# Parallel Matrix Multiplication

Presented By: Prasad Shirvandkar

CSE 708

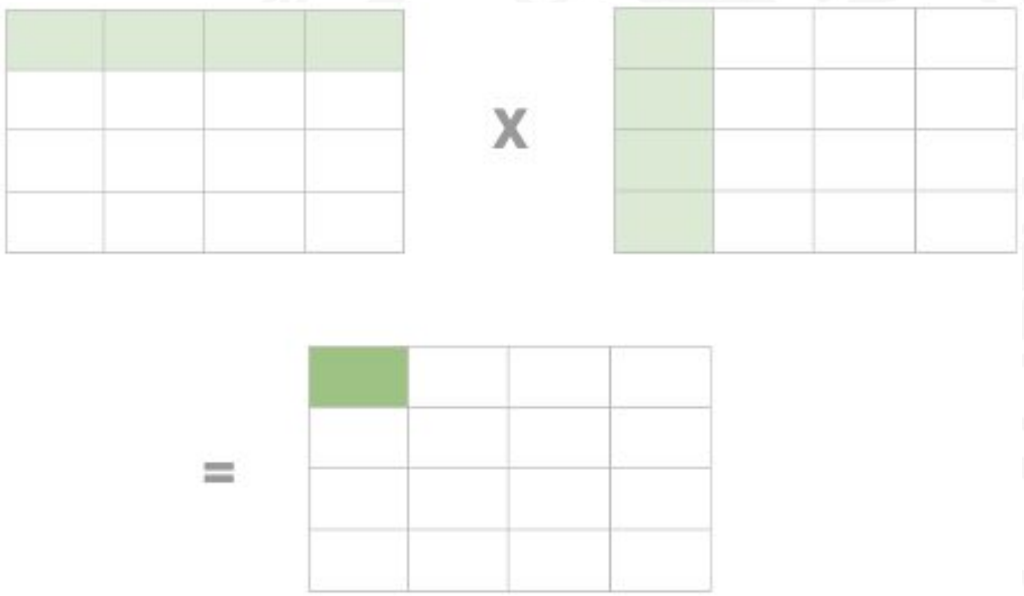
University at Buffalo The State University of New York

# Problem Statement

- Given two matrices of with matrix A being size m **x** n and another matrix B being of n **x** k
- Return Product matrix C with size m x k i.e. A x B

$$C = A_{i1}B_{1j} + A_{i2}B_{2j} + \dots\dots\dots + A_{in}B_{nj} = \sum_{m=1}^{n} A_{ik}B_{kj}$$

where i = 1 … m, j = 1 … k



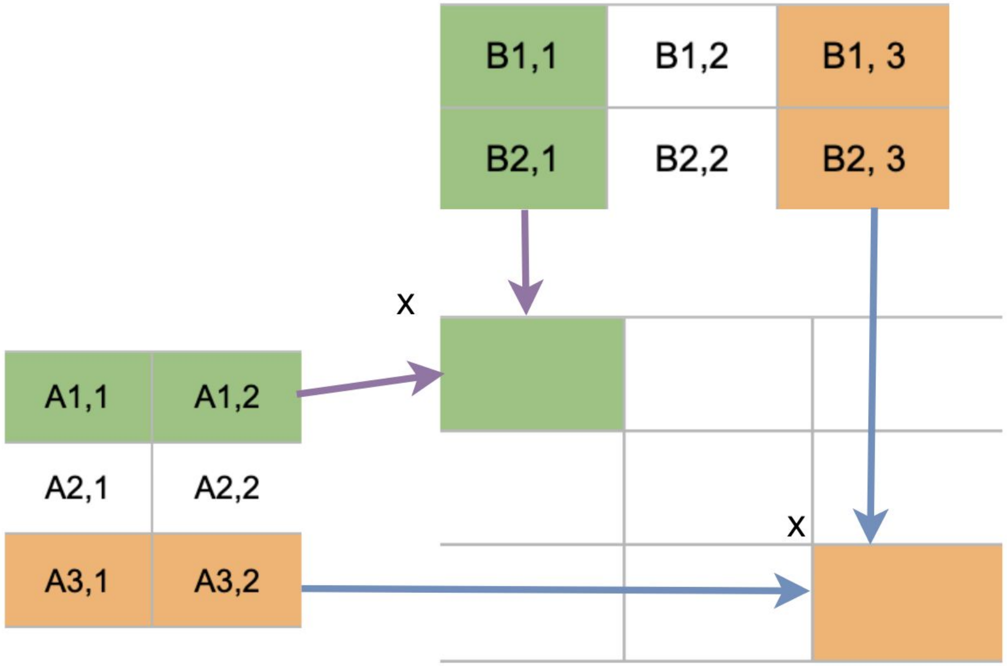

- Applications:
  - Image processing/filtering operations
  - Encryption
  - Machine Learning operations, etc.

# Sequential Approach

- Simple algorithm of Iterating over each matrices 3 times

```
for i from 1 to m:
    for j from 1 to n:
        // Iterating over rows/columns for
        // addition of product in grid[i][j]
        sum := 0
        for p from 1 to k:
            sum <- sum + (A[i][p] * B[p][j])

        C[i][j] = sum

return C
```

- Expensive operation. Takes $O(n^3)$
- Not suitable for large matrices




O (A * B * C)

# Sequential Approach 2

- Strassen Algorithm - Divide and Conquer Approach
- Divide matrix into 4 sub-matrices of n/2 dimensions recursively
- Calculate product using formulas
- Limitations:
  - Matrix Size: nxn
  - n power of 2

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned}
P_1 &= A_{11} \cdot (B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12}) \cdot B_{22} \\
P_3 &= (A_{21} + A_{22}) \cdot B_{11} \\
P_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21}) \cdot (B_{11} + B_{12})
\end{aligned}$$

$$\begin{aligned}
P_5 + P_4 - P_2 + P_6 &= C_{11} \\
P_1 + P_2 &= C_{12} \\
P_3 + P_4 &= C_{21} \\
P_5 + P_1 - P_3 - P_7 &= C_{22}
\end{aligned}$$

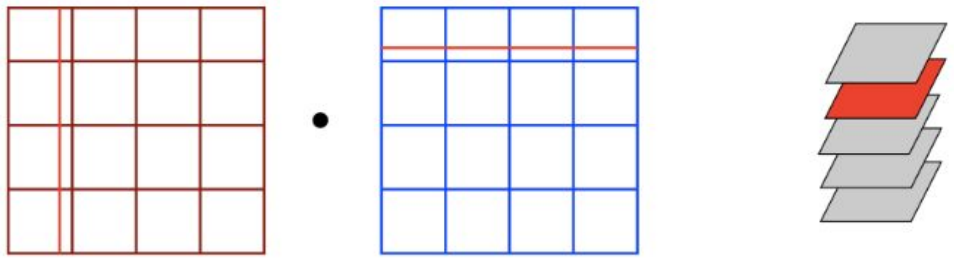
Runtime: $O\ (n^{2.80})$

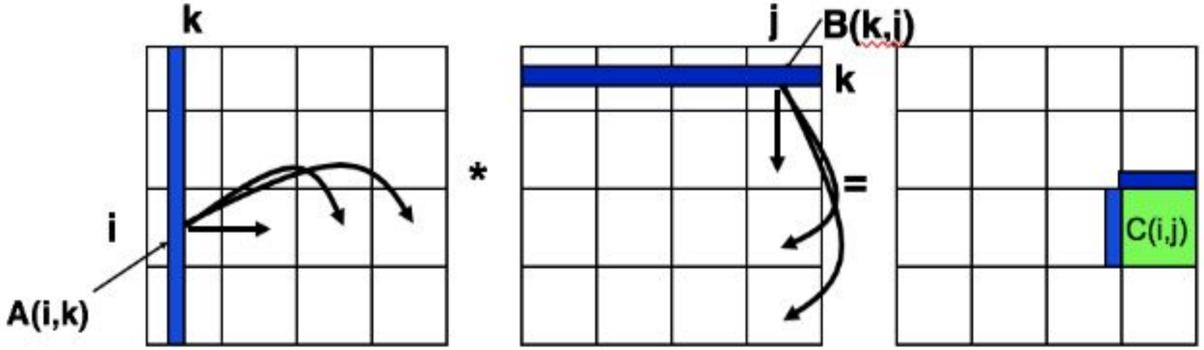
# Parallel Approach

- Based on SUMMA algorithm
- Distributed data across processors with p being √p and matrix size n X n
- Process of row K broadcasts matrix A row to the i-th row
- Process of column K broadcasts matrix B column to the j-th colum
- Perform matrix multiplication over small set of data locally on each processor

