

DESIGN AND ANALYSIS OF ALGORITHMS

Performance Analysis of Strassen Matrix Multiplication

15/April/2019

For Dr. Wasfi AlKhatib

|  |  |
| --- | --- |
| Group 15 | Work |
| Asaad AlGhamdi (201516850) | Strassen (60%), Manual Testing (100%) |
| Omar AlGhamdi (201528950) | Iterative (100%), Strassen (40%), Virtual Machine Testing (100%) |

Contents

[Implementation and Runtime 3](#_Toc6263473)

[Experiments and Performance Analysis 3](#_Toc6263474)

[Manual Testing Environment 3](#_Toc6263475)

[Analysis 4](#_Toc6263476)

[Automated Testing Environment 4](#_Toc6263477)

[Analysis 6](#_Toc6263478)

[Conclusion 7](#_Toc6263479)

# Implementation and Runtime

We implemented the straightforward iterative matrix multiplication in Matrix java class. Additionally, we implemented a set of auxiliary methods that perform matrix addition, subtraction, padding, trimming (of padded zeroes), and print. Strassen multiplication algorithm was implemented twice. We implemented Strassen initially using repetitive addition and subtraction and less recursive calls. Afterwards, we implemented a more efficient version of Strassen using fewer additions and subtractions and more recursive calls. Dimensions that are not power of two, were padded with the auxiliary matrix padding function using bitwise operation (n & (n-1)) == 0. Meaning, if n is a power of 2, the bitwise operation would return 0 otherwise 1. At the end, the padded matrix will be the closest power of 2 greater than n. In the same way we acquired the padded matrix, we would trim the zeroes before finally returning the Strassen multiplication result.

# Experiments and Performance Analysis

## Manual Testing Environment

Test Environment Specification: MSI Laptop Intel Core i7-8750H, 8 GB RAM, Windows 10 x64 Arch.

|  |  |  |  |
| --- | --- | --- | --- |
| n | Iter | Stras |  |
| 87 | 787756 | 3148013 |
| 107 | 1259501 | 2881671 |
| 294 | 44346779 | 163321583 |
| 2148 | 1.27E+11 | 6.349E+10 |
| 2614 | 2.27E+11 | 6.195E+10 |
| 2779 | 3E+11 | 5.634E+10 |
| 2819 | 3.2E+11 | 6.218E+10 |
| 3458 | 5.57E+11 | 6.217E+10 |
| 3624 | 7.31E+11 | 6.381E+10 |
| 3777 | 7.5E+11 | 6.223E+10 |
| 3784 | 8.14E+11 | 6E+10 |

|  |  |  |
| --- | --- | --- |
| n | Iter | Stras |
| 35 | 54910 | 50921 |
| 657 | 1.44E+09 | 1.4E+09 |
| 962 | 5.25E+09 | 1.37E+09 |
| 1337 | 2.72E+10 | 1.03E+10 |
| 1866 | 8.2E+10 | 9.76E+09 |
| 1891 | 8.65E+10 | 9.93E+09 |
| 2121 | 1.34E+11 | 7.21E+10 |
| 2379 | 2.01E+11 | 7.03E+10 |
| 2887 | 3.42E+11 | 6.97E+10 |
| 2972 | 3.49E+11 | 6.98E+10 |
| 3189 | 5.24E+11 | 7.13E+10 |
| 3447 | 6.07E+11 | 7.02E+10 |
| 1005 | 6.81E+09 | 1.23E+09 |

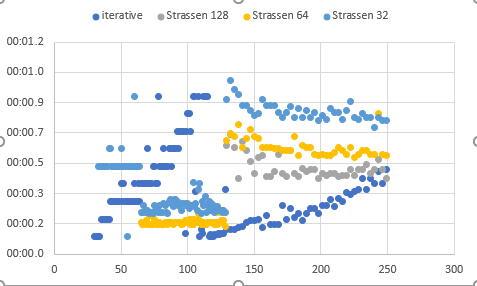
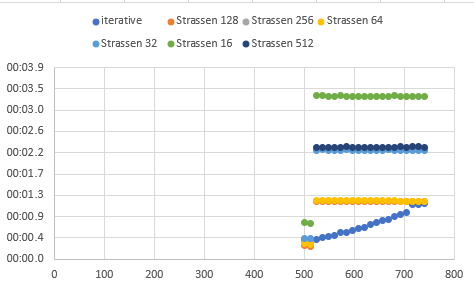
|  |  |  |
| --- | --- | --- |
| n | Iter | Stras |
| 116 | 1820105 | 1744789 |
| 322 | 60350543 | 60547367 |
| 1041 | 9.18E+09 | 1.9E+10 |
| 1601 | 4.24E+10 | 1.76E+10 |
| 1619 | 5.24E+10 | 1.65E+10 |
| 2287 | 1.6E+11 | 1.16E+11 |
| 2296 | 1.5E+11 | 1.17E+11 |
| 2624 | 2.56E+11 | 1.2E+11 |
| 2628 | 2.64E+11 | 1.22E+11 |
| 2835 | 3.42E+11 | 1.23E+11 |
| 3018 | 3.9E+11 | 1.18E+11 |
| 3082 | 3.96E+11 | 1.22E+11 |
| 3568 | 6.79E+11 | 1.3E+11 |

