AMS595 Project Report: Strassen matrix in C++

Chi-Sheng Lo

Student ID: 114031563

Chi-Sheng.Lo@stonybrook.edu

1. Project objectives

1.1. Introduction

How to conduct multiplication between two large matrices and more precisely, how to solve for the inverse of nonsingular matrix in $< C_2 n^{log_2 7}$ operations? The Strassen's algorithm is designed to fasten the matrix solution in such large-scale operation.

1.2. Background (Literature Review)

Prior to the discussion over this Strassen's matrix project, we should know about its past and present development. Originally, Strassen (1969) introduced this algorithm that computed the product of two matrices of order 2 by 7 multiplications and 18 additions. He applied the recursive block LDU factorization to compute the inverse of matrix operations and showed that the product of two matrices could be computed by $< (4.7) n^{log_2 7}$ operations. Bunch and Hopcroft (1974) used the Strassen's fast matrix multiplication to obtain the triangular factorization of a permutation of any nonsingular matrix of order n = 2^k in $< (2.04) n^{log_2 7}$ operations. More recently, Macedo (2016) extended Strassen (1969) by using two stepwise refines to the transformation of two matrices into the naïve algorithm and Strassen's matrix multiplication algorithm.

1.3. Aim and Objectives

By using the C++, this project aims to conclude the best cut-off value that spends the least running time. My objectives are to find a cut-off matrix size to switch to the naïve algorithm; otherwise, the overhead will be unacceptable when it goes down to 2 x 2 matrix. The multiplication between these two large matrices will be conducted with cut-off to different values. After creating a timer that calculates the time needed to run the algorithm, we will be able to compare the time required for naïve algorithm at the designated cut-off value.

2. Techniques and tools

Since the final output is about the speed, the computational power of the software and hardware combined are actually as important as the optimization of the code itself. Therefore, before I analyze the coding part for the Strassen algorithm, let me first introduce the accompanied tools: software and hardware. Then the latter sections will discuss the mathematical background, coding, and actual implementation for this project.

2.1. Environment

Operating system:

Windows 10

Software:

My programming language for this Strassen matrix project is structured and implemented in C++. My IDE for C++ is Dev C++ version 5.11 for Windows OS developed by the Bloodshed Software.

Hardware:

CPU: Intel i7-7700 at 3.60 GHZ/3.6 GHZ

Ram: 16 GB

2.2. Mathematics behind Strassen algorithm

Strassen (1969) first proposed a new algorithm to execute large matrices with faster speed. Technically speaking, it is an algorithm that is asymptotically faster than the naïve O(n^3) = O($n^{log_2 8}$) algorithm. Initially, the algorithm can be expressed as parallelized block matrices in which we can let A, B $\in \mathbb{R}^{nxn}$ and C = AB where n is a power of two. Then we need to bring down the number of sub-calls to matrix multiples

from O($n^{log_2 8}$) to O($n^{log_2 7}$). Therefore, it is also an example of recursive problem. More of Strassen algorithm in matrix expression will be illustrated in section 2.4.

2.3. Coding structure

From the top down point of view, there are two main parts: input and output. For the input part, when the input r is greater than or equal to n, only the naïve algorithm is used for calculation. When the input r is smaller than 2, the Strassen algorithm automatically switches to the naïve algorithm when it goes down to 2 x 2 matrices. For the output part, the program has only one output, which is the calculation time (seconds) for matrix multiplication.

The program starts off from the addition function in which I did the addition for one of the block of matrices A and B. The procedure is the same for subtraction function. Subsequently, I applied the cut function to copy one block from matrix A and then return a totally new matrix. Then I ran the multiplication by using multiplication function in which the time complexity is set at O(n^3). Thereafter, I made use of the Strassen function to run the recursive algorithm in which I first calculated the first ten auxiliary matrix from S1 to S10 and then calculate the multiplication from P1 to P7. As mentioned in the last paragraph, the Strassen algorithm is set to switch to the naïve algorithm when it goes down to 2 x 2 matrices if the input r is smaller than 2. The four partitioned block matrices will form a large matrix.

To get to the output, I must let n larger than 2 and n must be the exponent of 2. The bool is2n function helps to determine whether n meets the criteria in that n must larger than 2 and is the exponent of 2. In the contrast, there is less restriction for r as long as it is an integer. However, the matrix operation throughout the algorithm is divided by two every time; hence, r should be the exponent of 2. In this project, n is an important input

for calculating the size of matrix. Therefore, when n < 2, this means n = 1. During the "divide and conquer" process, we must let n x n matrix divided into four n/2 x n/2 matrix; therefore, n must be the exponent of 2. If n is odd, then n cannot be divided; thus, to ensure success in the process, we must let n becomes exponent of 2. Moreover, r is the value of the cut-off, when the matrix size become smaller than 4, I then use the normal way to conduct matrix multiplication or the so-called naïve algorithm. When the above is set, the randomly generated matrices A and B are in n x n size. The range of elements for each matrix is set at [-10, 10]. At the very last section of my code, I had to make a clock that calculates the time in seconds which is the desired output we need for each cut-off value.

