ARQUITECTURAS Y PROGRAMACIÓN PARALELA
Practico primero

Andrei Shumak

March 2018

Contents

1 Introduction

1.1 About

This document represents first practical task (1/4) 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]

Count multiplication time for every case using compiler and "rtemp" optimizations. For compiler optimization options -O0, -O1, -O2, -O3, -Ofast, -Os, -Og can be used. Next block of code represents "rtemp" optimization :

```
a = Matrix;
b = Matrix;
c = Matrix;
for (int i = 0;i < c.rows;i++)
    for (int j = 0;j< c.columns;j++)
    /*
       There could be 2 versions of multiplication

     First one is without rtemp:
     {
         for (int k = 0;k<a.columns;i++)
             c[i][j] += a[i][k] + b[k][j];
     }
     Second one is with rtemp
     {
       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. Than compare by time best found optimization mode with mode without optimizing(-O0) 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. For matrices representation class Matrix has been made:

```
class Matrix {
private:
    LL **table;
    int n; // rows
    int m; // columns
    /* some private methods */
public:
    /* some public methods */
    void multiply_v1(Matrix &A, Matrix &B);
    void multiply_v2(Matrix &A, Matrix &B);
};
```

LL[1] **table represents our matrix, n - amount of rows, m - amount of columns. There are 2 methods `void multiply_v1(...)` and `void multiply_v2(...)` which represent two types of multiplication. First one is without rtemp, second one is with rtemp.
For time measurements `chrono` library has been used. There were made 10 tests for every case, with absolutly random values.

2.2 Test results

2.2.1 Without rtemp

	O0	O1	O2	O3	Ofast	Os	Og
A	3.07345	0.983588	0.464423	0.402983	0.406929	0.905391	0.978053
B	4.29506	3.0854	2.85216	2.83545	2.88657	2.95676	3.04116
C	4.52452	2.23357	0.960759	0.898626	0.91247	1.76963	1.86038

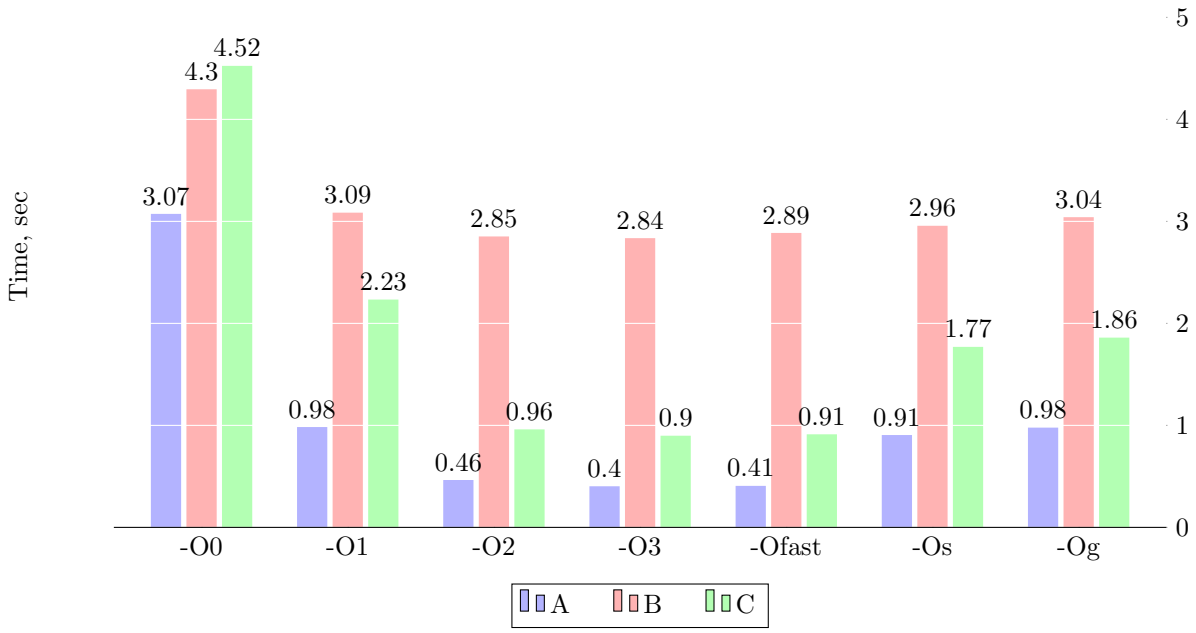


Table 1: Multiplying without rtemp

[1] #define LL long long int

2.2.2 With rtemp

	O0	O1	O2	O3	Ofast	Os	Og
