COMS4032A: Applications of Algorithms – Assignment 1

Nihal Ranchod
2427378
BSc Hons

August 12, 2024

Contents

1 Introduction

This assignment focuses on implementing and analysing three algorithms for square matrix multiplication: Square-Matrix-Multiply, Square-Matrix-Multiply-Recursive, and Strassen's Method, as presented in Chapter 4 of Thomas H et al. 2009. The goal is to implement these algorithms and to empirically evaluate their asymptotic running times by testing them on square matrices of varying dimensions.

This analysis will compare the three square matrix multiplication algorithms each implemented in C++ which can be found in the smm_algorithms.cpp file. All the code and results for this assignment can be found at https://github.com/nihal-ranchod/COMS4032A—Application-of-Algorithms-Assignments.

2 Implementation of Algorithms

2.1 Square Matrix Multiplication

The Square Matrix Multiplication algorithm is implemented according to Thomas H et al. 2009 page 75. The square matrix multiply procedure takes $\Theta(n^3)$ time, Thomas H et al. 2009.

2.2 Square Matrix Multiplication Recursive

The Square Matrix Multiplication Recursive algorithm is implemented according to Thomas H et al. 2009 page 77. To handle matrices that are not exact powers of 2, the matrices are padded with zeros to the next power of 2. The padding is done using the `padMatrix` function, which takes a matrix and the new size (next power of 2) as input and returns a padded matrix.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

This equation above gives the recurrence for the running time of this algorithm, which has the solution: $T(n) = \Theta(n^3)$.

2.3 Strassen's Method

The Strassen's Method is implemented according to Thomas H et al. 2009 pages 79-82. Similar to the recursive approach, Strassen's algorithm also requires the matrices to be of size that is a power of 2. If the input matrices are not of this size, they are padded using the `padMatrix` function to the next power of 2.

The following equation is the recurrence for the running time $T(n)$ of Strassen's Algorithm which has the solution $T(n) = \Theta(n^{lg7})$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

2.4 Padding Technique

The padding technique is crucial for both the recursive and Strassen's method when dealing with matrices that are not exact powers of 2. The `padMatrix` function takes a matrix and the desired new size as input and returns a new matrix with the original matrix elements centred and the remaining elements filled with zeros. This padded matrix can then be used in the recursive and Strassen's algorithms without any issues.

The `unpadMatrix` function is used to remove the padding after the multiplication is complete, restoring the matrix to its original size.

By using these techniques, it ensures that the implementations of the recursive square matrix multiplication and Strassen's algorithm can handle matrices of any size.

3 Graphing Methodology

3.1 Matrix Generation

The matrices used in the experiment were randomly generated with integer values between 1 and 10, ensuring that the results were not biased by any specific matrix structure.

3.2 Dimensionality Range

The experiments were conducted on square matrices with dimensions ranging from (64×64) to (512×512), and up to (2048×2048). This range was chosen to observe the performance of each algorithm across different scales and to clearly indicate the asymptotic growth of each algorithm as the dimensions increase.

3.3 Matrix Count

For each dimension, three randomly generated matrices were utilised. This repetition allowed for the averaging of any outliers, resulting in a more precise measurement of the running times.

3.4 Correctness

To validate the correctness of the matrix multiplication algorithms, a correctness check was performed:

For each dimension, two random matrices (A) and (B) were generated. These matrices were padded as necessary for the Recursive and Strassen's method. The results for each algorithm were computed and then unpadded if padding had been applied. The resultant matrices for each algorithm were printed and visually inspected to ensure they matched. The correctness check ensures that the implementations of the Recursive and Strassen's algorithm produce the same results as the Square Matrix Multiplication algorithm, validating their correctness.

3.5 Experimental Procedure

The experiment was divided into two stages. In the first stage, the algorithms were tested on smaller dimensions, with matrix sizes ranging from 64 to 512, increasing by 32 at each step. The second stage focused on larger dimensions, ranging from 64 to 2048. Each matrix dimension was tested three times, and the final results were obtained by averaging the runtime across these three executions.

4 Results

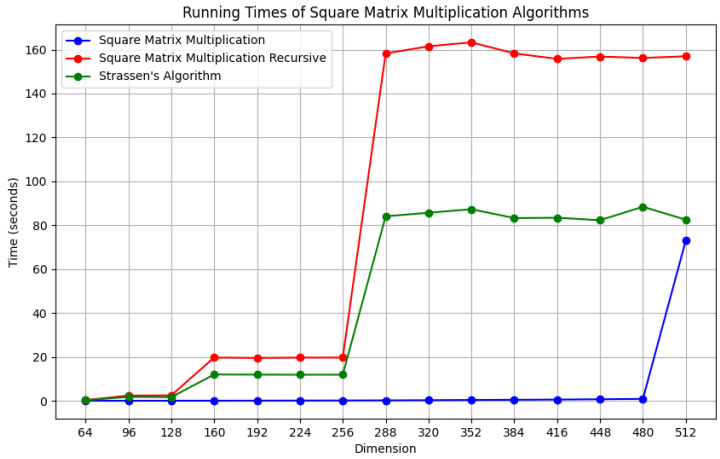

Figure 1: The run-times of the three square matrix multiplication algorithms for smaller dimensions: (64×64) to (512×512).

