Introduction

Matrix multiplication is a prevalent topic in computer science because it is used in a widespread of applications, such as computer vision and model fitting. Therefore, a great amount of time and effort has gone into developing an efficient algorithm. Matrix multiplication is defined as

where is an matrix and is an matrix. denotes the value of at row and column . This approach is referred to as Iterative matrix multiplication. An alternative approach to multiplying two matrices is with recursion. Using this approach, we can represent matrix multiplication as

where

, , .

This approach is referred to as Recursive matrix multiplication. This algorithm uses the divide and conquer technique by dividing the problem into smaller sub-problems and combining, or conquering, the result of the sub-problems into the solution. This is recursively done until the instance size is sufficiently small where the result is trivial. There is yet another approach to solving matrix multiplication using a technique developed by Volker Strassen, which uses the idea of recursion but with seven multiplications rather than eight. This approach is defined as

.

It can be shown that this is equivalent to the Recursive definition. This approach is referred to as Strassen matrix multiplication.

This report will document the efficiency of these three algorithms (Iterative, Recursive and Strassen). To extract necessary data, each algorithm was implemented in python and ran using a 2.5 GHz Intel Core i5 processor with 4 GB 1600 MHz DDR3 RAM. Matrices, consisting of all ones, of sizes of 2x2, 4x4, 8x8, 16x16, 32x32, 64x62, 128x128, 256x256 and 512x512 were used as input instances to the algorithms. Matrices consisting of all ones was chosen for simplicity because the values are independent with the performance of the algorithm. 512x512 was the max instance size because 1024x1024 took far too long to compute ten times for each algorithm. For each algorithm and each instance size, the time the algorithm took to run was computed for ten executions. The maximum and minimum time was then removed from the data set, and the average was calculated for the remaining eight entries. This was done to remove outliers from the final calculations.

Results

The results of the experiment are shown in a tabular form in Figure 1, where the data entry for each algorithm and each matrix size is in seconds.

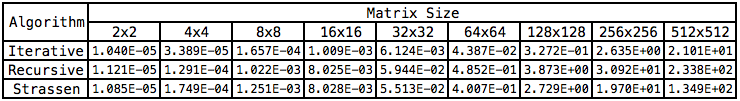


Figure 1 Results.

This figure allows for a simple comparison between algorithms for each input instance. For instance, the results show that matrix multiplication of two 2x2 matrices is computed almost instantaneously for each algorithm. For matrices of size 512x512, however, there is a clear order of efficiency, that being Iterative followed by Strassen followed by Recursive, from fastest to slowest.

Another representation of the results is shown in Figure 2.

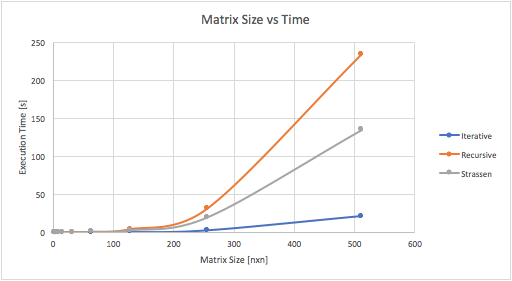


Figure 2 Graphical representation of the algorithm efficiency.

This figure shows a better representation of the growth trend for each algorithm. However, data is lost for the smaller instance sizes.

Analysis

The data from the experiment is surprising at first glance. This is because the time complexity for the algorithms indicates that the efficiency should be ordered, from most efficient to least efficient, as follows: Strassen, Iterative/Recursive. The time complexity of each algorithm is shown below.

The Recursive and Strassen time complexities are derived from the following recurrence relations, while the Iterative time complexity is straight forward (three loops from one to n).

The time complexity analysis alone would indicate that Strassen should be the most efficient, but there is more that must be considered.

First, the definition of Big-O is shown below.

Therefore, by saying that the Strassen algorithm is and the Iterative algorithm is alone, the and are missing from the discussion. That is, it is unclear at which point the algorithms are upper bound by and , respectively. For a sufficiently large there should be a clear indication on the graph where the Strassen algorithm would become more efficient than the Iterative algorithm for all . It is more than likely that the maximum instance size of 512x512 was not sufficient to indicate this transition. The costly operation that leads to this assumption is the method used to split the and matrices into four parts. For simplicity, it was chosen to create four new submatrices as their own data structure by looping the initial input matrix. This is of order , consequently causing the additional overhead in the recurrence relations.

Second, the space complexity must also be considered in the analysis. The space complexity of an algorithm is the amount of memory used. For example, the space complexity of the Iterative algorithm consists of only the input instances and a few looping variables. For the Recursive and Strassen algorithms, the stack continuously grows for each recursive call. This negatively affects the efficiency of the algorithm, as well as puts a constraint on the maximum instance size that the algorithm can handle. Since memory is finite, there exists some threshold for the amount of memory the process can have. If this is reached, an error will be thrown by the operating system, and the algorithm will abort.

Conclusion

Although the time complexity would indicate that Strassen would be the most efficient, the results indicate otherwise. Comprehensive analysis of Big-O and space complexity help explain why the results do not coincide with the expectations. Alternative approaches to the design of the Recursive and Strassen algorithms could improve efficiency but also improve design complexity. One idea is to pass the range of the matrices to the recursive calls so the range of the four submatrices can be determined without performing the costly operation of creating unique data structures.

The strengths of the experiment were the simplistic implementation of the algorithms and program as well as the consistency of the data. Although operating system constraints made it impossible to run a process to completion without additional processes being swapped on the CPU, maximum precautions were made to minimize this overhead. All additional programs were closed during execution and multiple runs were performed to compute the average, providing accurate results. The constraints of the experiment, on the other hand, were the acceptable instance sizes and the amount of memory of the system. Although ten executions of the Iterative algorithm could be performed with instance size of 1024x1024 in an acceptable amount of time, the Recursive and Strassen algorithms could not. Even so, algorithm execution with an instance size of 2048x2048 would take a rough estimate of at least a day to run. This is not realistic for the time frame of the experiment. Also, although this point was never reached, the process stack could have grown so large for the Recursive and Strassen algorithms that it exceeded the memory allocated for the process. This would have resulted in the process terminating before completion.

A future experiment would include a more efficient Recursive and Strassen algorithm where the submatrices were not computed but rather passed by index or range variables. Also, the experiment could be run on a faster system that could devote all its resources and time to the experiment. This would allow for larger instance sizes as well as less swapping of background processes.