processor with 4 physical cores). It all depends on task, where it will be used. If we had computer with 8 physical core we would see, that 8 threads are 2 times faster than 4.

5 References

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