Strassen's Algorithm

- I will first explain the intution behind this algorithm:
  - The base case would always be 2-matrices of size 2 being multiplied → this would mean our large matrices would have to be powers of 2.

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

⇓ by traditional matrix multiplication we have

$$\begin{aligned} c_{11} &= a_{11} \times b_{11} + a_{12} \times b_{21} \\ c_{12} &= a_{11} \times b_{12} + a_{12} \times b_{22} \\ c_{21} &= a_{21} \times b_{11} + a_{22} \times b_{21} \\ c_{22} &= a_{21} \times b_{12} + a_{22} \times b_{22} \end{aligned}$$

⇒ notice how we have to do 8 multiplication operations

⇓ This brings us to the first optimization

we will instead construct c_{11}, c_{12}, c_{21} & c_{22} using fewer multiplications namely $\boxed{7}$. We are doing this because multiplication is more expensive than addition.

$$\begin{aligned} M_1 &= (a_{11} + a_{12}) \times (b_{11} + b_{22}) \\ M_2 &= (a_{21} + a_{22}) \times b_{11} \\ M_3 &= a_{11} \times (b_{12} - b_{22}) \\ M_4 &= a_{22} \times (b_{21} - b_{11}) \\ M_5 &= (a_{11} + a_{12}) \times b_{22} \\ M_6 &= (a_{21} - a_{11}) \times (b_{11} + b_{12}) \\ M_7 &= (a_{12} - a_{22}) \times (b_{21} + b_{22}) \end{aligned}$$

⇒ Notice how only 7 multiplication operations instead of 8 are present.

F1 = Formula 1 ***

papergrid
Date: / /

=> Now using the ~~matrices~~ values $M_1, M_2, M_3, M_4, M_5, M_6$ & M_7 we compute matrix C.

F1 =>
$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1+M_4-M_5+M_7 & M_3+M_5 \\ M_2+M_4 & M_1-M_2+M_3+M_6 \end{bmatrix}$$

* Now that we have established how the base case can be calculated optimally

⇓

we will use a **divide and conquer approach** to divide large matrices into 2x2 matrices and combine them to get back our final answer.

ex: We will show multiplication for 4x4 matrices now:

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

(A_{11}, A_{12}, A_{21}, A_{22} and B_{11}, B_{12}, B_{21}, B_{22} mark the corresponding blocks)

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$

$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$

$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$

↳ we will evaluate this using formula F1 to reduce the number of multiplications.

papergrid

Date: divide n conquer.

Hence, we can now define the pseudo-code for our algorithm:

```
Algorithm MM(A, B, n)
{
  if (n <= 2)
  {
Step 0    C => Computed from C11, C12, C21 & C22
  }
  else
  {
    mid = n/2
Step 1 => MM(A11, B11, n/2) + MM(A12, B21, n/2)   Label this as
Step 2 => MM(A11, B12, n/2) + MM(A12, B22, n/2)   MM1, MM2, MM3
Step 3 => MM(A21, B11, n/2) + MM(A22, B21, n/2)   MM4, MM5, MM6
Step 4 => MM(A21, B12, n/2) + MM(A22, B22, n/2)   MM7, MM8
  }
}
```

⇒ So now that we have defined our serial algorithm, we can look for opportunities for parallelism:

* Firstly we notice that step 1, step 2, step 3 and step 4 at the same level in the recursive tree could be executed parallelly
* Furthermore lets consider step 1 for example we can execute MM(A11, B11, n/2) & MM(A12, B21, n/2) parallely, following which we would add these two only after they have successfully completed
⇒ Hence, each step despite being parallely executed would have to be synchronized ~~so that the~~ in itself so that the additions occur correctly.

* Now we analyze the parallelism in Step 0 using formula (1):

Next page

papergrid

Date: / /

$$\boxed{F_1} \Rightarrow \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1+M_4-M_5+M_7 & M_3+M_5 \\ M_2+M_4 & M_1-M_2+M_3+M_6 \end{bmatrix}$$

$\Rightarrow$ We can compute C_{11}, C_{12}, C_{21} & C_{22} in parallel after we have successfully determined $M_1, M_4, M_5, M_7, M_3, M_2$ & M_6

