PARALLEL ALGORITHM FOR MATRIX MULTIPLICATION

Presented by: Anuj Shah

Guided by:

Professor Dr. Russ Miller
Dr. Matt Jones

University at Buffalo The State University of New York

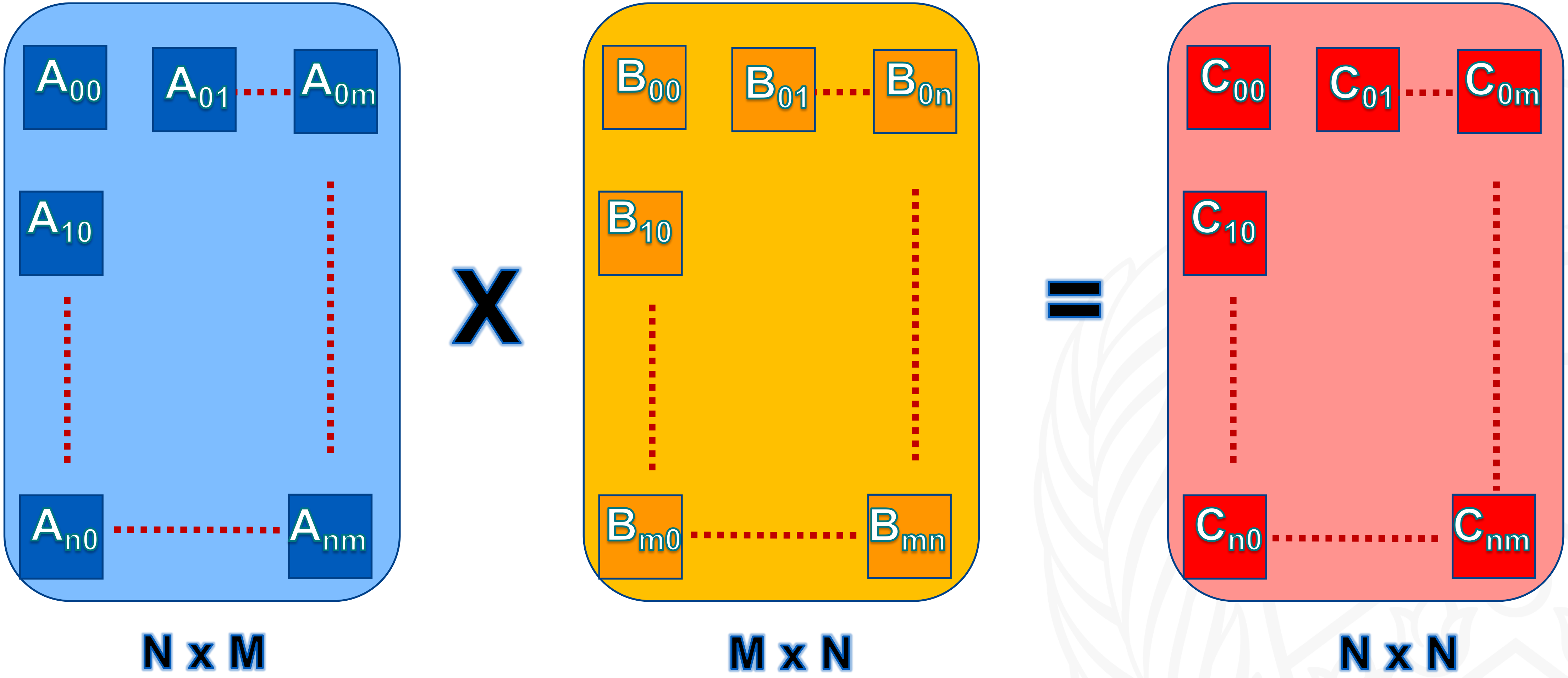
AGENDA

- Problem Definition
- Applications of Matrix Multiplication
- Parallel Implementation
- Results
- Challenges Faced
- Future Work
- Conclusion

Problem Definition

- Given a matrix A(n x m) n rows and m columns, where each of its elements is denoted A_{ij} with 1 ≤ i ≤ n and 1 ≤ j ≤ m, and a matrix B(m × p) of m rows and p columns, where each of its elements is denoted B_{ij} with 1 ≤ i ≤ m, and 1 ≤ j ≤ p, the matrix C resulting from the operation of multiplication of matrices A and B, C = A × B, is such that each of its elements is denoted C_{ij} with 1 ≤ i ≤ n and 1 ≤ j ≤ p, and is calculated follows

$$C_{r,c} = AB_{r,c} = \sum_{i=1}^{n} A_{r,i} * B_{i,c}$$



Matrix C has a total of N^2 entries $\Theta(n^2)$

Each of them require:

N Multiplications & n-1 Additions $\Theta(n)$

Total time = $\Theta(n^3)$

Matrix Multiplication

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 5 \\ 6 & 7 \\ 1 & 8 \end{bmatrix}$$

Sequential Algorithm

```
for (i = 0; i< n; ++i)
    for (j = 0; j< n; ++j)
        c[i][j] = 0;
        for(k =0; k<n; ++k)
            c[i][j] = a[i][k] * b[k][j];
        end for
    end for
end for
```

Applications of Matrix Multiplication

A few of them are:

- Recurrence Relations
- Physics
- Graph theory Problems
- Video Games
- Robotics

Parallel Implementation

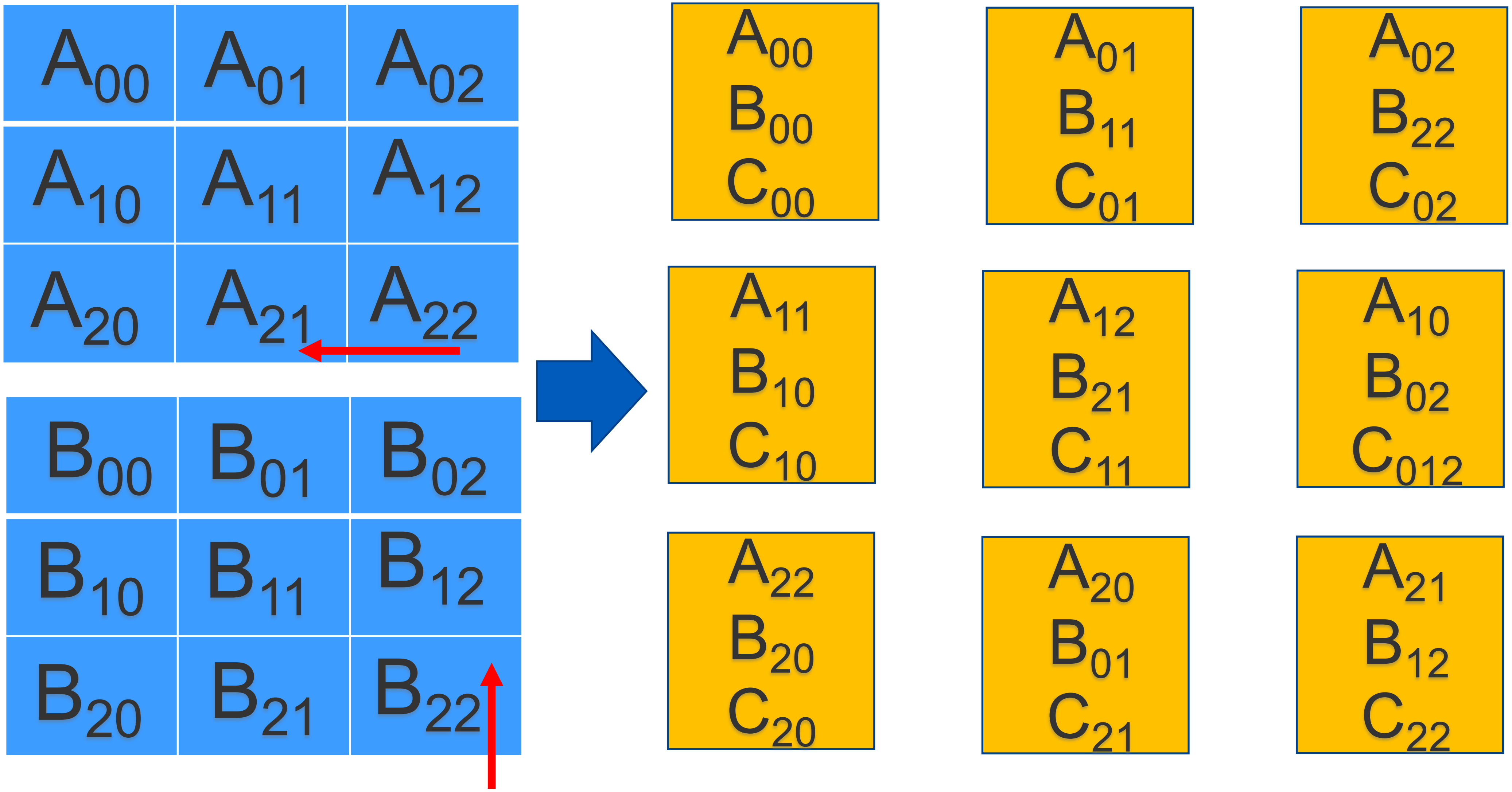
1. Partition these square matrices in p square blocks, where p is the number of processes available.

2. Create a matrix of processes of size $P^{1/2}$ x $P^{1/2}$ so that each process can maintain a block of A matrix and a block B matrix.

