CISC 7200X – Analysis of Algorithms

Matrix Multiplications

Programming Project

Oct 27, 2020 – Dec 1, 2020

Objective: Assume $n = 2^k$ is a power of 2 for some $k \geq 0$. Let A and B be two $n \times n$ matrices each containing n^2 positive integers. Code a program that outputs the product $C = A \times B$.

Method: Implement the direct $\Theta(n^3)$-operations method and the Strassen $\Theta(n^{\log_2 7})$-operations recursive algorithm.

Programming language: You may use any programming language.

Input: A valid input is represented by a sequence of $2n^2 + 1$ numbers that appear in $2n + 1$ lines. The first line contains one number which is n the size of the matrices, the next n lines contain the n rows of A each with n numbers, and the last n lines contain the n rows of B each with n numbers.

Example: Below is the valid input for multiplying the following matrices

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{pmatrix} \times \begin{pmatrix} 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 \\ 19 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \end{pmatrix}$$

```
4
1  2  3  4
5  6  7  8
9  1  2  3
4  5  6  7
11 12 13 14
15 16 17 18
19 11 12 13
14 15 16 17
```

Input feeding: Two ways to enter the input should be implemented:

- An interface that allows iteratively typing valid inputs one after another. The next valid input may be typed after the output of the previous input is displayed.
- A data file containing the $2n + 1$ lines.

Remark: Assume that n is a power of 2 and that the input is valid.

Output:

- For small enough n, display all three matrices as $C = A \times B$.

$$\begin{pmatrix} 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

- Otherwise, output a file with n lines each contains one row of the product matrix

```
4 4 4 4
4 4 4 4
4 4 4 4
4 4 4 4
```

Performance evaluation I: Count the exact number of operations (multiplications and additions and total) made by both implementations for multiplying two all-ones matrices A and B for as large as you can $n = 2^k$.

- For each method, summarize your results in a table that has four columns for n, number of additions, number of multiplications, and total number of operations. Each table would have $k+1$ rows if you did the evaluation for $n = 2^i$ for $0 \leq i \leq k$. See below the first two rows of the table for the Strassen method.

n	#additions	$\#multiplications$	$\#operations$
1	0	1	1
2	18	7	25

- In addition, plot three charts (on a logarithmic or even double logarithmic scales) for the number of operations made by both methods as a function of n. The first chart is for the number of additions, the second is for the number of multiplications, and the third for the number total operations. Each chart replaces one column taken from both tables.

Performance evaluation II: Measure the "real time" it takes both methods to multiply two all-ones matrices A and B for as large as you can $n = 2^k$.

- If n is too small, repeat the same computation m times for a very large m, and then take the average of the total time over m as the running time.
- Summarize the results for the real-time it took both implementations in a table and a plot as you did for the total number of operations.
- Note that here there is no distinction between additions and multiplications.

Extra I: Based on the above, find the threshold h for which the direct method outperforms the Strassen algorithm for $n \leq 2^h$ while the Strassen algorithm outperforms the direct algorithm for $n > 2^h$.

- Implement a "hybrid" algorithm that stops the recursion once the sizes of the matrices reach 2^h and then apply the direct method.
- Add a third table or for the performance of the hybrid algorithm and/or add its performance to both performance to the chart.

Extra II: Implement the Strassen recursive algorithm for any value of $n \geq 1$.

- Demonstrate the correctness of your implementation by showing, for $n = 5, 7, 9$, the outputs generated by both methods for some examples of your choice.
- Here, you do not need to do any performance evaluation.

Testing the code: Once you are ready, send an email to amotz@sci.brooklyn.cuny.edu. This email should contain the following:

- Instructions for how to test your code with both ways for entering valid inputs.
- The portion of the code that implements both methods.

Video meeting: In addition to the email, we will have a 1/2-hour video meeting in which you show me how your project works and explain to me your implementation.

Grading: You lose nothing by trying to complete the project because your grade will be 100 regardless of how well you do. For those who do well, the percentage by which the project will count in their final grade could be as high as 15% depending on the factors listed below. On the other hand, if you get it entirely wrong, the percentage could be zero. In any case, it will not hurt you to try, it can only help improve your grade.

Grading main factors:

- Correctness.
- Accuracy in implementing the algorithms.
- Friendly interface for the Input and clarity of the displayed Output.
- Doing the extras.
- Unexpected initiatives shown by the students!