COMS4032A - Application of Algorithms Assignment 1

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Lisa Godwin - 2437980

August 14, 2024

Table of Contents

1 Overview of Algorithms

1.1 Classical Matrix Multiplication

The classical method of square matrix multiplication (SMM) involves computing each element of the resulting matrix as the dot product of a corresponding row from the first matrix and a column from the second matrix.
Time Complexity: $O(n^3)$.

1.2 Recursive Matrix Multiplication

This method applies a divide-and-conquer approach by dividing each matrix into four sub-matrices and recursively computing the product using these sub-matrices. It leverages the recursive structure to potentially optimise computations.
Time Complexity: $O(n^3)$.

1.3 Strassen's Matrix Multiplication

Strassen's algorithm is an optimised recursive approach that reduces the number of necessary multiplications. Instead of performing eight multiplications as in standard recursion, Strassen's method requires only seven, with additional operations for matrix addition and subtraction.
Time Complexity: Approximately $O(n^{2.81})$.

2 Handling Non-Power-of-Two Matrices

Both Recursive and Strassen's algorithms are designed for matrices whose dimensions are of the form 2^k. For matrices that are not powers of two, padding is applied to achieve dimensions that are powers of two. This is necessary to maintain efficient performance with these algorithms. For example, a matrix of size 500x500 is padded to 512x512 with zeros.
The performance analysis includes matrix sizes of:

12, 45, 69, 101, 154, 203, 317, 420, 487, 556, 678, 721, 889, 912, 1000, 1011, 1176, 1298, 1425, 1553, 1682, 1799, 1818, 1907, 2045, 3301, and 4089.

Note that Recursive was not run for matrix dimensions 3301 and 4089 due to excessive computation time. A wide range of matrices was chosen to ensure coverage beyond just powers of 2.

3 Interpretation of the Graphs

3.1 Execution Time vs. Matrix Size

This graph compares the performance of three matrix multiplication algorithms: Classical, Recursive, and Strassen, as matrix dimensions increase.

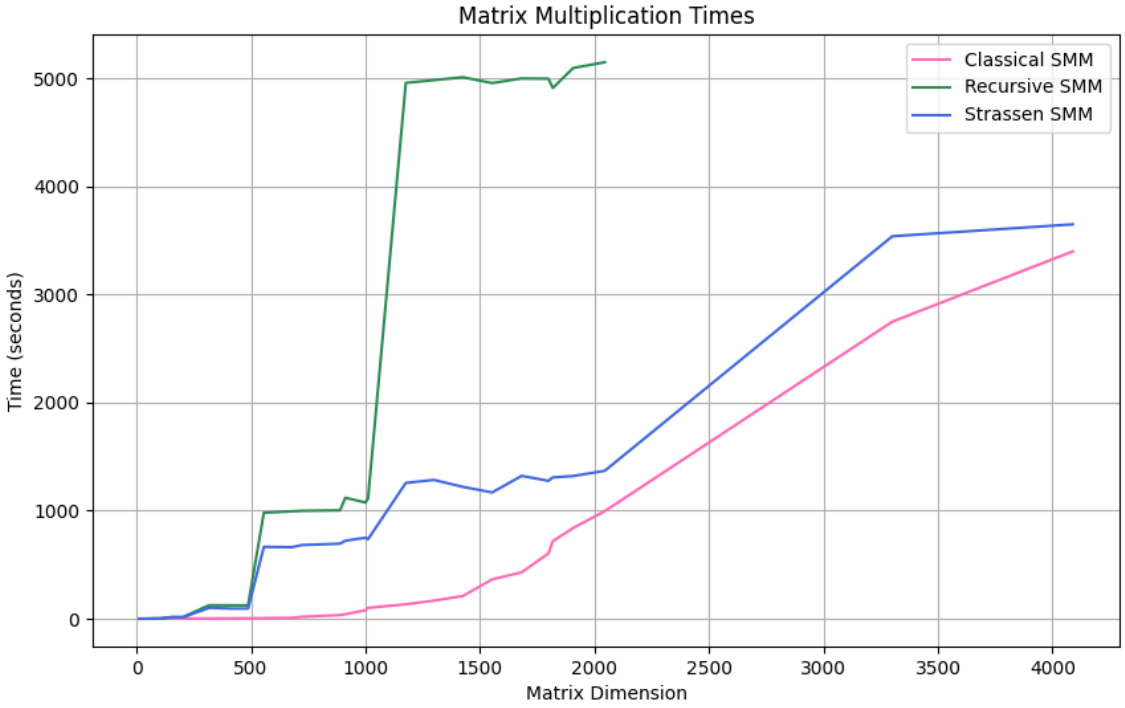

Figure 1: Execution time of Classical, Recursive, and Strassen's square matrix multiplication algorithms.

Key Observations:

- **Classical**: Begins with the lowest computation time and maintains a relatively steady increase as matrix dimensions grow. It remains the fastest algorithm for most matrix sizes.
- **Recursive**: Initially competitive, but experiences a sharp increase in computation time around a matrix dimension of 1000. It becomes significantly slower than both Classical and Strassen for larger matrices.
- **Strassen**: Starts with similar performance to Classical for smaller matrices but shows a steeper increase in time as dimensions grow. However, it begins to close the gap with Classical for matrices exceeding 2000.

Classical is the most efficient for small to medium-sized matrices. Strassen's algorithm shows potential for outperforming it as matrix dimensions continue to increase, especially in large-scale computations. As shown in Figure 1, Strassen's algorithm starts to approach similar computation times as Classical for larger matrices, suggesting that it could become more efficient than Classical for very large matrix sizes. Recursive becomes impractical for large matrices due to its significantly higher computation time.

