PARALLEL MATRIX MULTIPLICATION

Presented by: Aditya Sabnis

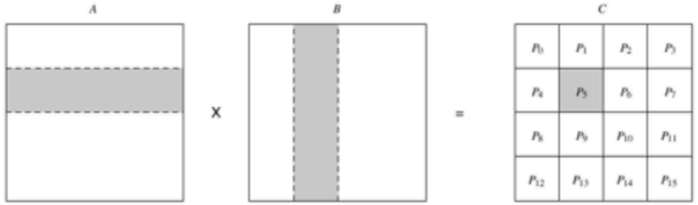
University at Buffalo The State University of New York

Outline

- Problem Statement
- Applications
- Sequential Matrix Multiplication
- Parallel Matrix Multiplication
- Cannon’s Algorithm
- Numerical example
- Results
- Next Steps

PROBLEM STATEMENT

If matrix A = [a_{ij}] is an m×n matrix and B = [b_{ij}] is an n×p matrix then the matrix multiplication A×B is an m×p matrix.

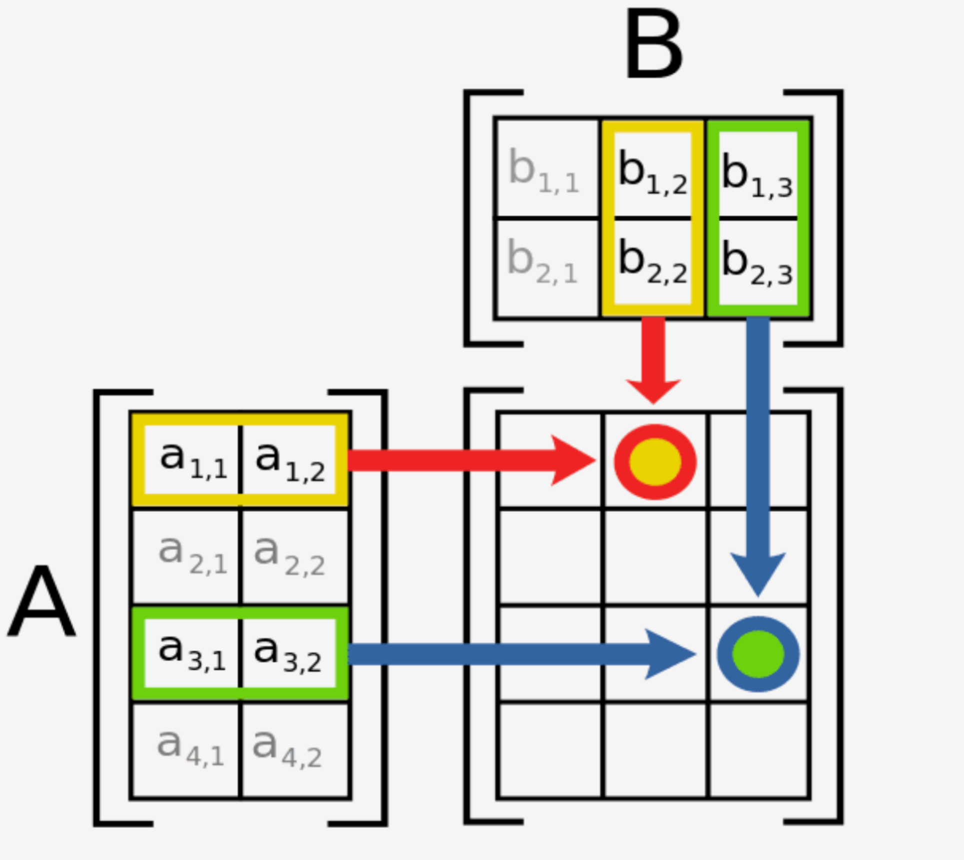


AB = [c_{ij}], where $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$

(The entry in the ith row and jth column is denoted by the double subscript notation a_{ij}, b_{ij}, and c_{ij}.)

PROBLEM STATEMENT (contd.)

Number of Columns in A match the number of Rows of B.