for $k := 0$ to $n-1$

$$C[:,:] \mathrel{+}= A[:,k] \cdot B[k,:]$$

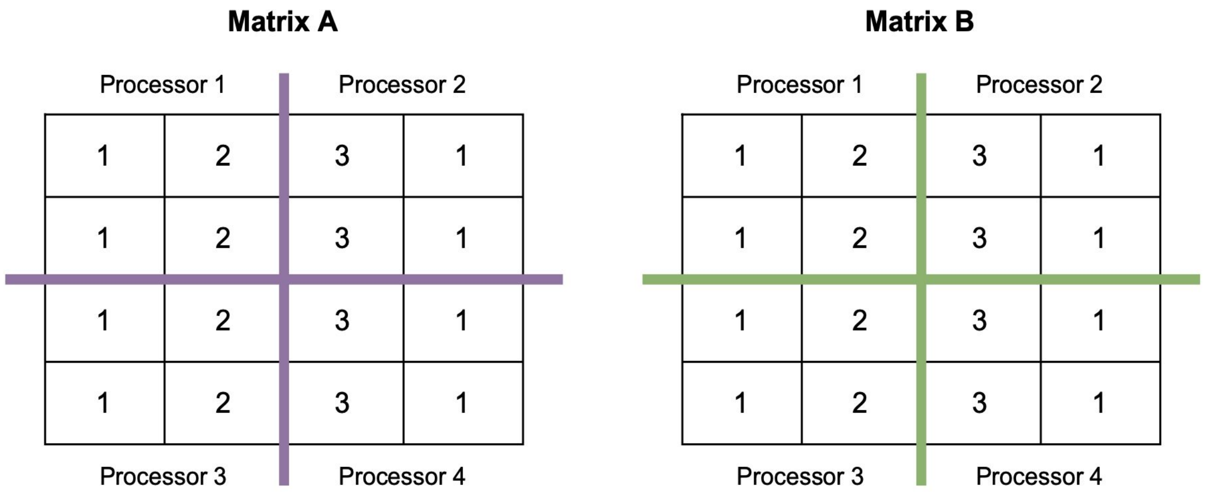





Source: http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Matrix%20Multiplication%20by%20Nur.pdf

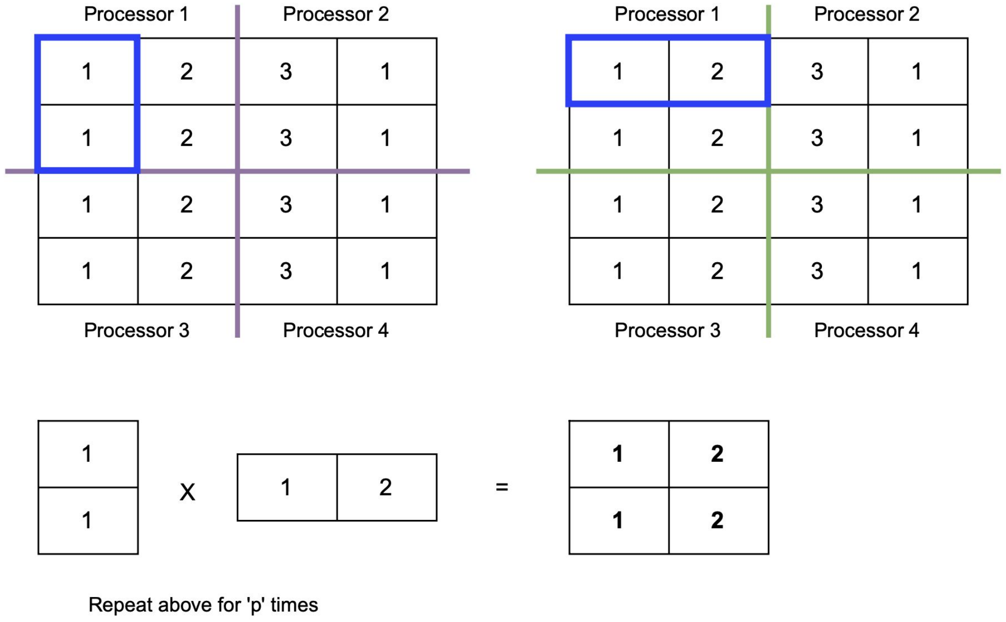
# Example

```
# Initial Data Distribution
P(i,j) contains A(i,j) and B(i,j)

for k <- 0 to √p:
    for i <- 0 to √p:
        P(i, k) broadcasts A(i,k) to i-th row
    for j <- 0 to √p:
        P(k, j) broadcasts B(k,j) to j-th column

    P(i,j) computes C(i,j) <- C(i,j) + [A(i,k) * B(k,j)]
end
```