A	1.82962	0.457324	0.380702	0.392276	0.375317	0.582567	0.659562
B	3.34484	2.86222	2.82007	2.83445	2.86467	2.87592	2.930066
C	2.72272	0.973169	0.87036	0.841198	0.941094	1.31991	1.36633

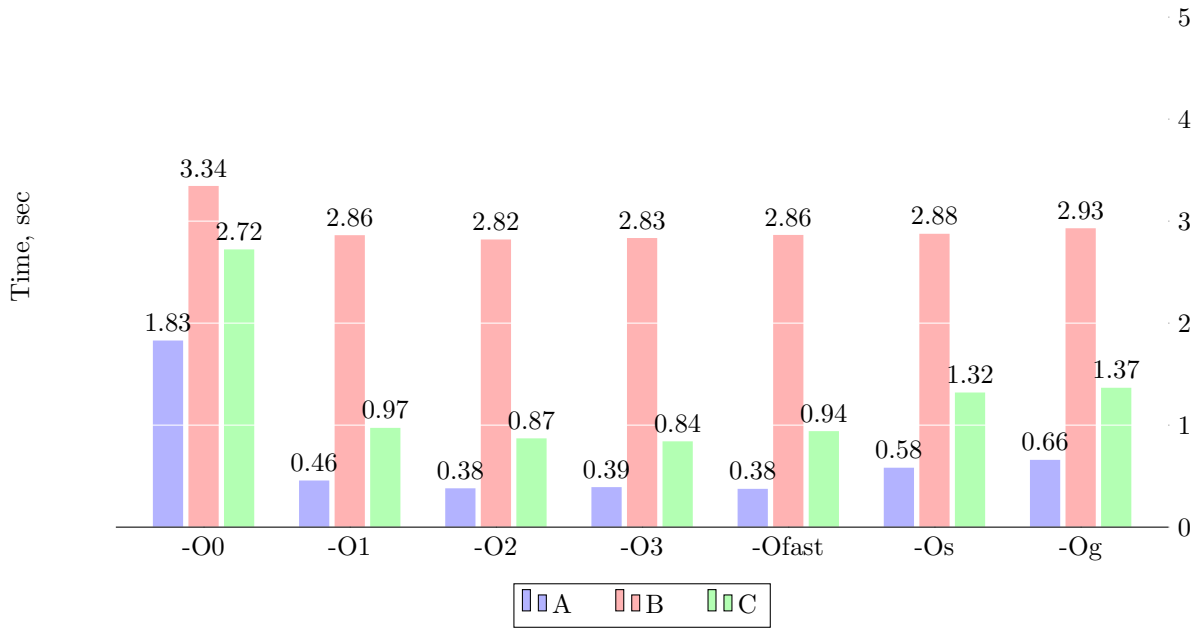
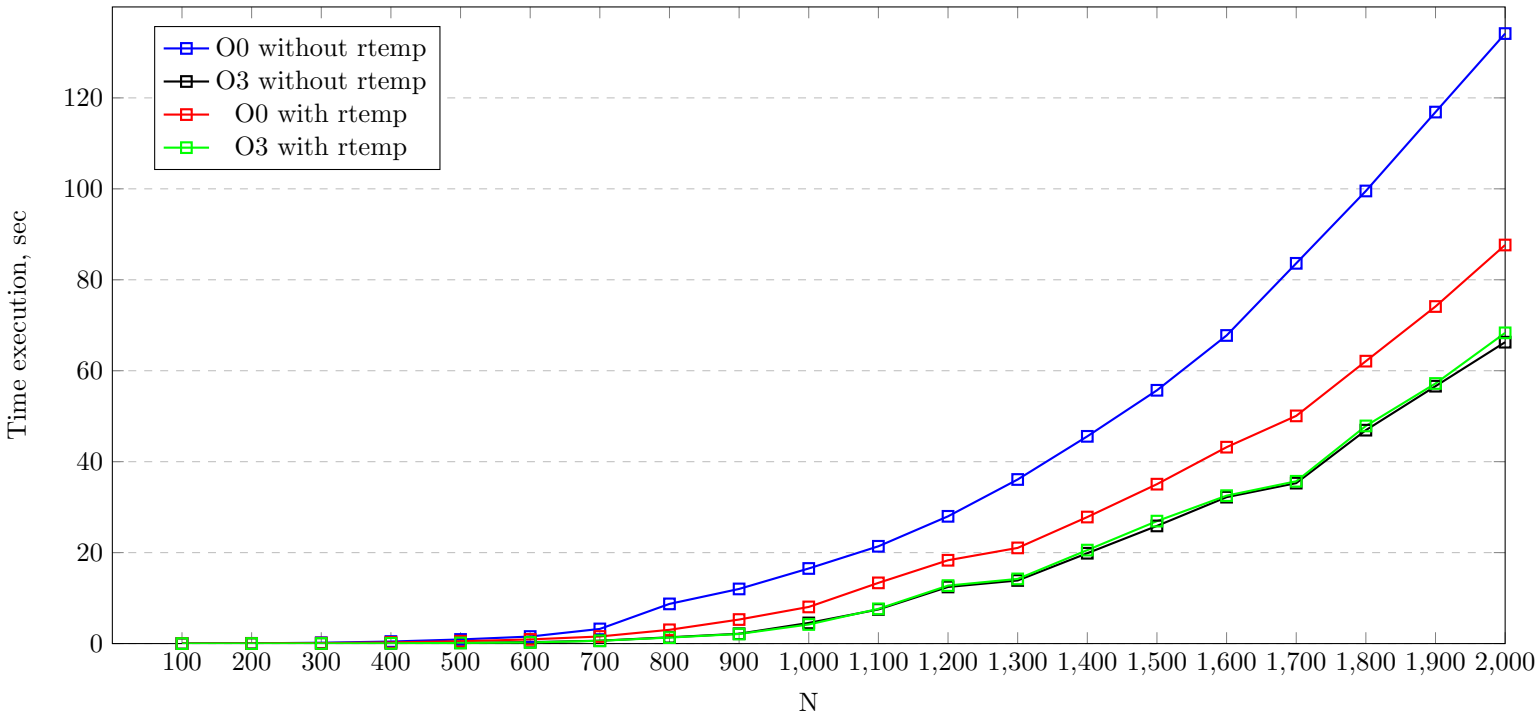


Table 2: Multiplying with rtemp

2.2.3 Square matrices multiplication

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

	Without rtemp		With rtemp	
	-O0	-O3	-O0	-O3
100	0.00598351	0.000844198	0.0033263	0.000664618
200	0.051074	0.00859191	0.0294428	0.00694997
300	0.18718	0.0297774	0.102197	0.0234747
400	0.445468	0.0748268	0.247854	0.0568424
500	0.903173	0.166209	0.512127	0.140952
600	1.54614	0.310886	0.913907	0.248233
700	3.21858	0.627665	1.55668	0.611602
800	8.72482	1.384	3.02727	1.31977
900	12.0134	2.16363	5.28504	2.10859
1000	16.5148	4.53926	8.05677	4.1982
1100	21.392	7.49996	13.3489	7.63495
1200	27.9826	12.4482	18.3188	12.7478
1300	36.0828	13.8744	21.0361	14.2194
1400	45.5712	19.8646	27.8248	20.5534
1500	55.6991	25.8721	35.0532	26.9261
1600	67.751	32.1973	43.1882	32.527
1700	83.6176	35.2835	50.0701	35.689
1800	99.5291	46.928	62.1055	47.8608
1900	116.88	56.602	74.1218	57.1945
2000	134.145	66.2761	87.6467	68.331



3 Results analysis

3.1 Without rtemp vs. with rtemp

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

3.1.1 Without rtemp

Let's take a look at first table's plot - "Multiplicatoin without rtemp".

- -O0: As we see cases B and C almost 1.5 times slower than case A. Reason of a such behaviour can be processor caching. It can be, that in the case A second matrix ([8*8]) is being cached and access to this memory is very fast.
- -O1: Usage of the first optimization causes a big time gain for all the cases, especially for the first one. Case A becomes 3 times faster, case B is almost 1.5 times faster and the case C is 2 times faster compared to -O0 optimization.
- -O2: Cases A and C become 2 times faster. Case B doesn't show any big progress.
- -O3: This optimization asks more time for compilation than anything else, but it gives us "minimal" available time execution of this code.