3. Each Process works with it respective sub block.

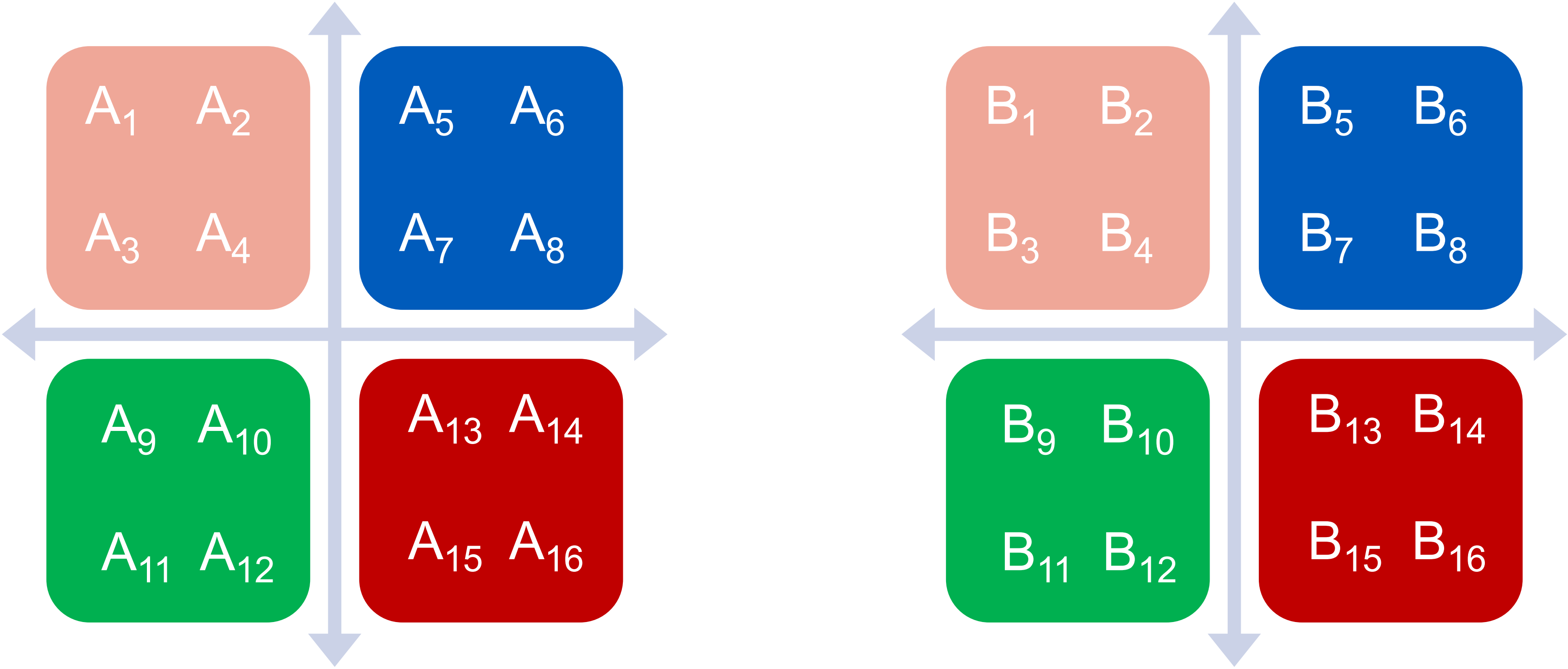
4. Initial arrangement is done with respect to the PEs such that each sub block of A is shifted to the left by its row number and each sub block of B is shifted up by its column number.

5. Repeat $\sqrt{p}$ times
   1. Perform Matrix Multiplication in each processor and add the result to the previous one.
   2. The sub-blocks of A are shifted one step to the left and the sub-blocks of B are shifted one step up.