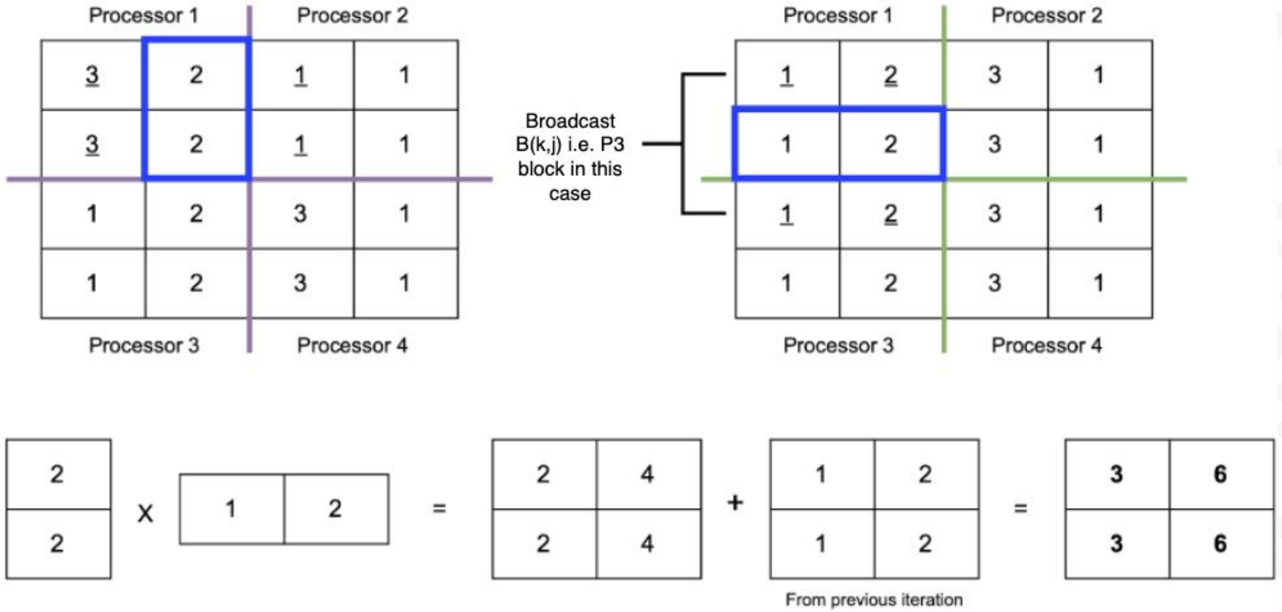


# Example

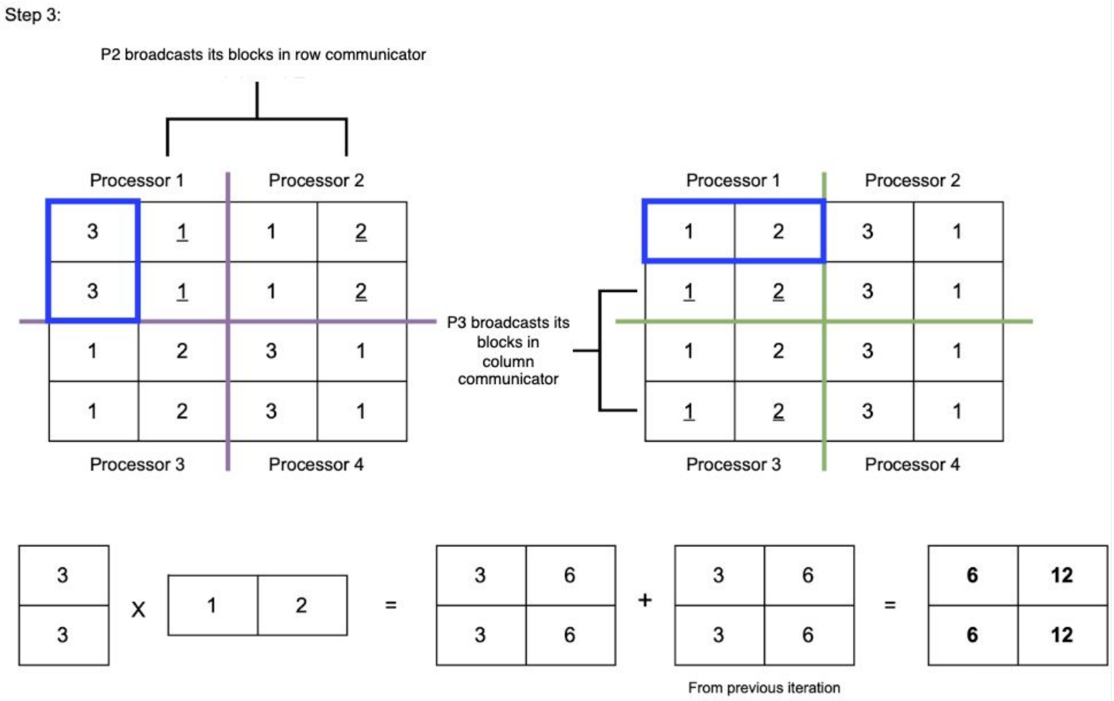
Step 1:

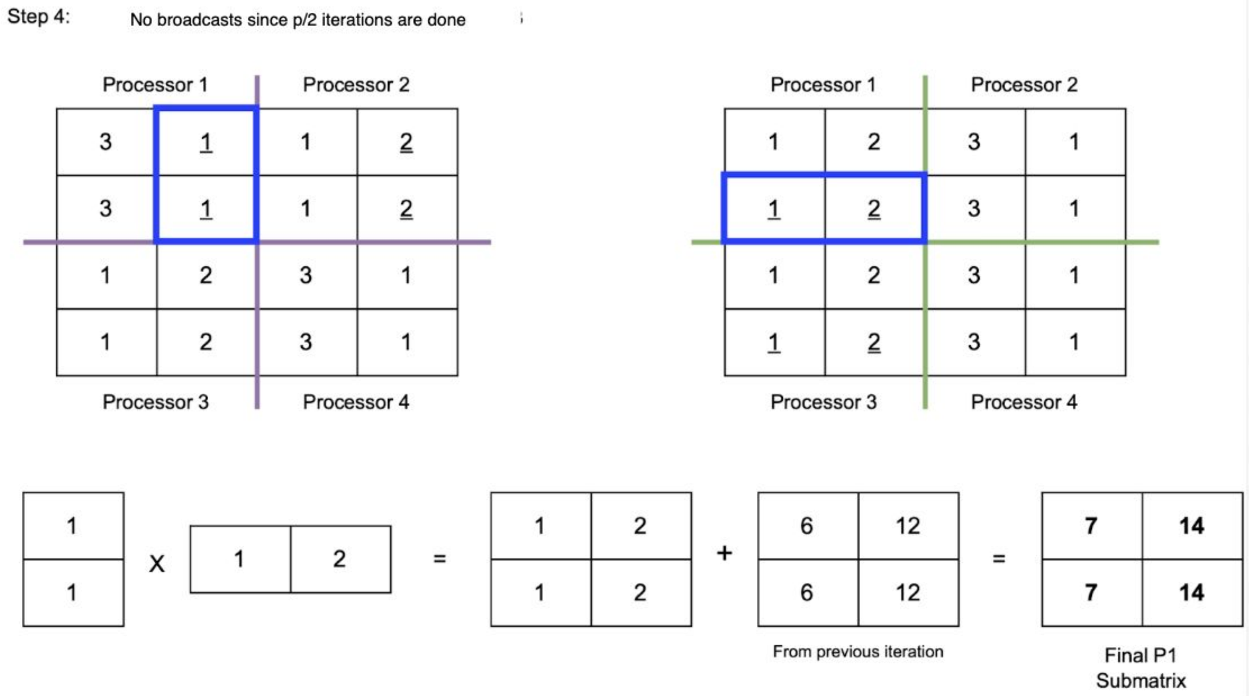



Step 2 Broadcast A(i,k) i.e. P2 block in this case

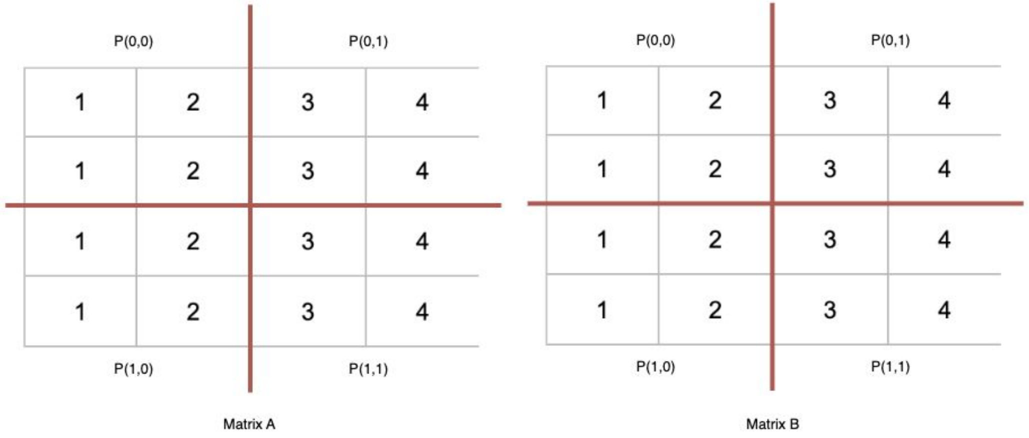