- -Ofast: This optimization should include -O3 optimization and show better results, its results are close, but not better than -O3.
- -Os: This optimization is like -O2 but with some flags turned off. Our executable becomes smaller than using -O2 but in the cases A and C time execution increases 2 times.
- -Og: Similar to -Os, but a little bit slower.

3.1.2 With rtemp

The optimization situation is very similar to "without rtemp". Usage of rtemp makes code runs faster even without any optimizations. But because of this we don't have big time "gaps" with optimization. For all the optimization -O1, -O2, -O3, -Ofast we have very similar time execution.

3.1.3 Compare with and without rtemp

rtemp gives us very good optimization. For example with -O0 we have 1.5 faster execution. -O1 optimization "with rtemp" has almost the same results as "without rtemp" -O2. There is a very small difference for -O2, -O3, -Ofast.

Looking at all the results, we see that the best optimization is -O3. That's why we used it for comparing in square matrices multiplication.

3.2 Square matrices multiplication

This plot shows the best how rtemp optimization is good. In time execution it was closer to -O3 then to -O0 without optimization. But the most interesting thing on this plot is that -O3 with rtemp optimization from `N>=1100` shows a little bit worse results then -O3 without rtemp. An extra test was conducted, when N = 8000:

- -O3 withoout rtemp: 6280 sec
- -O3 with rtemp: 6328 sec

4 Conclusion

Making all the tests we can conclude, the best choice for optimization is -O3. rtemp is a good option, when we use code without any optimizations or with optimizations like -O1, -Os, -Og. It gives really greater advantages than without it. -O2, -O3, -Ofast with rtemp are a little bit faster than without it. It should be mensioned, that rtemp gives advantages for -O3 in square multiplication only when `N<=1000`, after this value, we see, that it starts loosing to -O3 without rtemp compilation mode.

5 References

Here you can find documentation about GCC optimization.
Here you can find code sources.