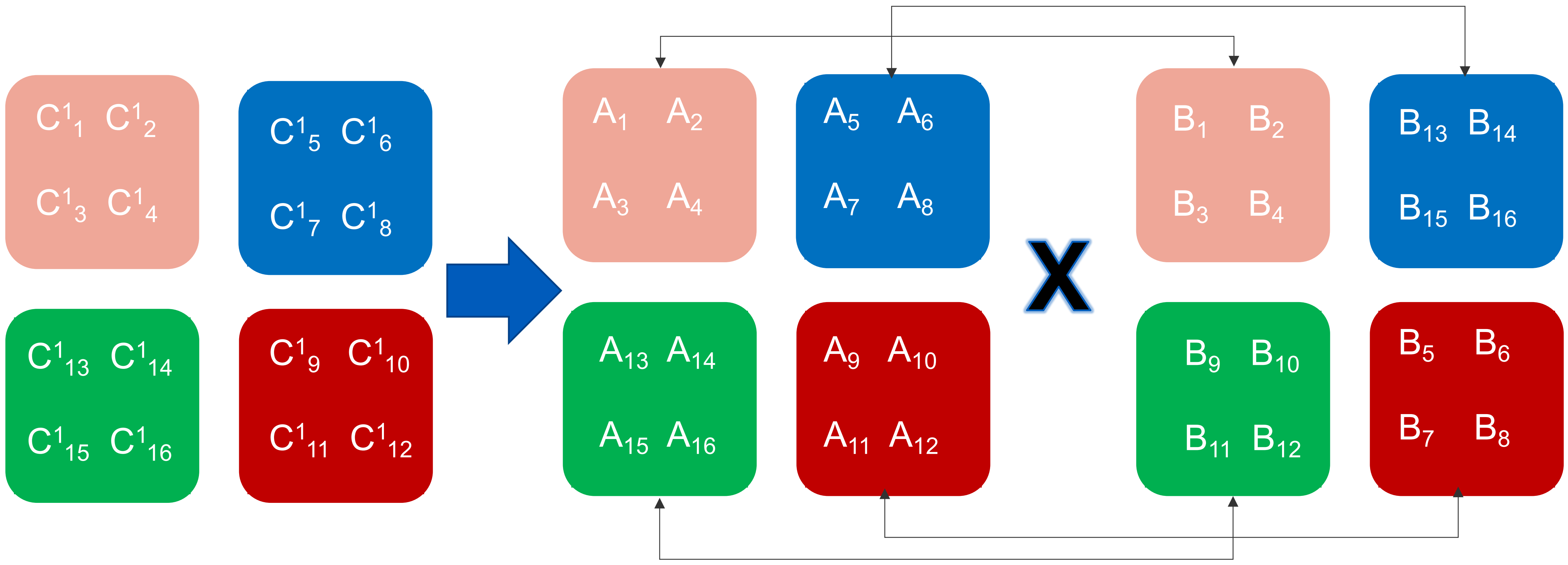


A_{00} A_{01} A_{02}
A_{10} A_{11} A_{12}
A_{20} A_{21} A_{22}
B_{00} B_{01} B_{02}
B_{10} B_{11} B_{12}
B_{20} B_{21} B_{22}
A_{00} B_{00} C_{00}
A_{01} B_{11} C_{01}
A_{02} B_{22} C_{02}
A_{11} B_{10} C_{10}
A_{12} B_{21} C_{11}
A_{10} B_{02} C_{012}
A_{22} B_{20} C_{20}
A_{20} B_{01} C_{21}
A_{21} B_{12} C_{22}

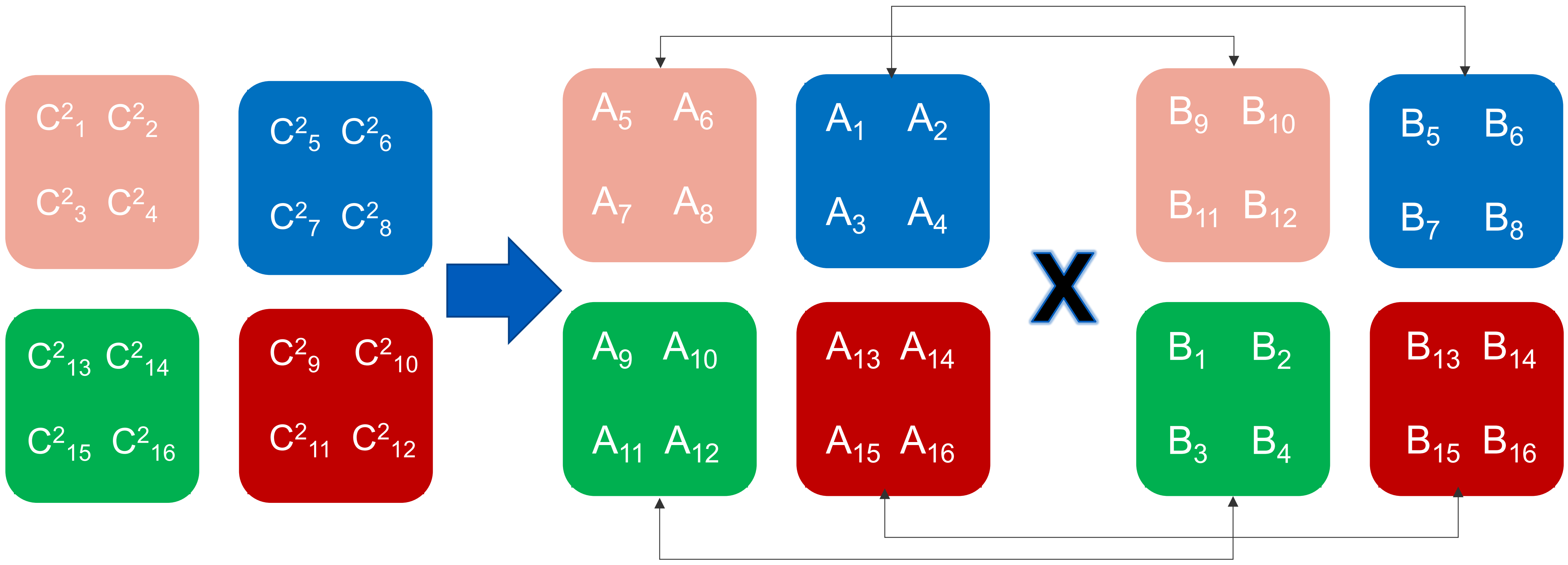
Initial Matrices being divided into 4 blocks and given to their processes:



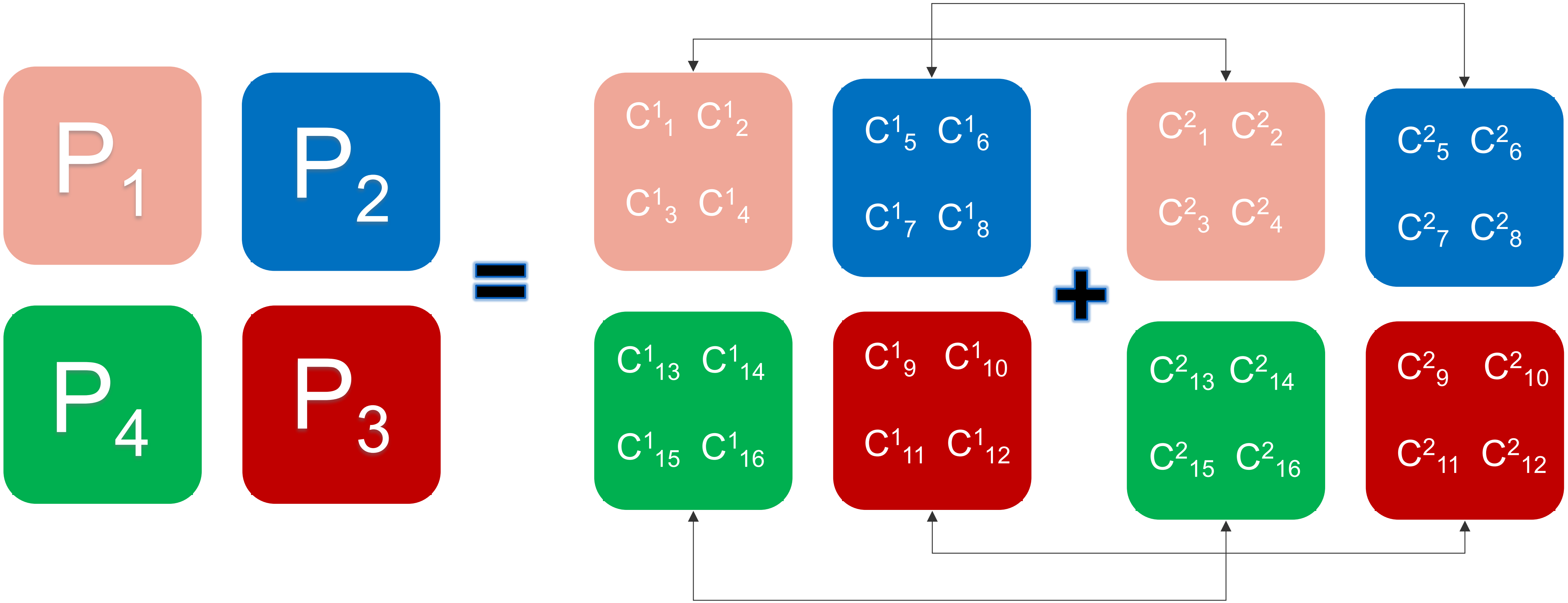
Initial arrangement & local multiplication:



Shift A left by one and B up by 1 & local Multiplication



Add the partial answers



Results

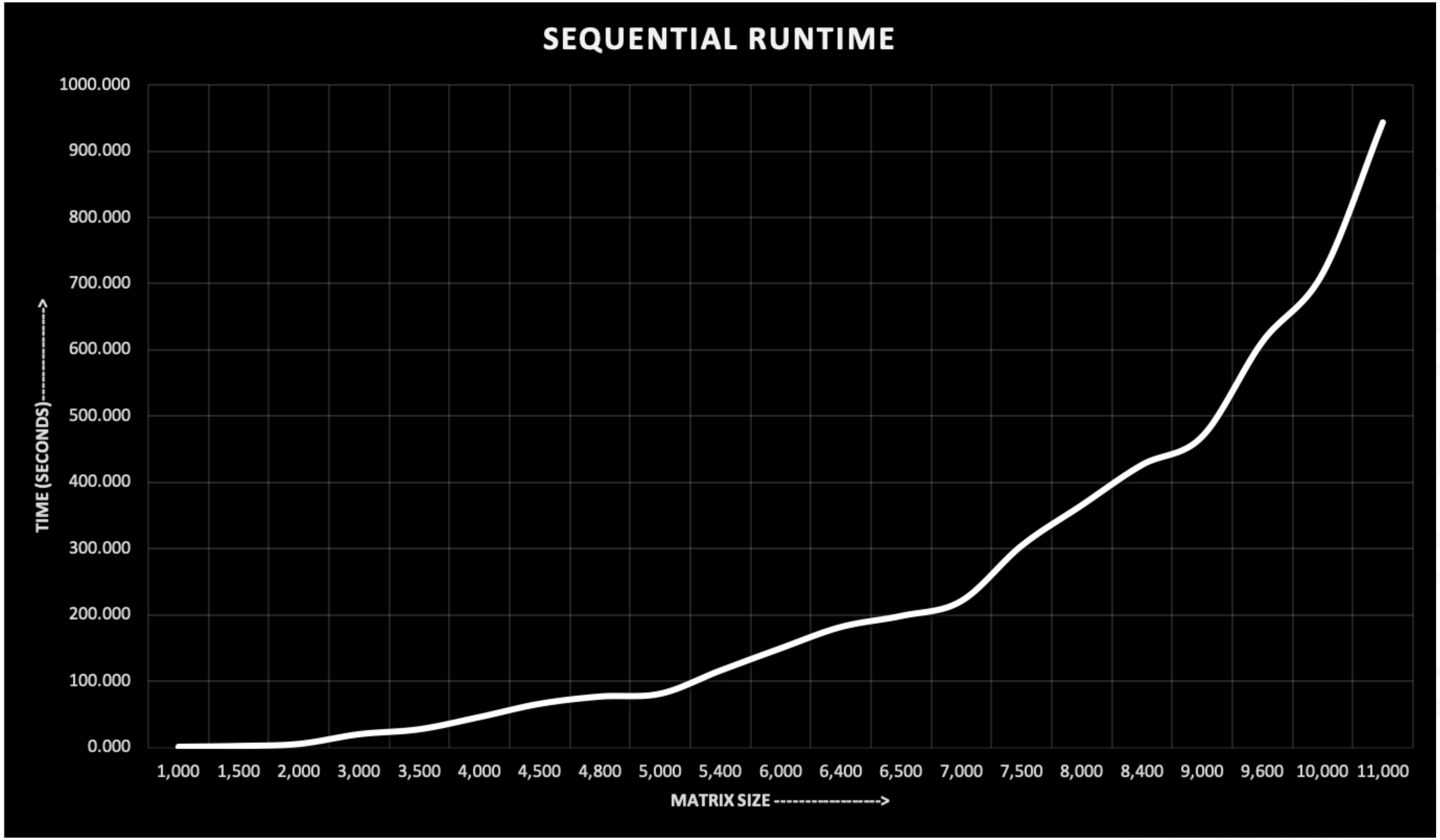
The final testing Parameters were as followed:

- Matrix dimensions ranged from 1000 to 18000.
- Both matrices were square and had the same dimensions.
- No. of processors used were 1, 4, 9, 25, 49, 81, 100, 144, 225 & 256.
- Each run was performed on Processors having 16-core nodes and 128GB of memory.

Sequential run

DATA	TIME(S)
1,000	0.518
1,500	1.751
2,000	4.878
3,000	19.497
3,500	27.031
4,000	45.462
4,500	65.480
4,800	76.380
5,000	80.716
5,400	115.722

DATA	TIME(S)
6,000	149.274
6,400	181.407
6,500	198.058
7,000	220.718
7,500	304.095
8,000	365.351
8,400	426.185
9,000	469.025
9,600	612.037
10,000	715.185
11,000	943.285


SEQUENTIAL RUNTIME
1000.000
900.000
800.000
700.000
600.000
500.000
400.000
300.000
200.000
100.000
0.000
TIME (SECONDS)------------->
1,000
1,500
2,000
3,000
3,500
4,000
4,500
4,800
5,000
5,400
6,000
6,400
6,500
7,000
7,500
8,000
8,400
9,000
9,600
10,000
11,000
MATRIX SIZE ------------------>