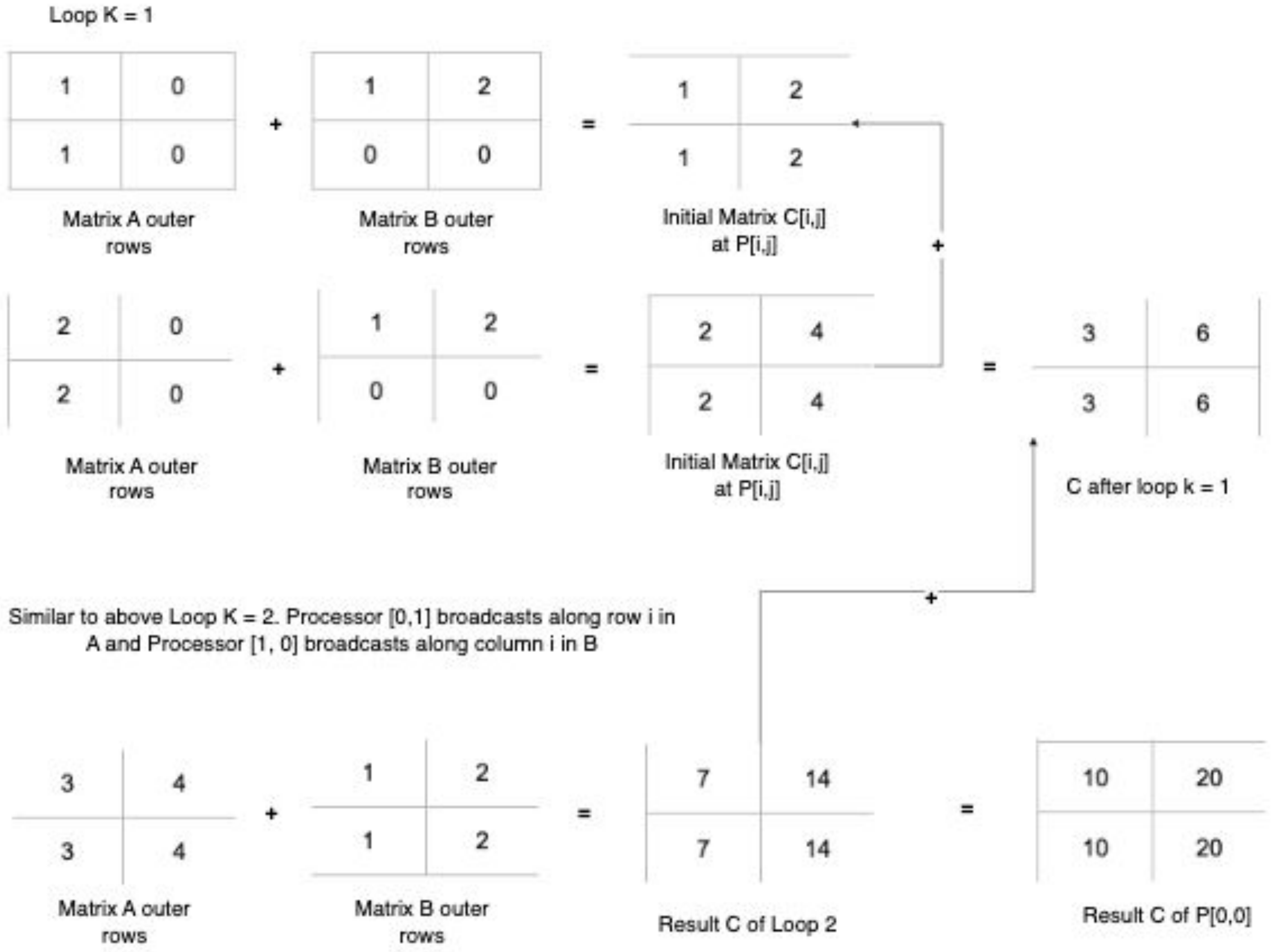
# Example


Step 3:
P2 broadcasts its blocks in row communicator
Processor 1
Processor 2
Processor 3
Processor 4
P3 broadcasts its blocks in column communicator
From previous iteration