2.4. Inside the Strassen algorithm

Every time we use the Strassen's matrix, the algorithm determines whether the size of matrix is smaller or equals to the cut-off value r; if the regular multiplication operation has its size larger then r, then the Strassen will take over.

The expression of Strassen's algorithm in matrix is shown below:

$$\mathrm{A} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \mathrm{B} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Creation of ten n/2 by n/2 matrices:

$$S_1 = B_{12} - B_{22}$$

$$S_2 = A_{11} + A_{12}$$

$$S_3 = A_{21} + A_{22}$$

$$S_4 = B_{21} - B_{11}$$

$$S_5 = A_{11} + A_{22}$$

$$S_6 = B_{11} + B_{22}$$

$$S_7 = A_{12} - A_{22}$$

$S_8 = B_{21} + B_{22}$

$S_9 = A_{11} - A_{21}$

$S_{10} = B_{11} + B_{12}$

After conducting addition and subtraction, I then obtained four blocks: A_{11}, A_{22}, B_{11}, and B_{22} for multiplication each by each by using the cut function. Thereafter, the recursive calculation for n/2 by n/2 matrix multiplication will be conducted for seven times. These seven results of recursion are:

$P_1 = A_{11} * S_1$

$P_2 = S_2 * B_{22}$

$P_3 = S_3 * B_{11}$

$P_4 = A_{22} * S_4$

$P_5 = S_5 * S_6$

$P_6 = S_7 * S_8$

$P_7 = S_9 * S_{10}$

Finally, the multiplication between matrix A and B becomes the matrix C which is composed of the four blocks. I used the merge function to obtain the matrix C which includes four n/2 by n/2 matrices that are C_{11}, C_{12}, C_{21}, and C_{22}. The expression in matrix is shown below:

$$\mathrm{C} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Where:

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

2.5. Execution

To execute the C++ program in the Dev C++ version 5.11 which is my IDE for C++, the first step is to click on the “compile and run” in execute section of the IDE. Since there is no error found in the code and so has successfully compiled, then there will be a pop-up of exe file. After that, we need to manually do the inputs in the exe. For example, for r = 2, we just need to input 1024 2 (then hit enter). Finally, the output will be presented in system running time by seconds.

To see the entire code and execute the program, please open the accompanied C++ file.

3. Conclusion

3.1. Significance

Although the project is based on Strassen (1969) and many other researches had made contributions by replicating and extending Strassen (1969), my coding structure and the corresponding output are not identical. The algorithm for this Strassen's matrix has been optimized for several times at my best performance. Overall, I had successfully reached my goal in finding the best cut-off value. The following sections will analyze my findings.

3.2. Result

To generate the output, at first, I must create multiplication of two matrices or precisely speaking, the 1024*1024 operation. As mentioned earlier, when the matrix size is less than or equal to r, the naïve algorithm immediately comes into play. If opposite, we will use the Strassen algorithm for recursive calculation.

In this project, I set the cut-off values r as 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, and 2048, respectively. Each value of r is at the increment of two times the previous r to meet the strict requirement in which we must set the exponentiation with number two as the base and integer as the exponent. Table 1 shows the recorded experimental results; apparently, when r is equal to 1024, it is equivalent to using only the naïve algorithm. The output also shows that when the cut-off value r is set at 64, the Strassen algorithm has the fastest running speed, followed by r is set at 32. To contrast, it takes the longest time for r equals to 2 which is at the threshold of moving to naïve algorithm.

