Shivani Kushwaha

Algorithms for Bioinformatics

Programming Assignment Analysis

Data Structures: Justification, Implementation choices, and design decisions

The main data structures used in this assignment are lists. Utilizing list comprehensions can speed up operations, making them faster than conventional techniques. In this method, "for" loops are used to execute list multiplication. It is simple to iterate across lists with a "for" loop. The normal method for matrix multiplication is as follows.

- Consider the two matrices A and B:

$$A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$$
$$c_{12} = a_{11} \times b_{12} + a_{12} \times b_{12}$$
$$c_{21} = a_{21} \times b_{11} + a_{22} \times b_{21}$$
$$c_{22} = a_{21} \times b_{12} + a_{22} \times b_{22}$$

A * B = Matrix C (Matrix C is the result of matrix multiplication between A and B).

Matrix multiplication typically involves 4 formulas and 3 loops, resulting in 8 recursive calls. The time complexity in this case is basic $O(n^3)$. We still obtain an $O(n^3)$ runtime using the Master Theorem with $T(n) = 8T(n/2) + O(n^2)$. However, with Strassen method, we only require 7 calls overall with some additional addition and subtraction operations, rather than the usual 8 recursive calls.

The space complexity of a normal matrix multiplication is $O(n^2)$, where n is the dimension of the matrices being multiplied. After multiplying matrices, A and B, we need to create a new matrix C with dimensions n x n to hold the result. The memory required for normal matrix multiplication grows in a quadratic fashion based on the dimensions of the input matrices, therefore we might have to store temporary values throughout the operation.

Strassen's matrix algorithm:

- Strassen's method uses divide and conquer strategy for multiplication. The core objective of the Strassen algorithm is to divide the matrix multiplication task into smaller subsets that can be resolved iteratively.

- The Strassen algorithm divides each matrix into four n/2 x n/2 sub-matrices and uses a series of recursive operations to calculate the result rather than conducting the normal multiplication of two n x n matrices.

- When the matrix's dimensions are greater than 2, this technique will be quite effective at cutting down on time. It is often used for large matrices. The matrix can be partitioned into submatrices as follows.

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ \multicolumn{2}{c|}{A_{11}} & \multicolumn{2}{c}{A_{12}} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ \multicolumn{2}{c|}{A_{21}} & \multicolumn{2}{c}{A_{22}} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \quad B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ \multicolumn{2}{c|}{B_{11}} & \multicolumn{2}{c}{B_{12}} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ \multicolumn{2}{c|}{B_{21}} & \multicolumn{2}{c}{B_{22}} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

We can use the following formula to compute the matrices, where the algorithm performs only seven multiplications.

P1 = (a11 + a22) * (b11 + b22)
P2 = (a21 + a22) * b11
P3 = a11 * (b12 - b22)
P4 = a22 * (b12 - b11)
P5 = (a11 + a12) * b22
P6 = (a21 - a11) * (b11 + b12)
P7 = (a12 - a22) * (b21 + b22)

And four formulas:
C11 = A11 * B11 + A12 * B21
C12 = A11 * B12 + A12 * B22
C21 = A21 * B11 + A22 * B21
C22 = A21 * B12 + A22 * B22

Efficiency with respect to time and space

Using the Master Theorem to analyze the time complexity, we have:
$T(n) = 7T(n/2) + O(n^2)$
This results in a runtime of $O(n^{\log_2 7})$.

Time complexity of traditional matrix multiplication is $O(n^3)$, and time complexity for Strassen matrix is $\sim O(n^{2.809})$, which is better than the traditional matrix.

Strassen attempted to increase the algorithm's efficiency by reducing the number of multiplication operations, which are more time-consuming than addition and subtraction operations. He added more addition and subtraction operations as a result, which decreased the total number of multiplications.

The Strassen approach is theoretically faster than the normal algorithm, however, because of the overhead of the recursive function calls and the increased memory usage, it may not always be faster. Compared to standard multiplications, the Strassen algorithm has a higher space complexity of $O(n^{\log_2 (7)})$.

The total amount of storage space needed for storing the submatrices at each level of recursion is seven times that needed to keep the original matrices. Because the algorithm involves log2(n) stages of recursion, the overall space complexity is as follows:
Space complexity: O($n^{2.809}$)

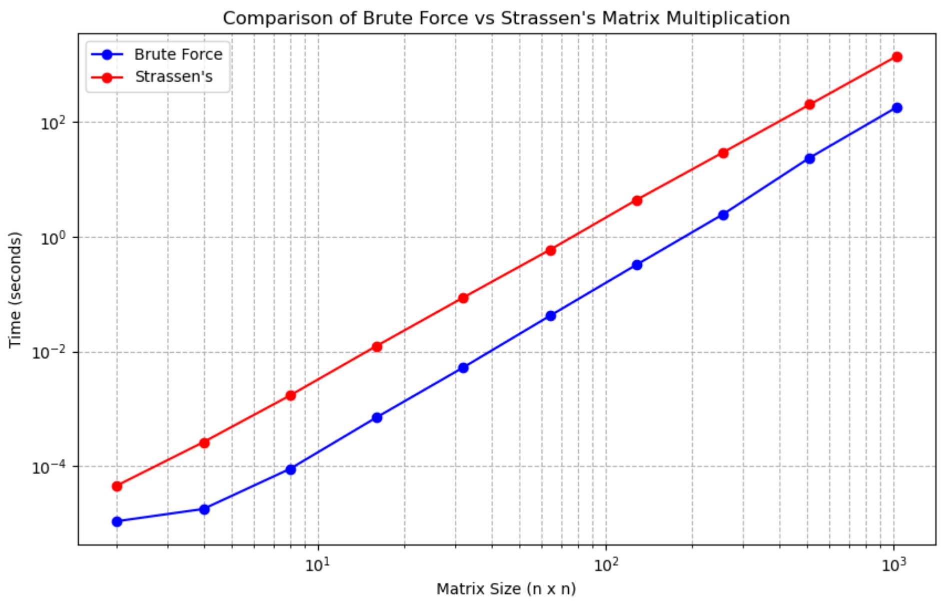
Output from the provided input:

Matrix Order	Strassen multiplication	Normal multiplication
2x2	7	8
4x4	49	64
8x8	343	512

Output from custom input:

Matrix Order	Strassen multiplication	Normal multiplication
2x2	7	8
4x4	49	64
8x8	343	512
16x16	2401	4096
64x64	117649	262144
128x128	823543	2097152

Observations with respect to asymptotic efficiency



Time comparison: Strassen’s algorithm vs Simple matrix

Matrix Order	Strassen Matrix Time	Normal Matrix Time
2x2	0.00005	0.00001
4x4	0.00035	0.00002
8x8	0.00274	0.00013
16x16	0.01746	0.00103
64x64	0.86794	0.06069
128x128	6.17051	0.48534

Comparison of theoretical efficiency to the observed efficiency

The theoretical efficiency did not match the observed efficiency. An algorithm's theoretical efficiency, as measured by its time complexity, gives an upper constraint on its growth rate in relation to input size. Real-world performance, on the other hand, can be influenced by a variety of factors such as overhead, memory utilization, optimization strategies, hardware characteristics, and numerical stability.

For instance, even though Strassen's matrix multiplication approach is theoretically faster than conventional matrix multiplication for big matrices, its performance in the actual world can vary due to aforementioned issues. In my test cases, I found that for matrices of order up to 128, Strassen's method took longer than the traditional matrix multiplication technique. While Strassen's algorithm is theoretically faster, the practical results from my tests did not align with this expectation, indicating a discrepancy between theoretical and observed efficiency.

What I learned and what I can do differently next time

Through this assignment, I learned that multiplications require greater computation than addition and subtraction do. The number of multiplications should be kept to a minimum to optimize processing time. Strassen's matrix multiplication has taken a lot of my time and effort to implement, but I think there is still opportunity to improve the algorithm's effectiveness. Furthermore, I could have used NumPy to generate views of sub-matrices with substantially lower overhead, this operation on matrices might have made the Strassen algorithm faster and simpler to implement. I'll spend more effort in subsequent iterations refining the functionality of my program.

Applicability to Bioinformatics

When thinking of bioinformatics, Strassen's matrix multiplication and other fast matrix multiplication algorithms may not be the first things that come to mind. But matrix operations are essential to lots of computations, and bioinformatics is no exception. Here's how quick matrix multiplication algorithms like Strassen's could be useful in bioinformatics:

Phylogenetic analysis: The evolutionary relationships between a group of species are represented by phylogenetic trees in evolutionary biology. Some approaches for building or analyzing these trees use matrix operations. These procedures can be accelerated, particularly when working with many species.

Genome-wide Association research (GWAS): GWAS research include examining big datasets to identify genetic variations that are linked to specific phenotypes. Some advanced statistical approaches employed in GWAS can benefit from quick matrix computations, especially when dealing with large populations and high-dimensional data.

Prediction of Protein Structure: A key challenge in bioinformatics is the prediction of proteins' three-dimensional structures from their amino acid sequences. Some methods, such as molecular dynamics simulations or energy minimization, rely on matrix operations that can be sped up by rapid matrix multiplication techniques.

Systems Biology: Systems biology uses a variety of mathematical and computational models to attempt to understand biological systems. These models can involve enormous sets of differential equations, the solutions to which may necessitate matrix operations.

Protein-protein interaction networks and gene regulatory networks, for example, can be shown as graphs using network analysis. Matrix operations are used in some graph algorithms and analysis, particularly those involving graph spectra.

Even though Smith-Waterman and Needleman-Wunsch, two common sequence alignment algorithms, don't directly utilize matrix multiplication, certain more sophisticated techniques or extensions might. For instance, some machine learning or kernel algorithms for sequence analysis may use matrix operations.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press

Fawzi, A., Balog, M., Huang, A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**, 47–53 (2022). https://doi.org/10.1038/s41586-022-05172-4

Uesaka, K., Oka, H., Kato, R., Kanie, K., Kojima, T., Tsugawa, H., Toda, Y., & Horinouchi, T. (2022). Bioinformatics in bioscience and bioengineering: Recent advances, applications, and perspectives. Journal of bioscience and bioengineering, 134(5), 363–373. https://doi.org/10.1016/j.jbiosc.2022.08.004

Li, R., Chang, C., Tanigawa, Y., Narasimhan, B., Hastie, T., Tibshirani, R., & Rivas, M. A. (2021). Fast numerical optimization for genome sequencing data in population biobanks. Bioinformatics (Oxford, England), 37(22), 4148–4155. https://doi.org/10.1093/bioinformatics/btab452

Sohail, A., & Arif, F. (2020). Supervised and unsupervised algorithms for bioinformatics and data science. Progress in biophysics and molecular biology, 151, 14–22. https://doi.org/10.1016/j.pbiomolbio.2019.11.012

Sompairac, N., Nazarov, P. V., Czerwinska, U., Cantini, L., Biton, A., Molkenov, A., Zhumadilov, Z., Barillot, E., Radvanyi, F., Gorban, A., Kairov, U., & Zinovyev, A. (2019). Independent Component Analysis for Unraveling the Complexity of Cancer Omics Datasets. International journal of molecular sciences, 20(18), 4414. https://doi.org/10.3390/ijms20184414