Step 4:
No broadcasts since p/2 iterations are done
Processor 1
Processor 2
Processor 3
Processor 4
From previous iteration
Final P1 Submatrix

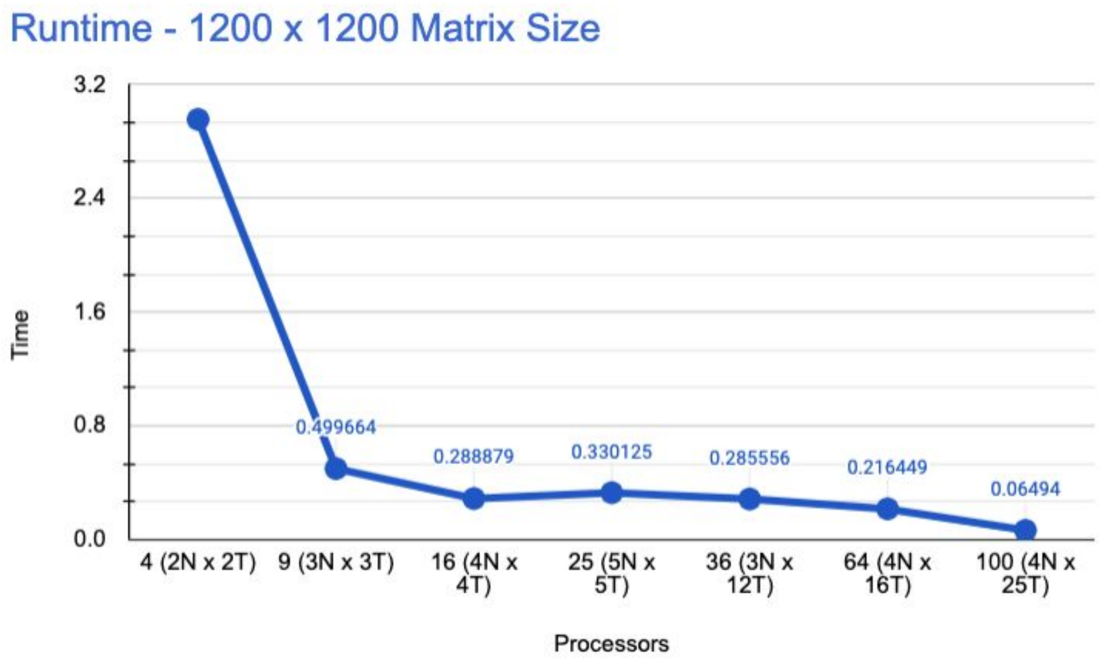
# Example

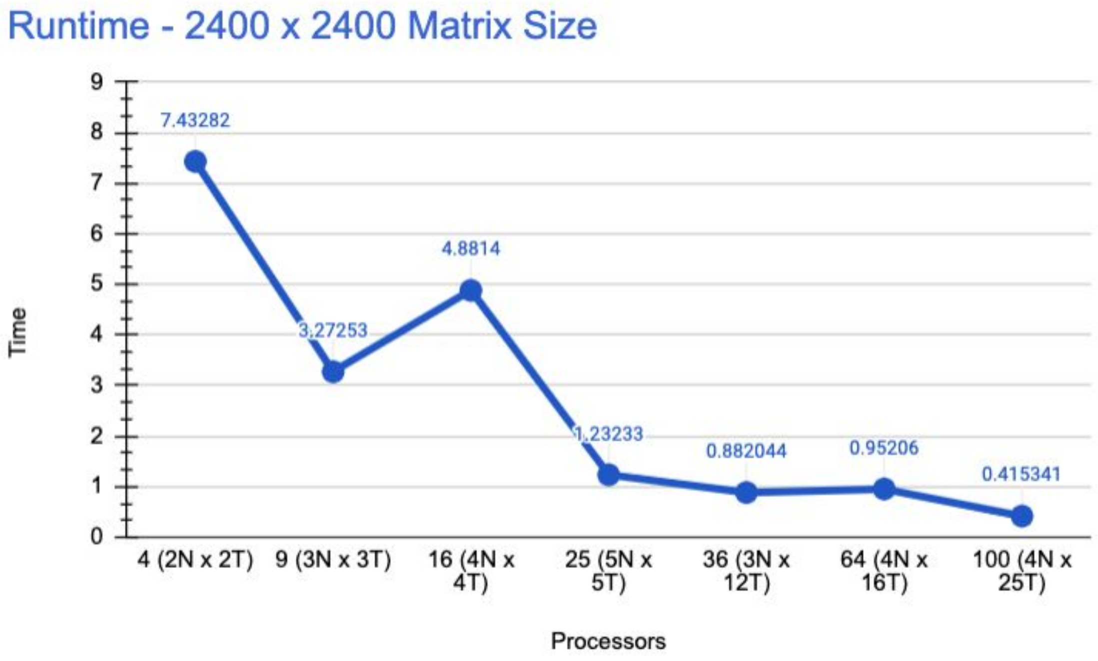

P(0,0)
P(0,1)
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
P(1,0)
P(1,1)
Matrix A
P(0,0)
P(0,1)
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
P(1,0)
P(1,1)
Matrix B


Loop K = 1
Matrix A outer rows
Matrix B outer rows
Initial Matrix C[i,j] at P[i,j]
Matrix A outer rows
Matrix B outer rows
Initial Matrix C[i,j] at P[i,j]
C after loop k = 1
Similar to above Loop K = 2. Processor [0,1] broadcasts along row i in A and Processor [1, 0] broadcasts along column i in B
Matrix A outer rows
Matrix B outer rows
Result C of Loop 2
Result C of P[0,0]

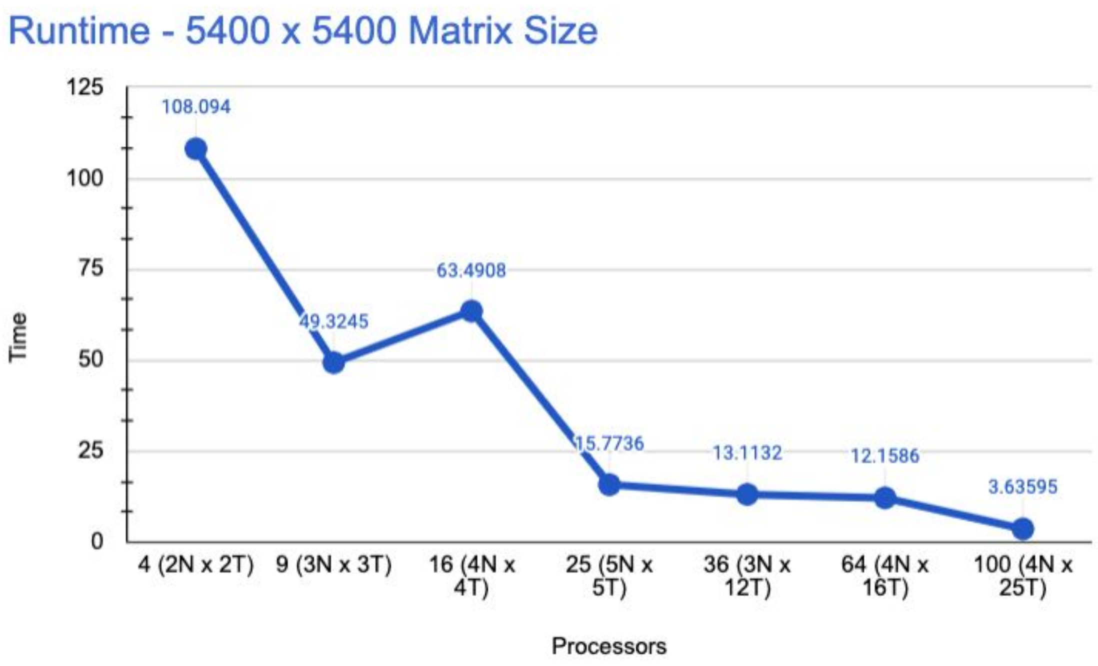
# Results (Runtime)