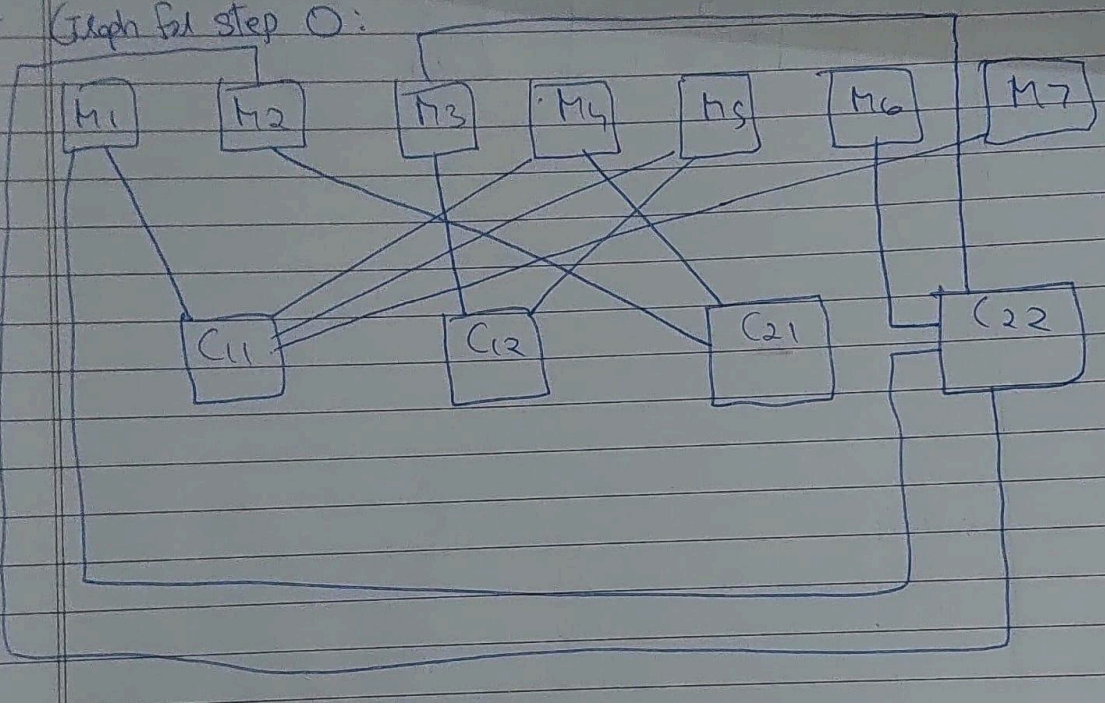
$\Downarrow$

$\Rightarrow$ Furthermore each of the M_is can be computed in parallel. This completes our discussion on parallelism. We can begin coding.

papergrid

Date: / /

- Graph for step 0:



- **Apart from the parallelism shown above, I have parallelized the matrix Addition, Subtraction, and also the partition and combine operations in the matrices, as they can also be done parallely for each element.**

Instructions to run the code:

For the pthreads code:

gcc -o strassen_pthreads strassen_pthreads.c -lpthread

./strassen_pthreads

For the openMP code:

gcc -fopenmp strassen_openmp.c -o strassen_openmp

./strassen_openmp

Results Obtained (present in the files appropriately labelled:

For Pthreads:

```
Enter the dimension of the matrices (n x n), where n is a power of 2: 4
Populate matrix A:
Enter the elements for a 4x4 matrix:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Populate matrix B:
Enter the elements for a 4x4 matrix:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16

1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16

Result of Strassen's Multiplication:
90      100     110     120
202     228     254     280
314     356     398     440
426     484     542     600
```

```
Serial Time: 0.00900300
Parallel Time: 0.00893400
Speedup: 1.00772330
```

For OpenMP:

```
Enter the dimension of the matrices (n x n), where n is a power of 2: 4
Populate matrix A:
Enter the elements for a 4x4 matrix:
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
Populate matrix B:
Enter the elements for a 4x4 matrix:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1       2       3       4
1       2       3       4
1       2       3       4
1       2       3       4

1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16

Result of Strassen's Multiplication:
90      100     110     120
90      100     110     120
90      100     110     120
90      100     110     120

Serial Time: 0.00035600
Parallel Time: 0.00013800
Speedup: 2.57971014
```

Speed up obtained in the case of OpenMP is clearly higher and it is expected because the OpenMP library optimizes at the compiler level, and it handles various other aspects of parallelizing underneath on its own.

For the further details regarding the implementation of the code, kindly refer to the code, I have commented it appropriately to under