3.2 Ratios vs. Matrix Dimensions

The analysis of time ratios for the Classical and Strassen's matrix multiplication algorithms provides insight into their relative performance across different matrix sizes. The time ratio is calculated as the ratio of computation times between successive matrix sizes, which helps to understand how the performance of each algorithm scales with increasing matrix dimensions.

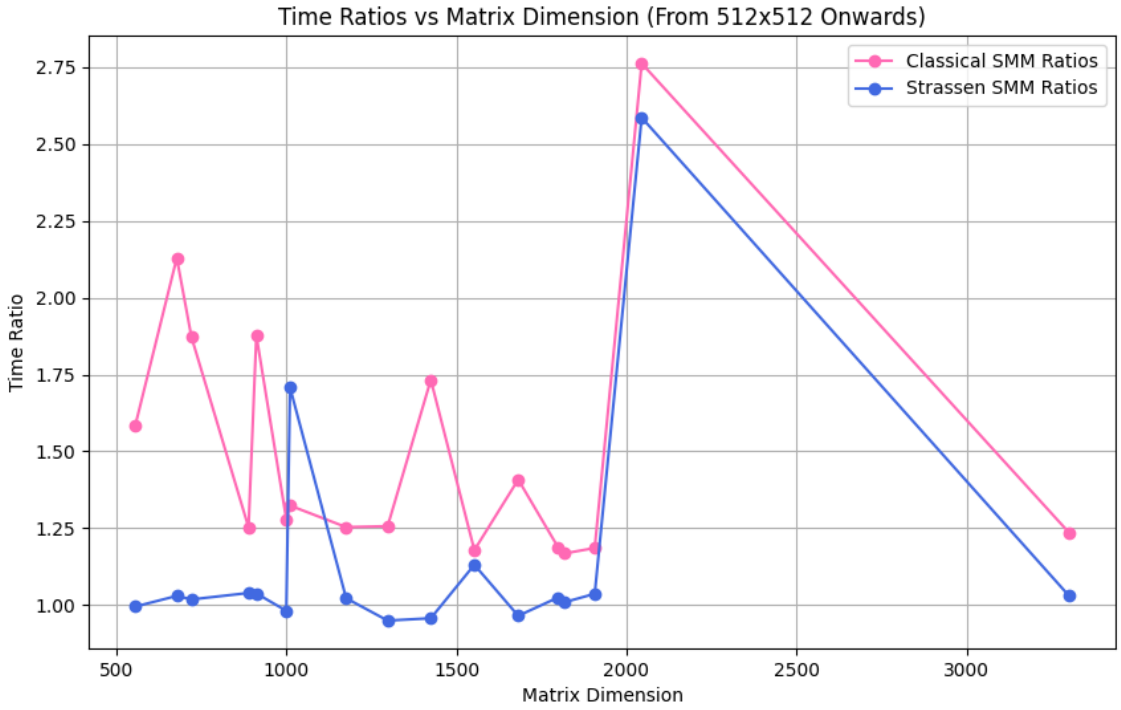

Figure 2: Time ratios of Classical and Strassen's matrix multiplication algorithms relative to matrix dimensions, starting from 512x512.

The analysis shows that Strassen's algorithm demonstrates consistent performance across matrices of sizes that are powers of 2 (2^k), maintaining ratios close to 1.0. This indicates that Strassen's efficiency remains relatively stable as matrix dimensions increase, though there are noticeable performance jumps when moving from one power-of-2 matrix size to the next.

In contrast, Classical SMM consistently exhibits increasing time ratios as matrix sizes grow, meaning it takes progressively longer to compute larger matrices. This steady increase in computation time for Classical highlights its less efficient scaling compared to Strassen's algorithm.

Given that Strassen's algorithm remains relatively efficient and performs better than Classical for larger matrices, it is expected that Strassen's algorithm will eventually outperform Classical for sufficiently large matrix sizes. The consistent performance of

Strassen's algorithm for larger 2^k matrices suggests that, as matrix dimensions continue to grow, its advantage over Classical will become more pronounced.

4 Conclusion

- **Classical**: This algorithm is most efficient for small to medium-sized matrices. It consistently performs well but shows a steady increase in computation time as matrix size grows. Classical remains the fastest among the three algorithms for smaller matrices, making it a suitable choice when matrix dimensions are relatively modest.

- **Strassen's Algorithm**: Strassen's algorithm exhibits promising performance for larger matrices. Although its computation time initially aligns closely with Classical for smaller matrices, it starts to demonstrate its efficiency advantage as matrix sizes increase. Figure 1 shows that Strassen's algorithm's execution time begins to approach that of Classical for larger matrices, suggesting that it could become more efficient for very large matrix sizes. The time ratios in Figure 2 support this, showing that Strassen's algorithm maintains relatively stable performance across matrices of power-of-2 sizes.

- **Recursive** : While initially competitive, Recursive becomes significantly slower for larger matrices, as indicated by the sharp increase in computation time. This makes it less practical for handling very large matrices compared to the other two algorithms.

Overall, Strassen's algorithm shows the most promise for large-scale computations, with its efficiency becoming more pronounced as matrix dimensions grow. The analysis suggests that Strassen's algorithm is likely to outperform Classical for sufficiently large matrices. Future work could focus on optimising Recursive further or exploring additional matrix multiplication techniques to handle a wider range of matrix sizes more effectively.