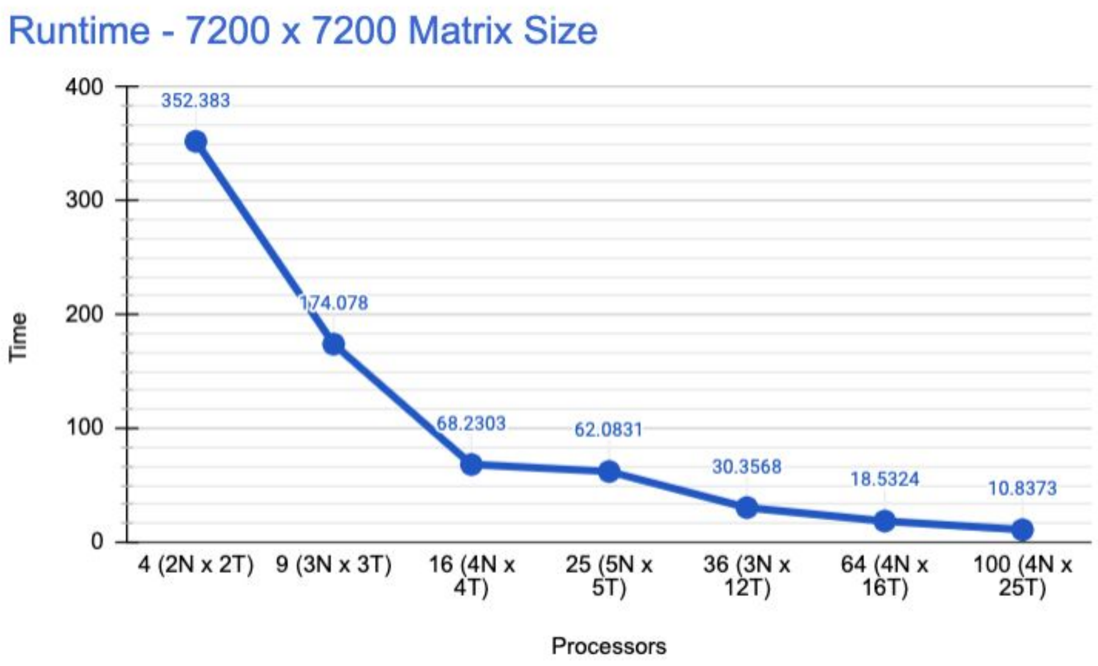

# Results (Runtime)



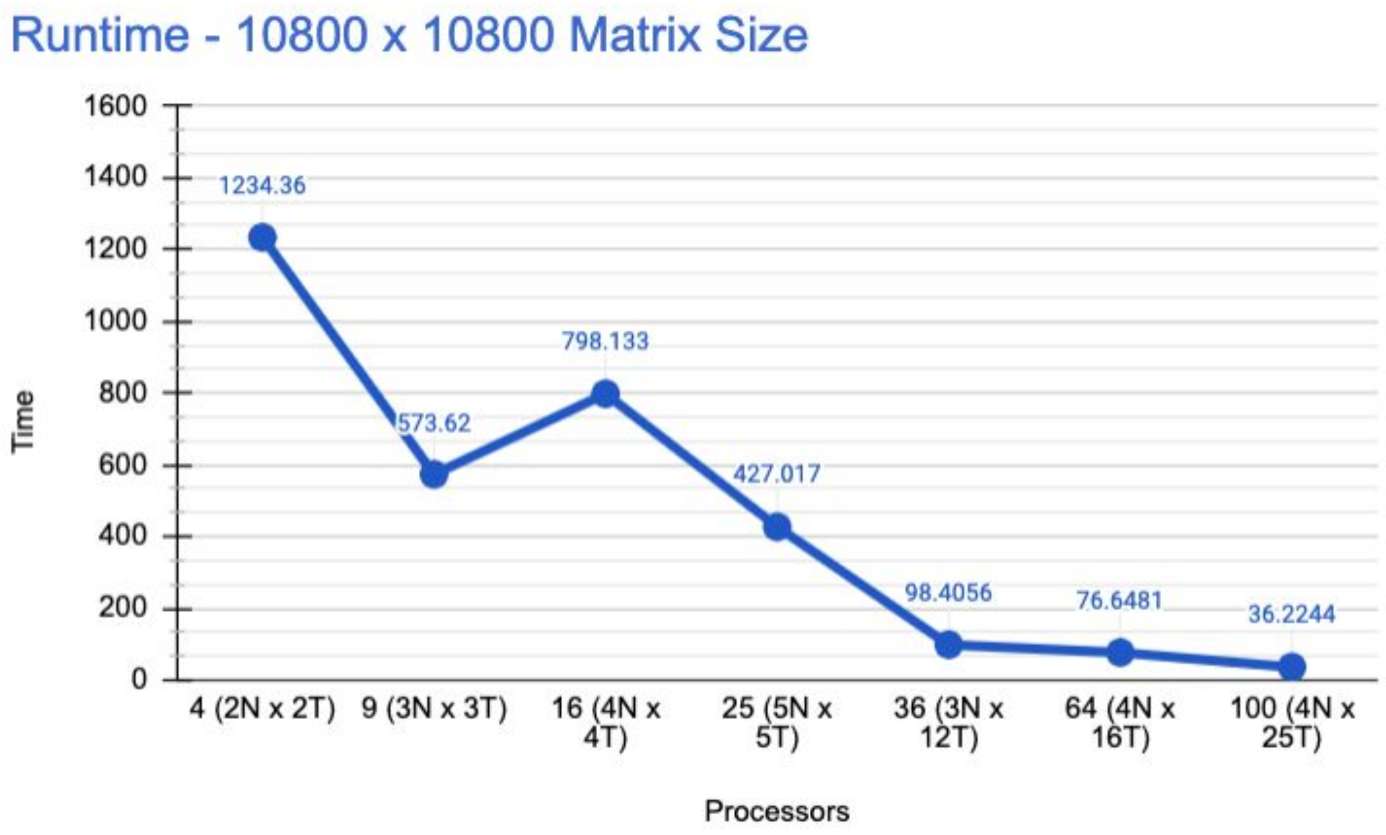



# Results (Runtime)

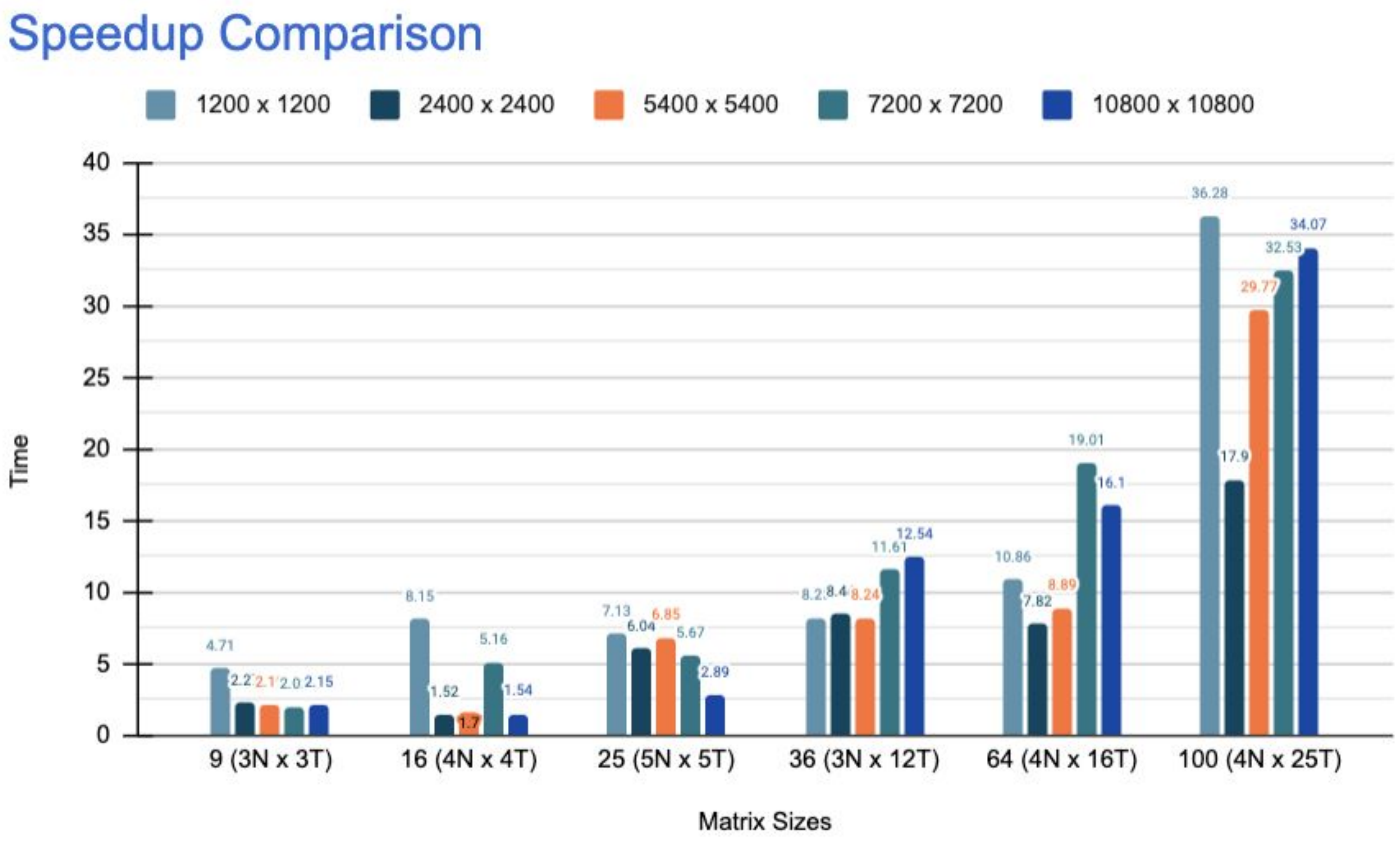


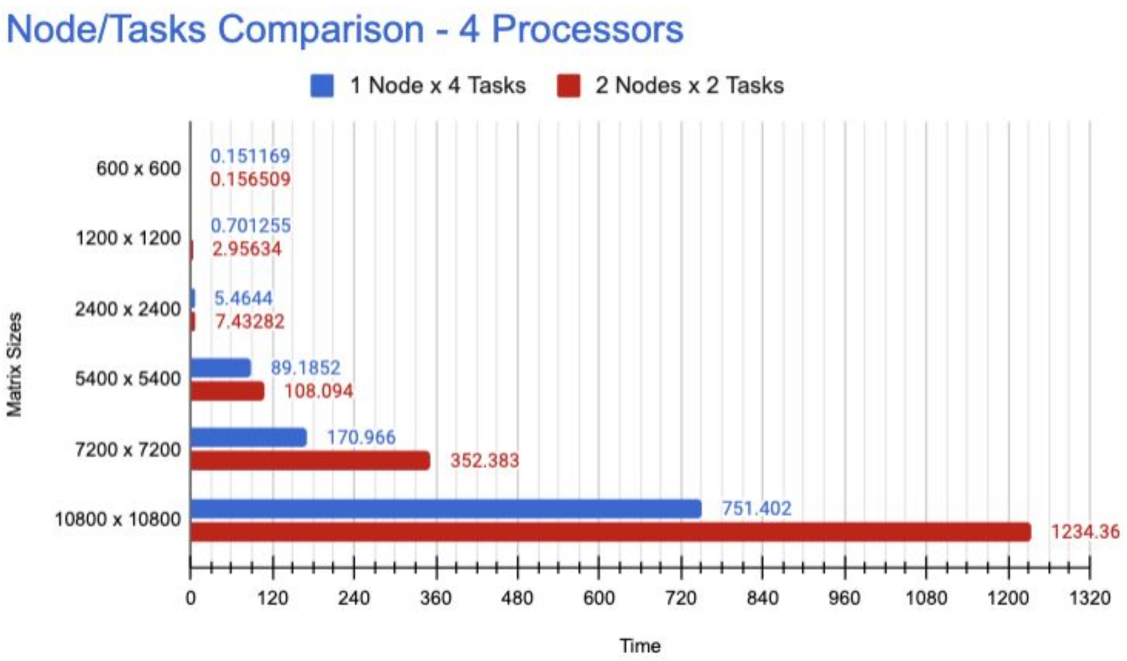
# Results (Runtime)