Depiction of matrix multiplication, taken from Wikipedia

Applications

- Used in image filtering using 2D convolution
- Used in Machine learning algorithms
- Used for quantum mechanics

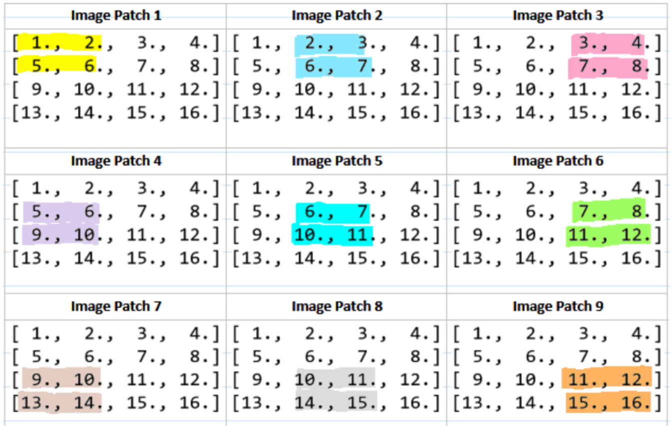


Image Patch 1

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 2

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 3

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 4

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 5

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 6

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 7

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

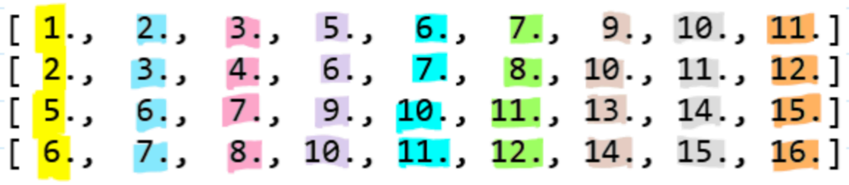
Image Patch 8

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

Image Patch 9

```
[ 1.,  2.,  3.,  4.]
[ 5.,  6.,  7.,  8.]
[ 9., 10., 11., 12.]
[13., 14., 15., 16.]
```

All the possible 2 × 2 image patches in **X** given the parameters of the 2D convolution. Each color represents a unique patch



```
[ 1.,  2.,  3.,  5.,  6.,  7.,  9., 10., 11.]
[ 2.,  3.,  4.,  6.,  7.,  8., 10., 11., 12.]
[ 5.,  6.,  7.,  9., 10., 11., 13., 14., 15.]
[ 6.,  7.,  8., 10., 11., 12., 14., 15., 16.]
```

The matrix of image patches, **P**

```
[1., 2., 3., 4.]
```

W as a flattened row vector **K**

```
[ 44.,  54.,  64.,  84.,  94., 104., 124., 134., 144.]
```