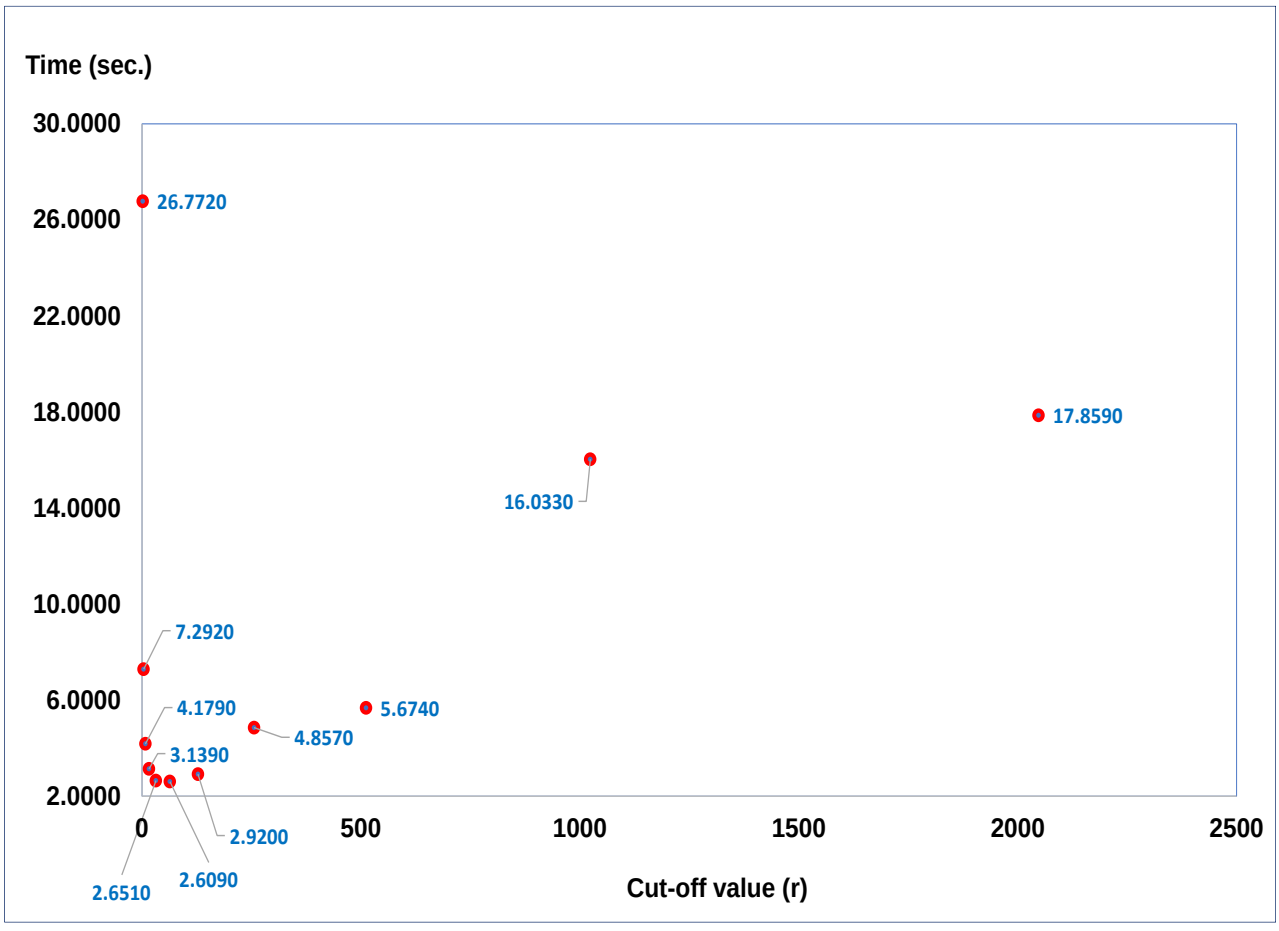
Exhibit 1 is just the reflection of table 1 but in terms of XY plane. As we can see from exhibit 1, the r-time relation is an illustration of asymmetric parabola. The input-output

result are shown below:

Table 1

r	Time (sec.)
2	26.7720
4	7.2920
8	4.1790
16	3.1390
32	2.6510
64	2.6090
128	2.9200
256	4.8570
512	5.6740
1024	16.0330
2048	17.8590

Exhibit 1



3.3. Implication from the output result and concluding remark

The above outcome illustrated in last page (and described in section 3.2) is caused by a combination of the asymmetric relationship between cut-off value and running time and computing usage. Moreover, how optimize the code structures and how strong the computer performs could all impact the outcome and possibly by significant margin. Even under the same condition, every single experiment is unique from each other due to memory usage and CPU. As we know, the CPU cannot maintain at the top speed at all times. For instance, the run time for Strassen matrix should get significantly slower when the computer is overheated. Under normal condition, the running time will be different by millisecond but overall, the pattern stays the same.

The Strassen algorithm tends to consume a large amount of memory during the calculation; hence, the deeper the recursion of algorithm, the more the utilization of memory. When the call has deep recursion, the Strassen algorithm runs lengthier, even longer than the naïve algorithm. In the contrast, when the recursive call is depthless, the calculation time optimized by the Strassen algorithm will not take as much time as the time of recursion and memory application combined. Having said that, the time consumption of the Strassen algorithm will still be longer than the naïve algorithm.

Throughout this project, I had learned more about the execution and optimization of matrix under the C++ environment. Besides coding, I had also strengthened my knowledge in advanced linear algebra theory from Strassen (1969) and Bunch and Hopcroft (1974) who particularly laid the foundation in fast multiplication. Since this project is also related to optimal running time for matrix operation, the future extensions can be led into two direction. One is to continue the same project but with focus on the performance optimization. Another direction is to conduct research in the state-space

models which also requires large-scale multiplication, sparse matrices, and Cholesky decomposition for obtaining likelihood estimation.

References

Bunch, J and Hopcroft, JE, 1984, 'Triangular factorization and inversion by fast matrix multiplication', *Mathematics of Computation,* vol. 28: 125, pp. 231-236.

Macedo, HD 2016, 'Gaussian elimination is not optimal, revisited', *Journal of Logical and Algebraic Methods in Programming,* vol. 86, pp. 999-1010.

Strassen, V 1969, 'Gaussian elimination is not optimal', *Numerische Mathematik,* vol. 13, pp. 354-356.