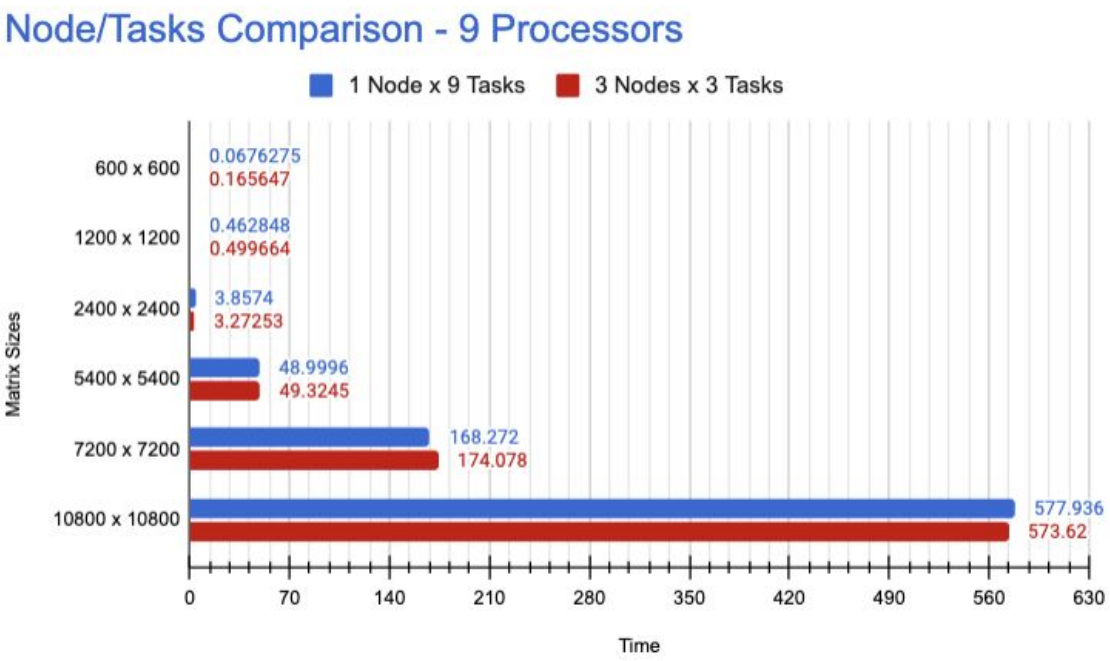


# Results (Speedup)

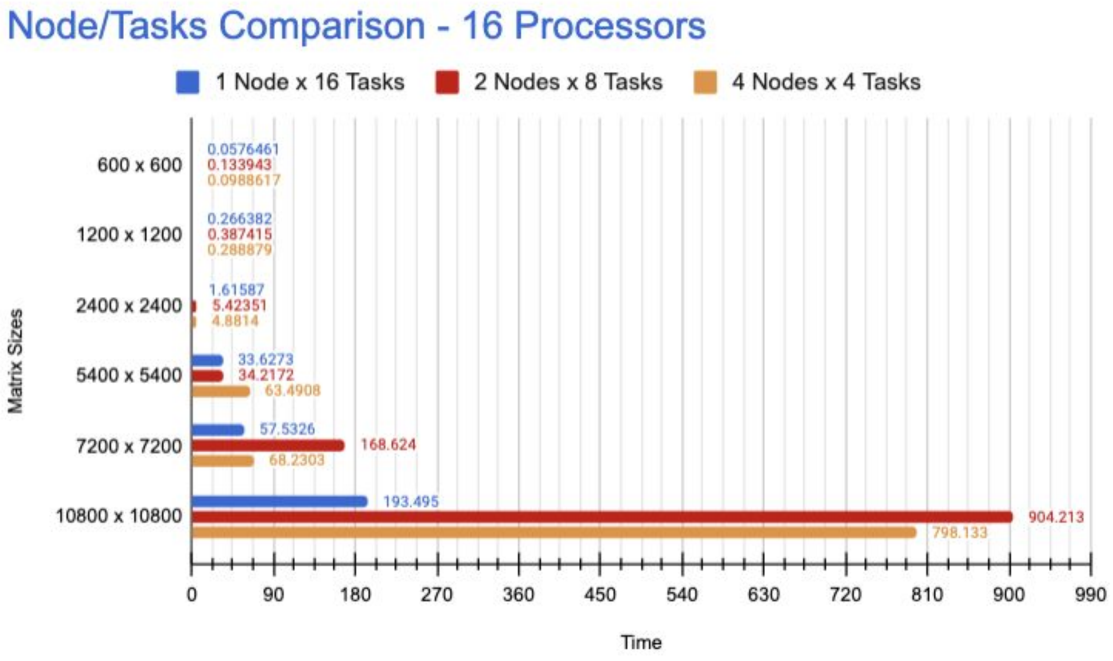


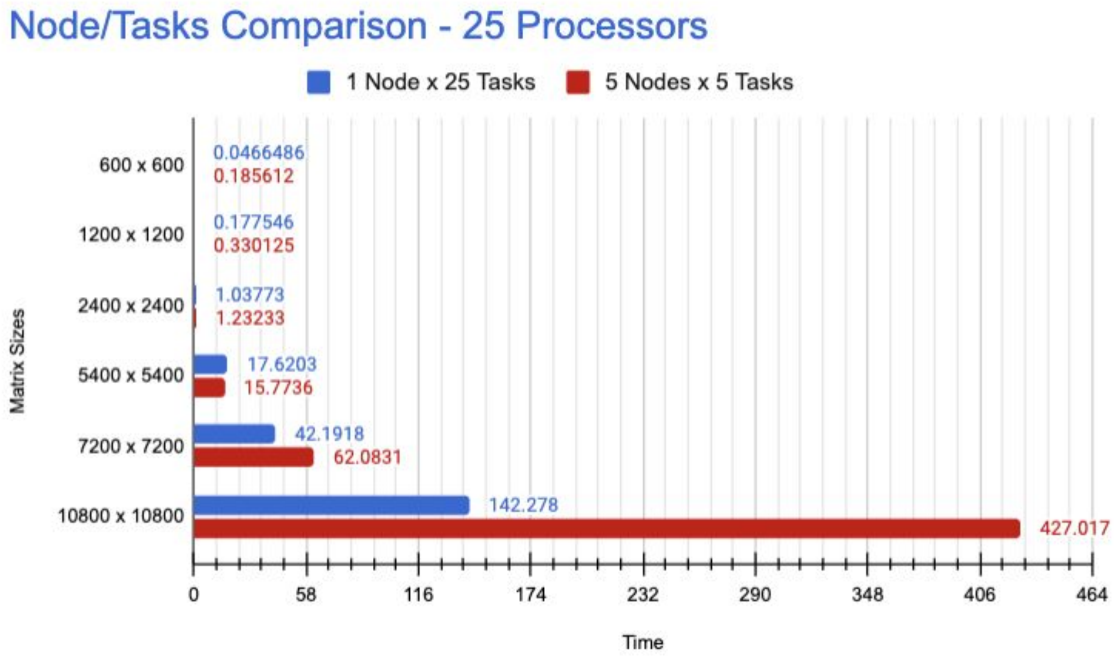
# Results (Distributed Comparison)



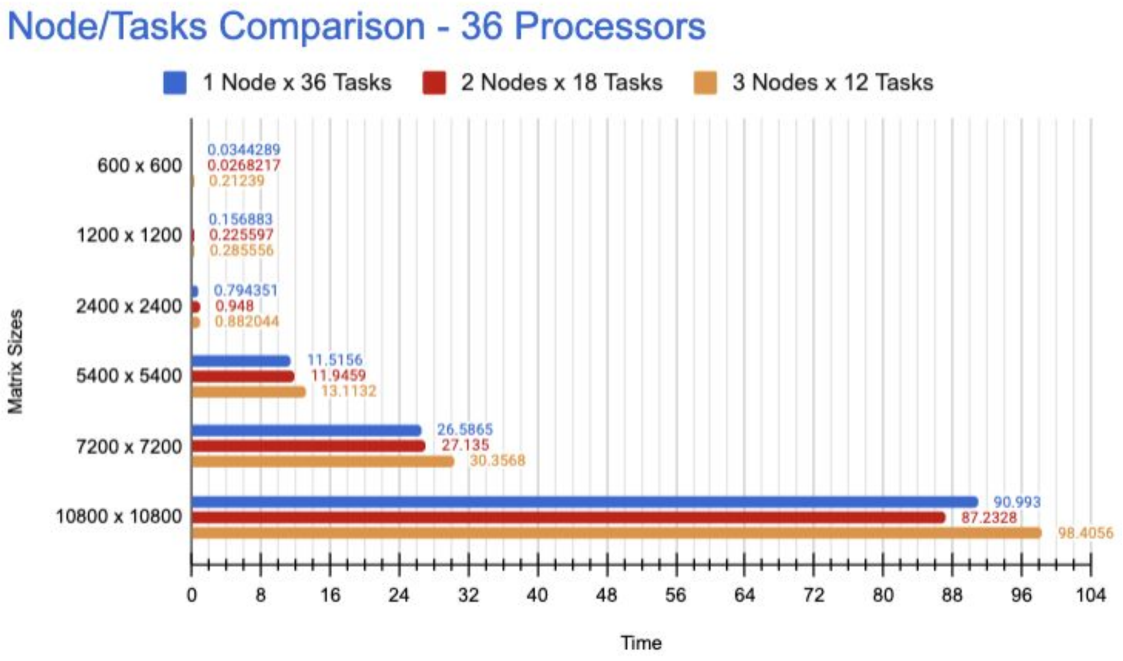



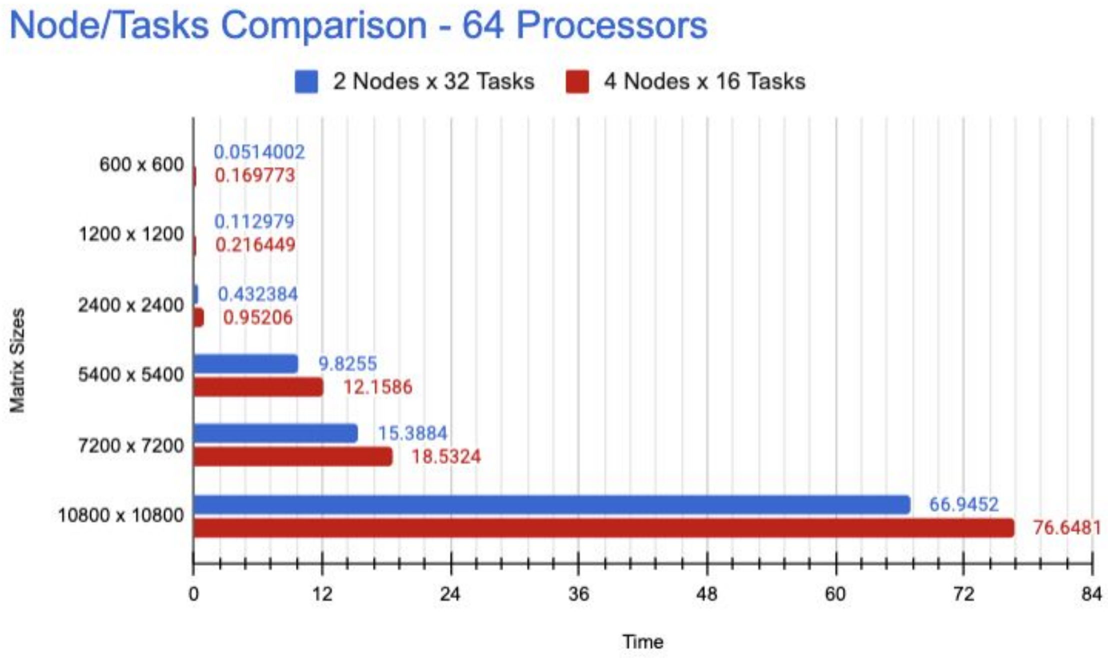
# Results (Distributed Comparison)