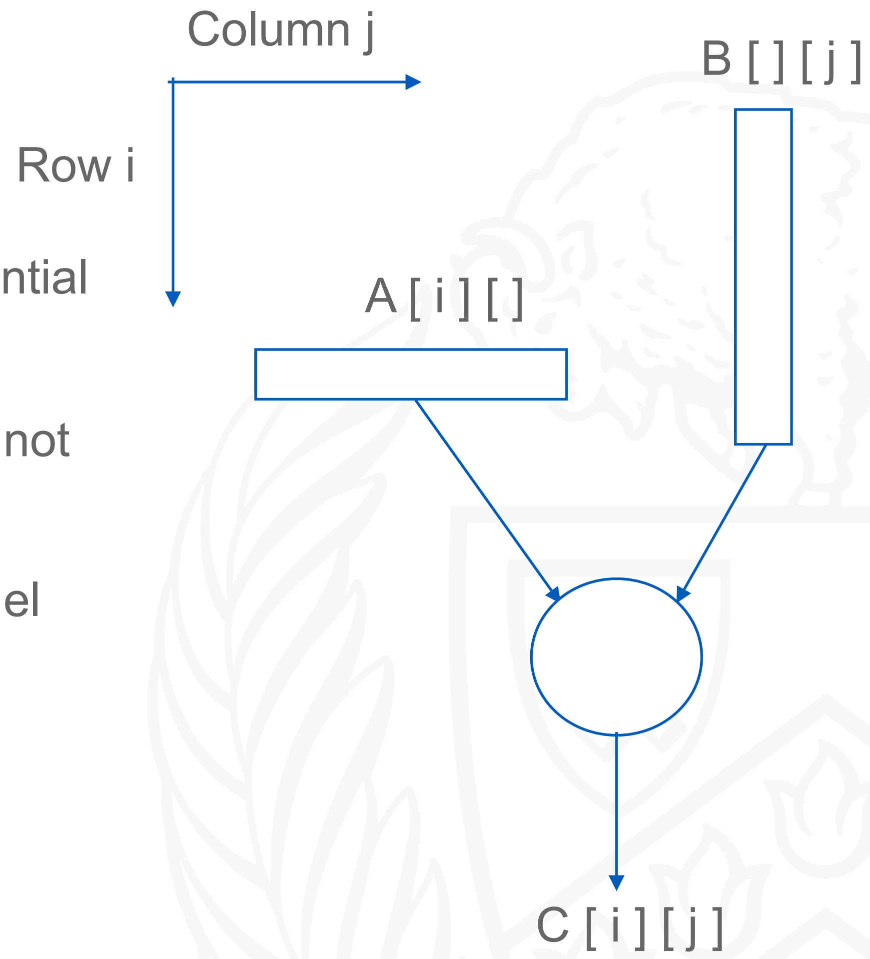
The result of **K P**

Sequential Matrix Multiplication

```
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] = c[i][j] + a[i][k]*b[k][j]
              }
        }
}
```

NOTE: The total number of steps are n*n*(2n-1). So the complexity of the problem is $O(n^3)$

Parallel Matrix Multiplication



- Parallel matrix multiplication is usually based on the sequential matrix multiplication algorithm.
- The computation in each iteration of the two outer loops is not dependent upon any other iteration.
- Each instance of the inner loop could be executed in parallel
- Complexity of $O(n^2)$ is obtainable with n processors
- Complexity of $O(n)$ is obtainable with n^2 processors
- Complexity of $O(\log n)$ is obtainable with n^3 processors

Parallel - Cannon’s Algorithm

Step 1: Divide the matrix A and B into P square blocks, where P is the number of Processors.

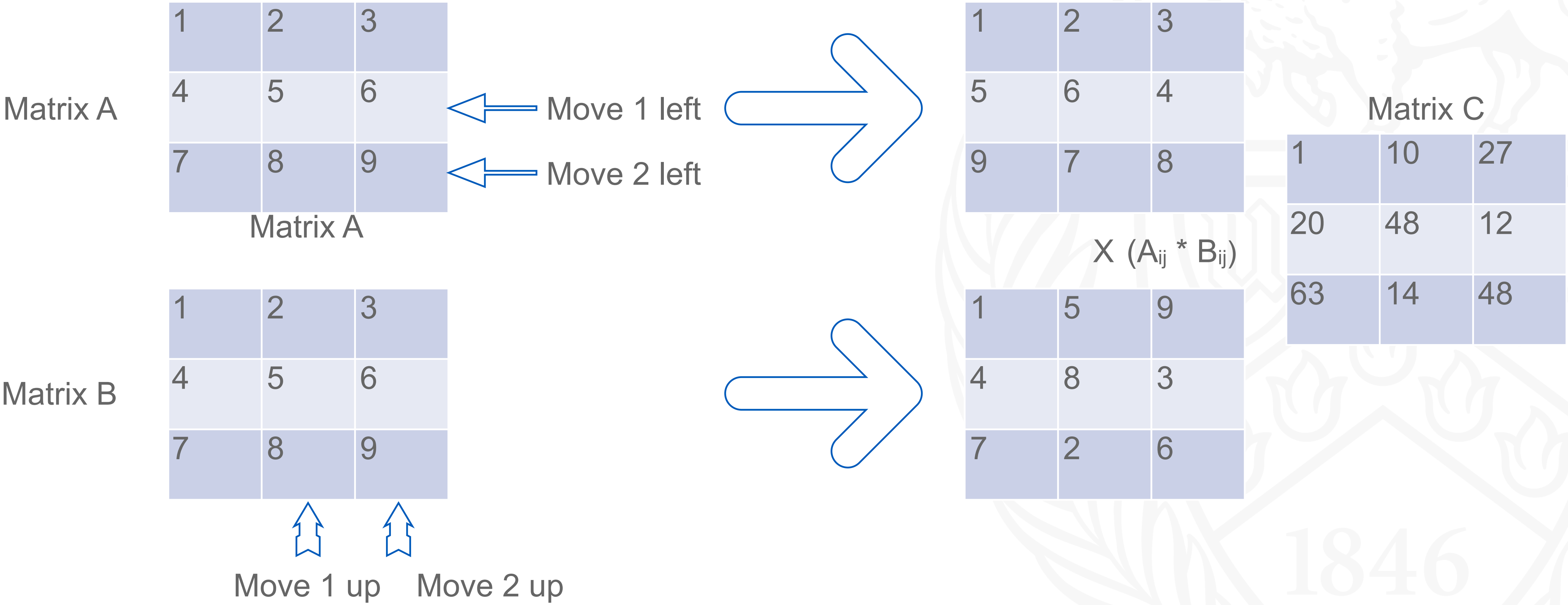
Step 2: Create grid of processors of size $P^{1/2} * P^{1/2}$ So that each process had block of A and block of B

Step 3: Each process has a sub block C to which we add the results after Multiplying the sub-blocks in the processor.