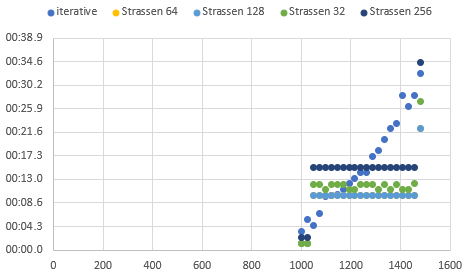
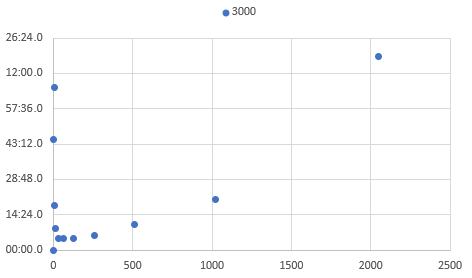
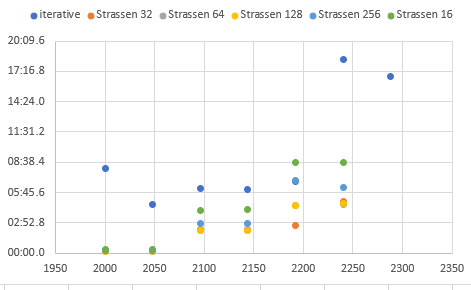
### Analysis

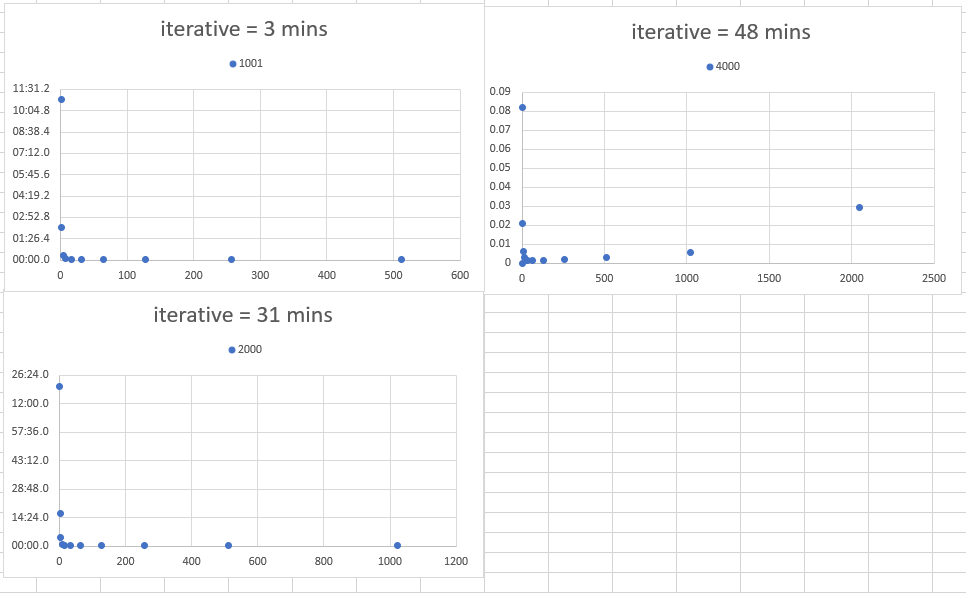
Strassen algorithm performs optimally with base 64 when n > 1500. Base 256 performs worse than base 64 in general but manages to outperform the iterative algorithm slightly earlier than base 64 when n>1300. Base 512 performs worse than base 64 and base 256 in general but outperforms the iterative algorithm earlier when n>1000. Evidently, the optimal base for Strassen would be 64. The performance of the two algorithm begins to deviate when n is approximately greater than 1000.

## Automated Testing Environment

The test was done on Azure cloud services using virtual machine of 4 processing units and 16GiB of memory. The graphs on the next page represent the results







### Analysis

in the case of matrices of size less than approximately 1300 \* 1300. The iterative version of matrix multiplication performed better than Strassen Algorithm no matter what the base case is. The reason that iterative algorithm performs better is because of Strassen algorithms tries to minimize the number of multiplications in the expenses of increasing the number of additions. The problem with that is real modern computers can multiply as fast as they do addition or subtraction. Iterative multiplication algorithm results in n3multiplications & n3-n2 additions. While Strassen algorithm results in nlg7 multiplications & 6nlg7-6n2. Combining these we conclude that the optimal multiplication algorithm is the one has smallest number of additions + multiplications.

In the case of matrix sizes greater than 1300 Strassen algorithm perform better than the iteration version when the base case is nether too small nor too large. Which results in fewer arithmetic operations than iterative multiplication or pure Strassen Algorithm (base=1). It also utilizes the memory hierarchy better because in the recursive step of Strassen Algorithm the size of matrix under multiplication halved which result in more cache hits without allocating lots of memory on the stack.

# Conclusion

To conclude, in general the preferred algorithm is the one that leads to minimum number of multiplications and additions. In our test cases, Strassen algorithm (base 64) is preferred over Iterative algorithm for multiplication of matrices with n approximately greater than 1000. Strassen algorithm generally performs matrix multiplications in the shortest time when the base case is 64 (Given same matrices). Therefore, the best base case would be 64. Strassen will perform worst when n is less than the base case.