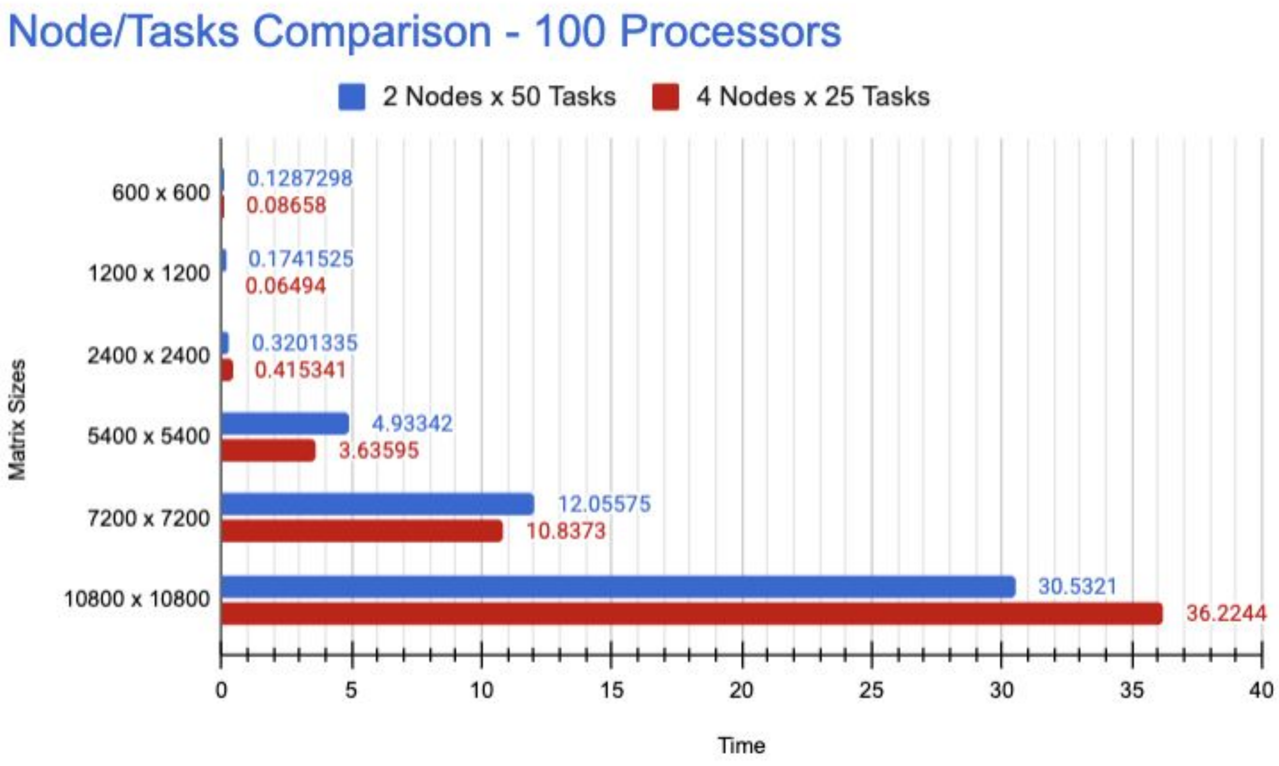



# Results (Distributed Comparison)



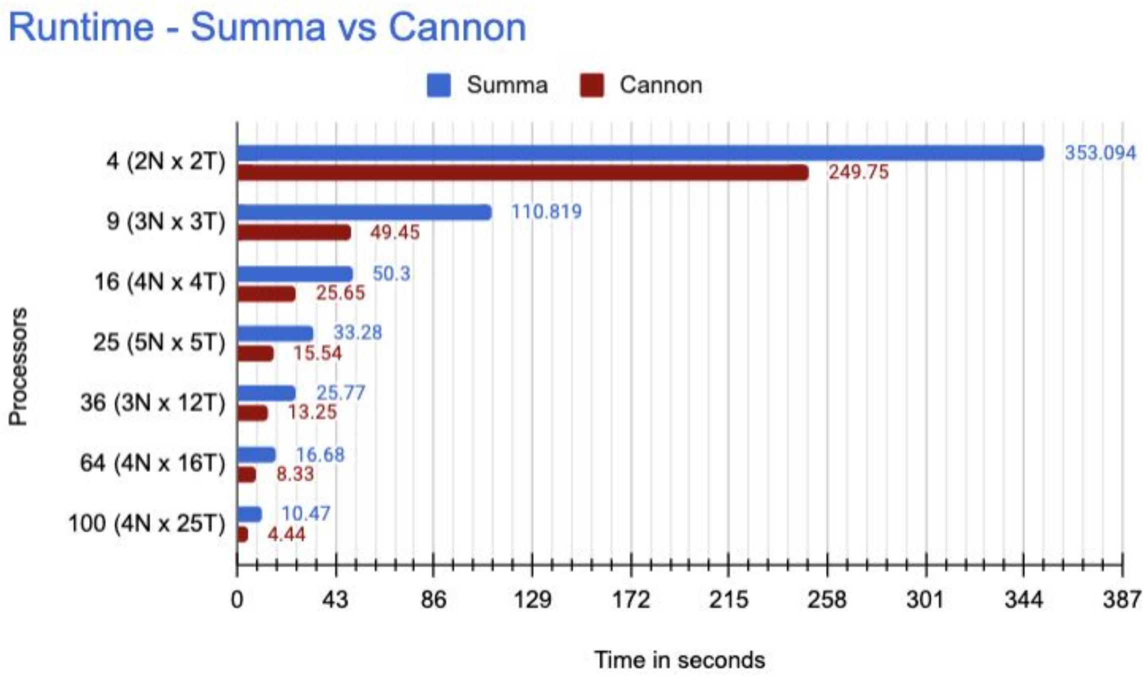



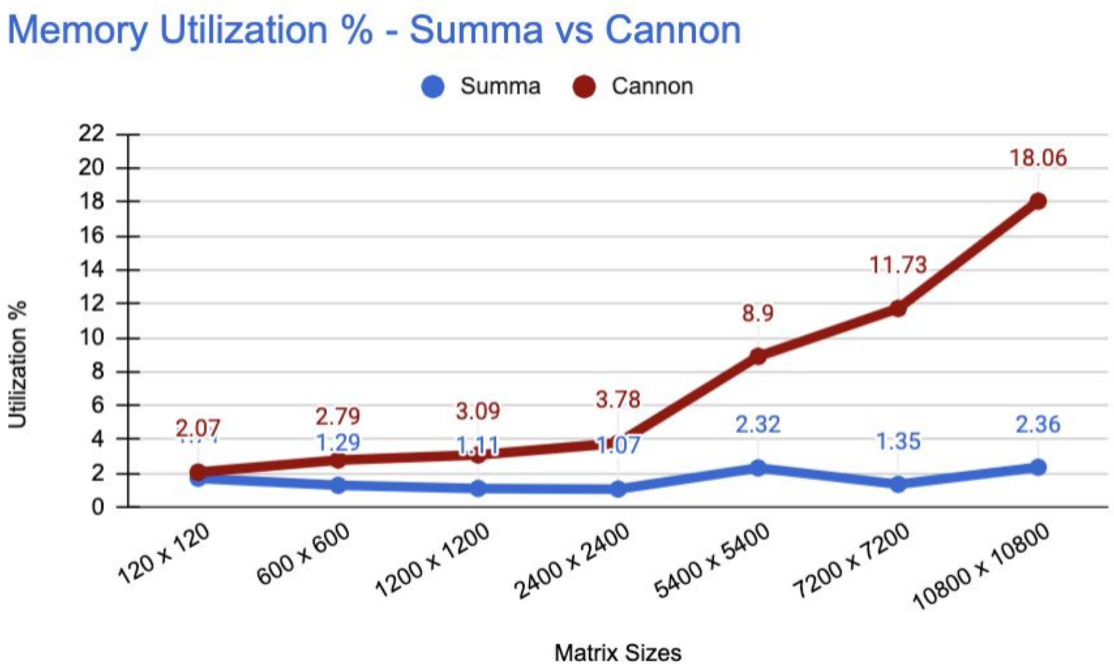
# Results (Distributed Comparison)



# Results (Memory)

- Comparing memory utilization of Summa vs Cannon’s Algorithm
- Summa uses broadcast of blocks vs Cannon’s circular shift



# Takeaways

- The higher number of distributed nodes, the more the effect is on program runtime and speedup.
- Understanding of MPI Communicators and Carts for processor grids.
- Learned a lot about processor communication.

# References


- http://www.netlib.org/lapack/lawnspdf/lawn96.pdf
- https://dl.acm.org/doi/10.5555/899248
- https://cs.iupui.edu/~fgsong/LearnHPC/summa/index.html
- http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Matrix%20Multiplication%20by%20Nur.pdf
- https://cseweb.ucsd.edu/classes/sp11/cse262-a/Lectures/262-pres1-hal.pdf

# Thank You