Step 4: The sub-blocks of A are shifted one step to the left and the sub-blocks of B are shifted one step up.

Step 5: We repeat the steps 3 and 4 for square root of P times.

Numerical Example



Matrix A

1	2	3
4	5	6
7	8	9

Move 1 left

Move 2 left

Matrix A

Matrix B

1	2	3
4	5	6
7	8	9

Move 1 up

Move 2 up

1	2	3
5	6	4
9	7	8

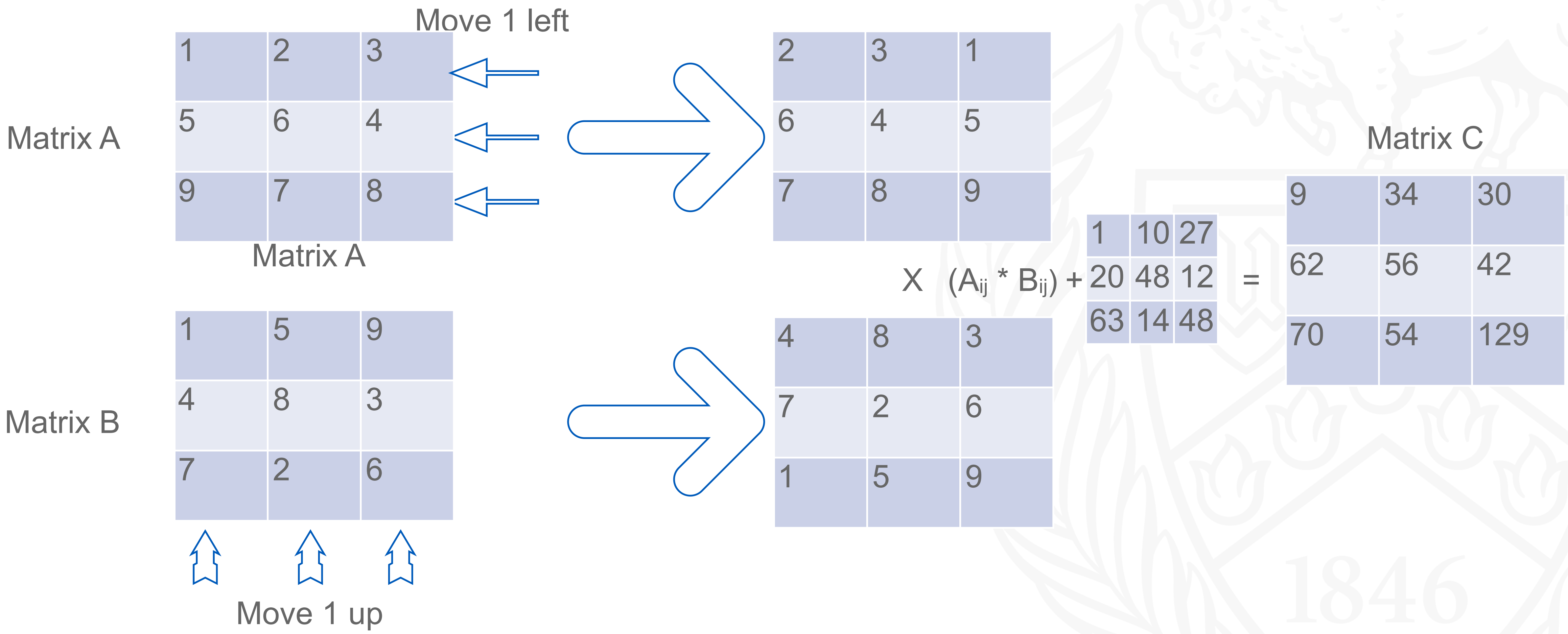
X $(A_{ij} * B_{ij})$

1	5	9
4	8	3
7	2	6

Matrix C

1	10	27
20	48	12
63	14	48

Numerical Example (Contd.)