Parallel run

No of Processors : 4

DATA	TIME(S)
1,000	0.189
2,000	1.2082
5,000	21.471
7,000	59.600

No of Processors : 9

DATA	TIME(S)
1500	0.241
3000	1.744
5400	12.295
9000	60.977

No of Processors : 25

DATA	TIME(S)
4,000	2.15
6,000	6.31
8,000	13.751
10,000	25.9

No of Processors : 49

DATA	TIME(S)
3,500	0.783
7,000	11.248
8,400	11.594
14,000	51.61

No of Processors : 81

DATA	TIME(S)
4,500	0.942
6,300	2.407
7,470	4.66
9,000	9.103

No of Processors : 100

DATA	TIME(S)
4,500	0.923
6,500	2.347
7,500	3.559
11,000	13.812

No of Processors : 144

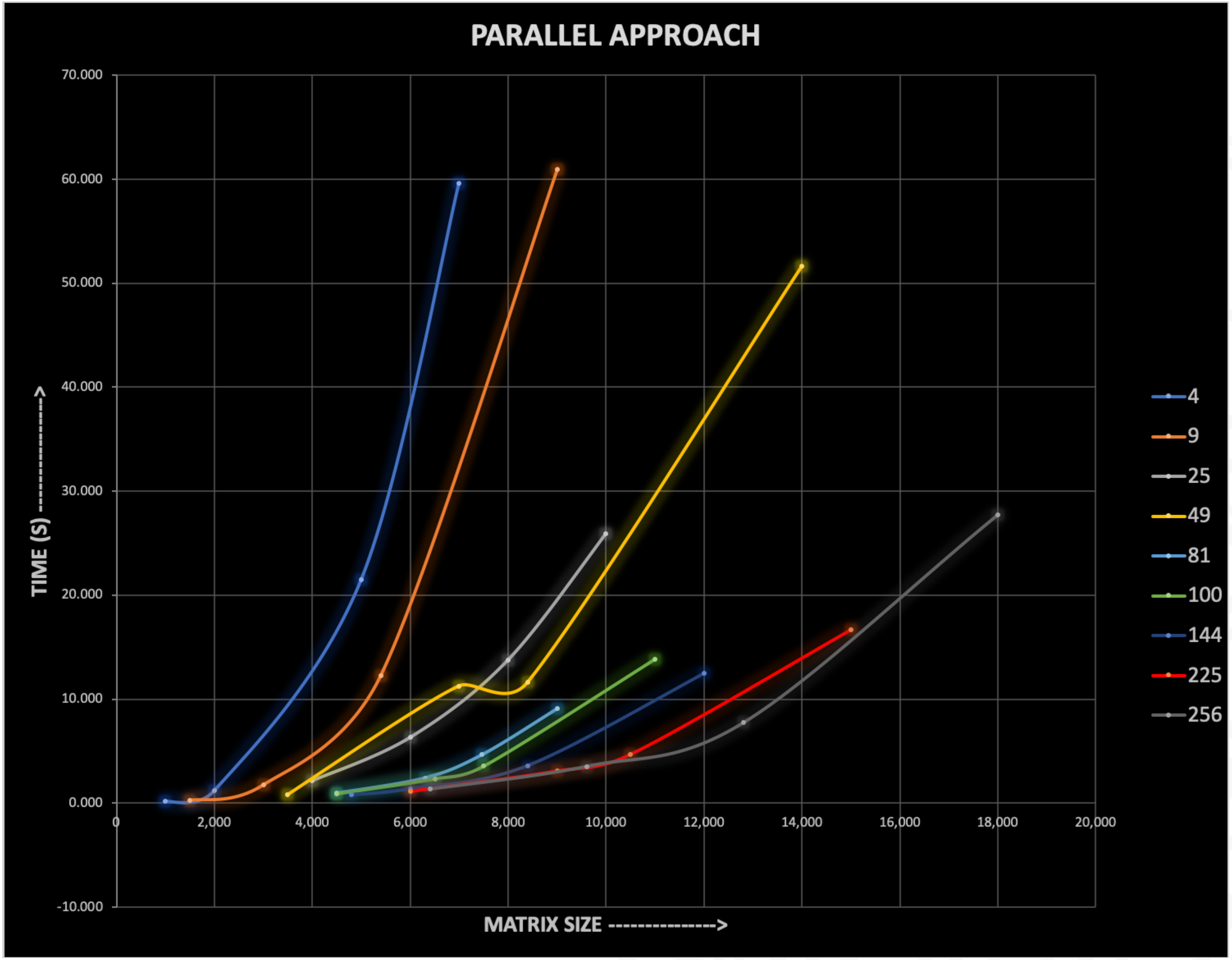
DATA	TIME(S)
4,800	0.789
6,000	1.362
8,400	3.605
12,000	12.509