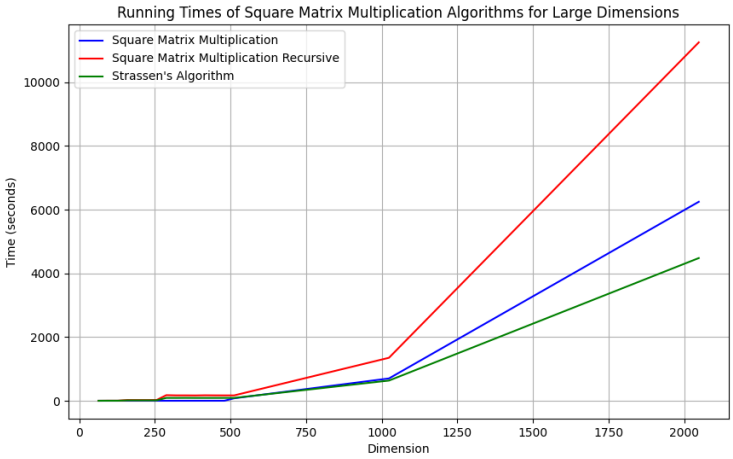

Figure 2: The run-times of the three square matrix multiplication algorithms for large dimensions: up to (2048×2048).

5 Analysis

The results are illustrated in Figures 1 and 2, which show the algorithms' performance for small and large dimensions, respectively.

5.1 Square Matrix Multiplication

The Square Matrix Multiplication algorithm exhibits a time complexity of $\Theta(n^3)$, which is evident from the plots. As the matrix dimension increases, the runtime increases cubically, consistent with the theoretical time complexity. For instance, as seen in Figure 1, the runtime for a matrix of dimension 64 is approximately 0.0016 seconds, while for a matrix of dimension 512, it spikes to around 73.04 seconds. This growth highlights the inefficiency of the algorithm as the matrix size increases, yet it performs efficiently for smaller dimensions due to the absence of overhead from recursive calls or additional operations.

In Figure 2, for larger matrices, the algorithm's cubic time complexity becomes more pronounced, with the runtime increasing significantly. This reinforces the understanding that while Square Matrix Multiplication is straightforward and efficient for small matrices, its utility diminishes as the matrix size grows.

5.2 Square Matrix Multiplication Recursive

The recursive version of the Square Matrix Multiplication algorithm, despite also having a time complexity of $\Theta(n^3)$, incurs additional overhead due to recursive calls and increased memory usage. As reflected in Figure 1, the runtime for dimension 64 is already higher at approximately 0.29 seconds compared to the non-recursive method. By the time the dimension reaches 512, the runtime escalates to nearly 157 seconds, significantly higher than the non-recursive method.

This overhead makes the recursive approach less practical for large matrices. The higher complexity and the cost associated with recursion lead to substantial performance degradation, which is further exacerbated as the matrix dimension increases, as seen in both figures. In Figure 2, the inefficiency of this approach is even more evident, with the algorithm becoming impractically slow for larger matrices, reflecting the drawbacks of the recursive implementation.

5.3 Strassen's Method

Strassen's Method, with its time complexity of approximately $\Theta(n^{2.81})$, offers a more efficient alternative, particularly for larger matrices. For smaller dimensions, as seen in Figure 1, Strassen's algorithm does not immediately outperform the non-recursive Square Matrix Multiplication. For example, at dimension 64, it runs in approximately 0.22 seconds, which is slower than the non-recursive method but faster than the recursive approach.

However, as the matrix size increases, Strassen's Algorithm demonstrates its strength. By dimension 512, it runs in about 82.5 seconds, significantly faster than the recursive method but still slower than the non-recursive method due to overhead and padding for non-power-of-two dimensions. In Figure 2, this algorithm's efficiency becomes more apparent for larger matrices, where its lower asymptotic complexity provides substantial performance gains over both the recursive and non-recursive methods, making it the most suitable choice for very large matrices.

6 Conclusion

The results clearly show that:

- Square Matrix Multiply is the fastest for small matrices due to its straightforward implementation and lack of overhead. However, it becomes inefficient for larger matrices as its cubic time complexity dominates.
- Square Matrix Multiply Recursive is generally slower due to the overhead from recursive calls and increased memory usage. This approach is not practical for large matrices as the performance degradation is significant, making it the least efficient of the three.
- Strassen's Method excels in larger matrices due to its lower asymptotic complexity, despite initial overhead. For smaller matrices, the algorithm is slower than the non-recursive Square Matrix Multiply due to the complexity of additional operations and padding. However, as matrix size increases, it becomes the most efficient, outperforming both other algorithms in large-scale matrix multiplication.

References

Thomas H, Cormen et al. (2009). *Introduction to Algorithms Third Edition*.