Matrix A

Move 1 left

1	2	3
5	6	4
9	7	8

Matrix A

2	3	1
6	4	5
7	8	9

Matrix B

1	5	9
4	8	3
7	2	6

Move 1 up

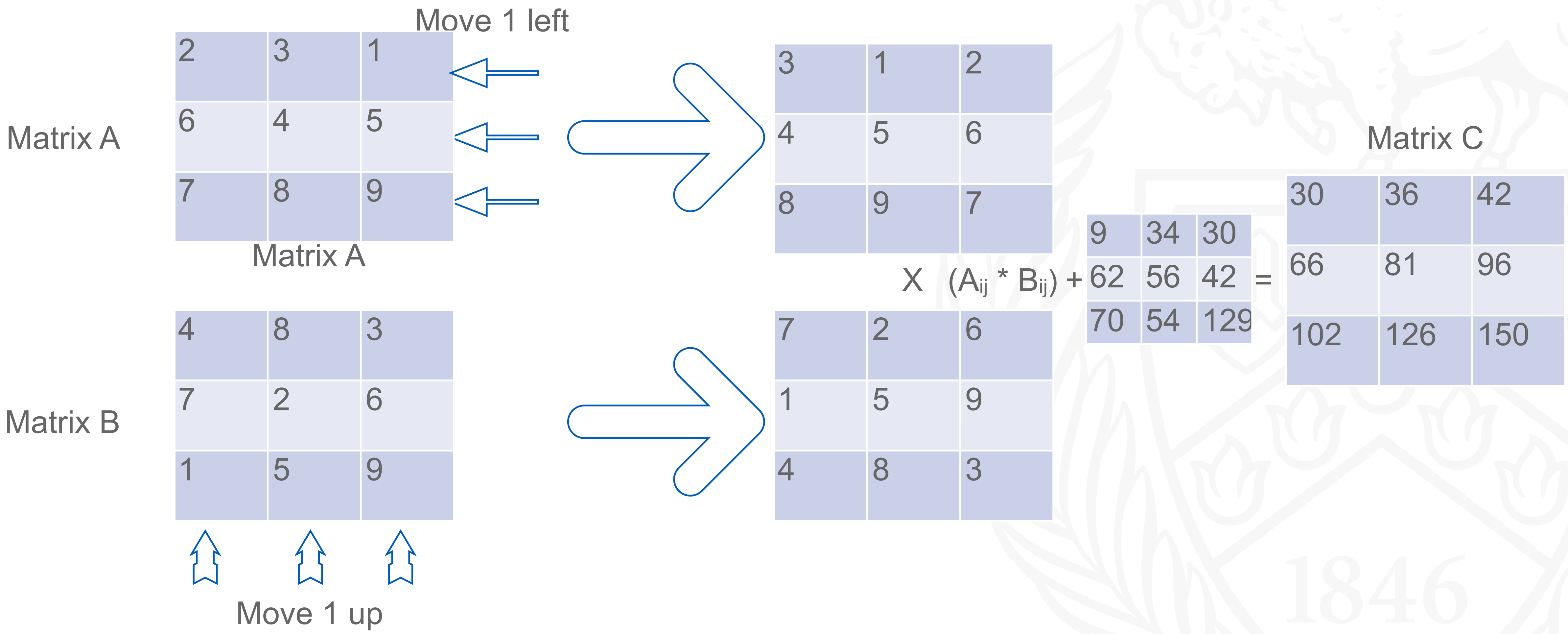
4	8	3
7	2	6
1	5	9

$$X \quad (A_{ij} * B_{ij}) + \begin{bmatrix} 1 & 10 & 27 \\ 20 & 48 & 12 \\ 63 & 14 & 48 \end{bmatrix} =$$

Matrix C

9	34	30
62	56	42
70	54	129

Numerical Example (Contd.)



