UPPSALA
UNIVERSITET

High Performance Programming
Final project

Sy-Hung Doan

March 19, 2019

Contents

1 Introduction

This report briefly summarizes the efforts of performance optimizing for the problem: Matrix Multiplication using Strassens Algorithm. The computational results and relevant comparisons are shown later on with corresponding tables and graphs.

This report is divided into 6 sections. The second section shall describe the problem. My solution is present in the third section and some experiments and performance measurements are done in the fourth one. The fifth section is the conclusion and documents that I consulted are shown in the last section, references.

For the purpose of reproducibility, the configuration of the machine that I used is below and I/O time is still included in the results.

My optimization and parallelization is obviously trivial. I hope you can give me comments on how I can improve more.

Table 1: Machines used for performance testing in this report

Ref. Name	**CPU info / Operating System / Compiler**
Dell Inspiron 7447	Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz × 8 , Ubuntu 18.04 LTS, gcc 7.3.0

2 Problem description

The problem that I need to implement and optimize is the Matrix Multiplication using Strassens Algorithm. It computes faster than the standard matrix multiplication algorithm [1]

Below is an example of how a naive solution would be, which takes a time complexity of $\mathcal{O}(n^3)$.

```
// Consider
// A is the first matrix, B is the second one
// C = A * B
// N is obviously the size of ea
void matrix_multiply(int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
```

```
    }
}
```

The main idea of Strassen's method is to reduce the number of recursive calls (to calculate matrix multiplication) to 7. With each input matrix's size of $N = 2^n$ Strassen's method divides matrices to sub-matrices of size N/2 x N/2 as shown in the following equation.

$$\begin{matrix} \mathrm{A}_{00}A_{01} \\ A_{10}A_{11} \\ B_{10}B_{11} \\ C_{10}C_{11} \end{matrix} \times \begin{matrix} B_{00}B_{01} \\ = C_{00}C_{01} \end{matrix}$$

However, in Strassen's method, the four sub-matrices of the result are calculated using following formulae.

$$P_1 = A_{00}\times(B_{01}-B_{11})P_2 = (A_{00}+A_{01})\times B_{11}P_3 = (A_{10}+A_{11})\times B_{00}P_4 = A_{11}\times(B_{10}-B_{00})P_5 = (A_{00}+A_{11})\times(E \tag{1}$$

$P_1, P_2, P_3, P_4, P_5, P_6, P_7$ are all N/2 x N/2 matrices. Addition and Subtraction of two matrices takes O(N2) time. Thus, time complexity of Strassen's algorithm can be written as:

$$T(N) = 7T(N/2) + O(N2)$$

According to Master's Theorem [2], time complexity of above method is $\mathcal{O}(N^{\log 7})$, which is roughly $\mathcal{O}(N^{2.8074})$ [3].

3 Solution method

3.1 How to run my code

To run my code, simply use `make` and then use the command `./main N` with N is the size of the matrix. Note that N must be a power of 2 since this algorithm is specific for that kind of matrix.

3.2 Data structure

In my solution, I use dynamically-allocated arrays of `double` to store matrices as the stack is likely to overflow if the matrix's size is large enough. (I could not declare three matrices with size 512x512).

3.3 Input

I use matrices of `double` and each number is in range from -100.0 to 100.0 randomly.

Initially, in order to check the correctness of the implementation, I stored each matrix with size X in a separate file. I had a total of 9 binary files, 3 for each pair of X matrices and its product namely `input_521_1.mat`, `input_521_2.mat`, `output_512.mat`, `input_1024_1.mat`, `input_1024_2.mat`, `output_1024.mat`, `input_2048_1.mat`, `input_2048_2.mat`, `output_2048.mat`, since with `X=1024`, a single matrix takes up to `8.4MB`.

(The output files were generated using the naive method and the compare.c file is responsible for checking the correctness of the output).

For the final solution, I used a function that generates 2 input matrices with the given size, and the size is one of the command line arguments.

3.4 Optimization

The first attempt that I made was to find a minimum size of the matrix `N_min` so that when a recursive call to multiply matrices with size `N_min`, the multiplication will be performed with the naive algorithm to avoid overhead of deeper recursive calls and utilize the cache.

I tried with 2, 4, 8 up to 512 and found that 256 was a good value as it gave shorter computational time.

I also tried to manually implement the SIMD, but it was marginally faster, but even worsen for the 2048x2048 matrices. Then, `-O3` was used, which halved execution time (it by default has option `-ftree-vectorize`).

In any part of code that involves 3 nested codes with order `i, j, k`, I tried different permuations of `i, j, k`, and `i, k , j` had the best performance.

3.5 Parallelization with OpenMP

I used OpenMP for the simplicity. Some parts of my code was modified so that it can be inside a `for` loop. Then I used `#pragma omp parallel for` for each loop. The problem is that the number of loops is just 4, so there can be a maximum of only 4 threads.

4 Experiments

Computational time for the inital Strassen algorithm's implementation was shown in the table below:

Matrix size	**Computational time**
512x512	0.764s
1024x1024	5.276s
2048x2048	40s

After optimizing serial code, the result was really better

Matrix size	Computational time	Speedup
512x512	0.071s	10.76
1024x1024	0.596s	8.85
2048x2048	2.853s	14.02

Parallelization with `NUM_THREADS=4`:

Matrix size	Computational time	Speedup
512x512	0,050s	15.28
1024x1024	0.411s	12.83
2048x2048	1.291s	30.98
4096x4096	7.56s	
8192x8192	67.158s	

5 Conclusions

The result is not very optimal. For the parallelization section, I planned to put all the work (like addition, subtraction, multiplication) into a queue, so that having more than 4 threads still improves the performance, but I was not sure how to implement that. Could you give me some hints if possible ?

6 References

[1] Wikipedia, "Strassen algorithm," last accessed: March 20, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Strassen_algorithm

[2] ——, "Master theorem (analysis of algorithms)," last accessed: March 20, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)

[3] GeeksforGeeks, "Divide and conquer — set 5 (strassen's matrix multiplication)," last accessed: March 20, 2019. [Online]. Available: https://www.geeksforgeeks.org/strassens-matrix-multiplication/