No of Processors : 225

DATA	TIME(S)
6,000	1.129
9,000	3.094
10,500	4.691
15,000	16.655

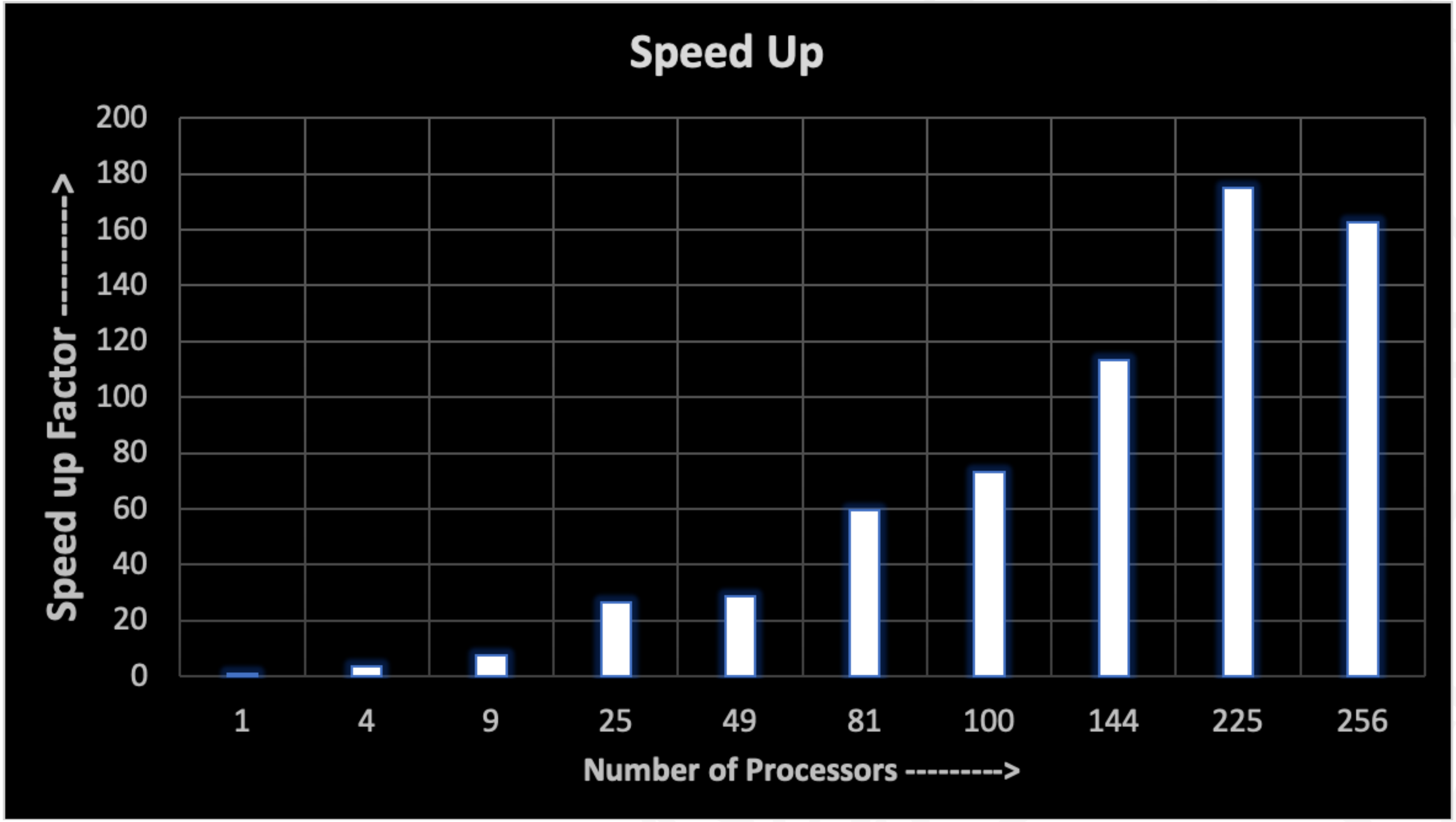
No of Processors : 256

DATA	TIME(S)
6,400	1.354
9,600	3.531
12,800	7.721
18,000	27.707



Speed Up Factor

Number of Processors	Speed Factor
1	1
4	3.72
9	7.34
25	26.51
49	28.53
81	59.61
100	73.20
144	113.26
225	175.18
256	162.44



Challenges Faced

- The number of processors must be a proper square.
- The data should be equally distributed amongst all the processors.
- The results of running time varies with the change of processor specification and their allocation.

Future Work

- Can use files to read and write data.
- Use of Strassen's algorithm for sequential matrix multiplication.
- Compare the performance results using OpenMP.

Conclusion

- It is not always worth taking up the additional cost of extra processors vs the speed up achieved.
- The decision highly depends on the task requirements and incoming data
- Increasing the number of processors doesn't always speed up the process.
- In my opinion for a matrix of size up to 11k * 11k one should go for 25 Processors ("sweet point").

Questions??

Thank You!