Move 1 left

Matrix A

2	3	1
6	4	5
7	8	9

Matrix A

3	1	2
4	5	6
8	9	7

Matrix B

4	8	3
7	2	6
1	5	9

Move 1 up

7	2	6
1	5	9
4	8	3

X $(A_{ij} * B_{ij}) +$

9	34	30
62	56	42
70	54	129

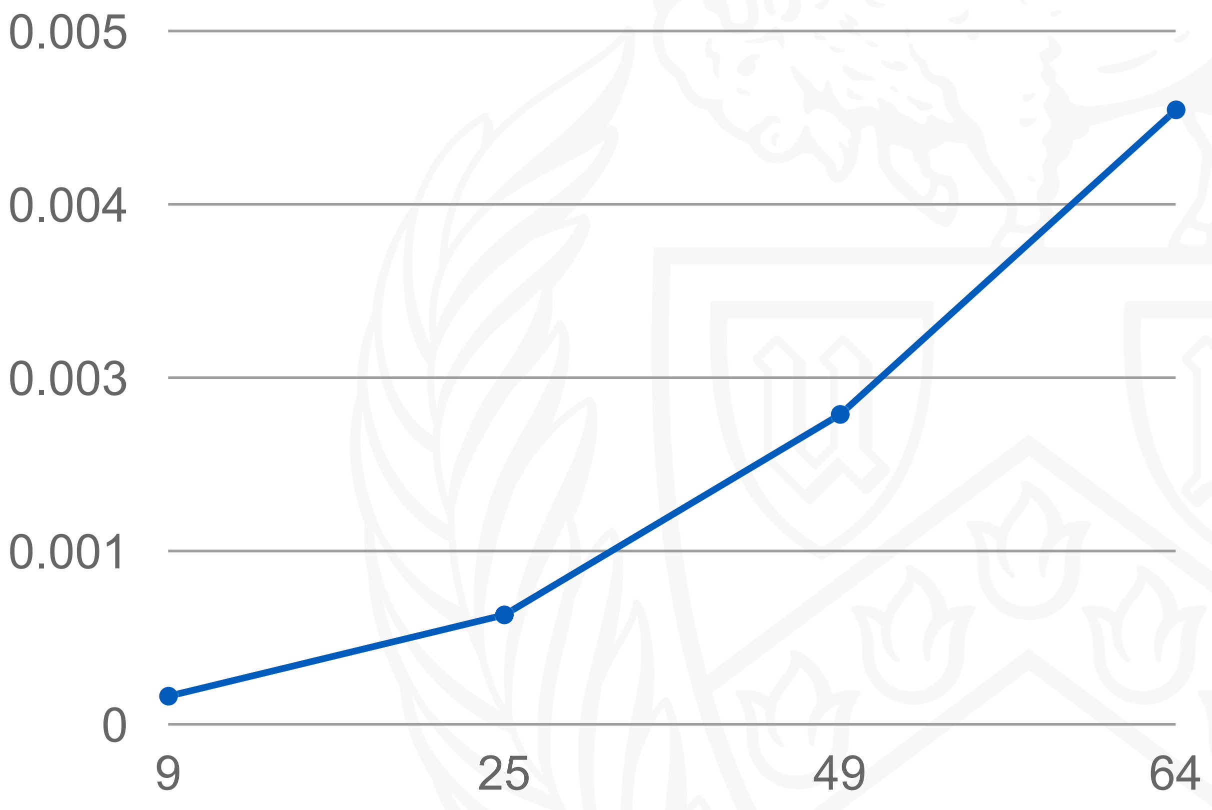
=

Matrix C

30	36	42
66	81	96
102	126	150

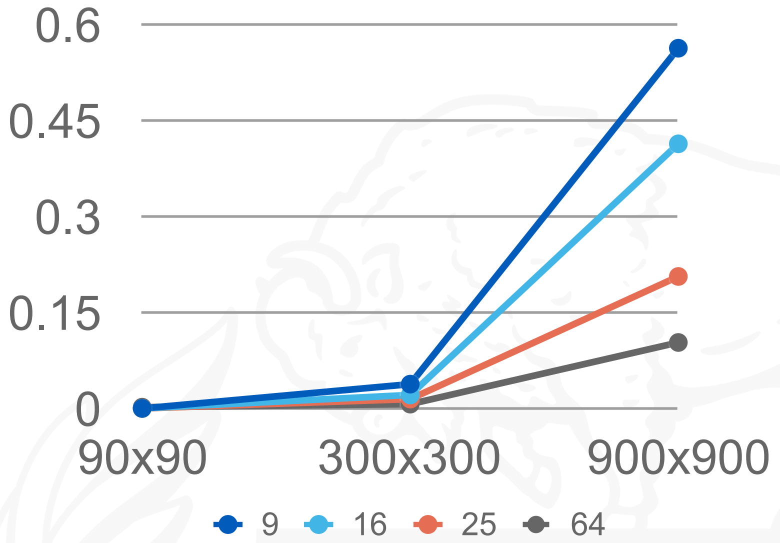
Results

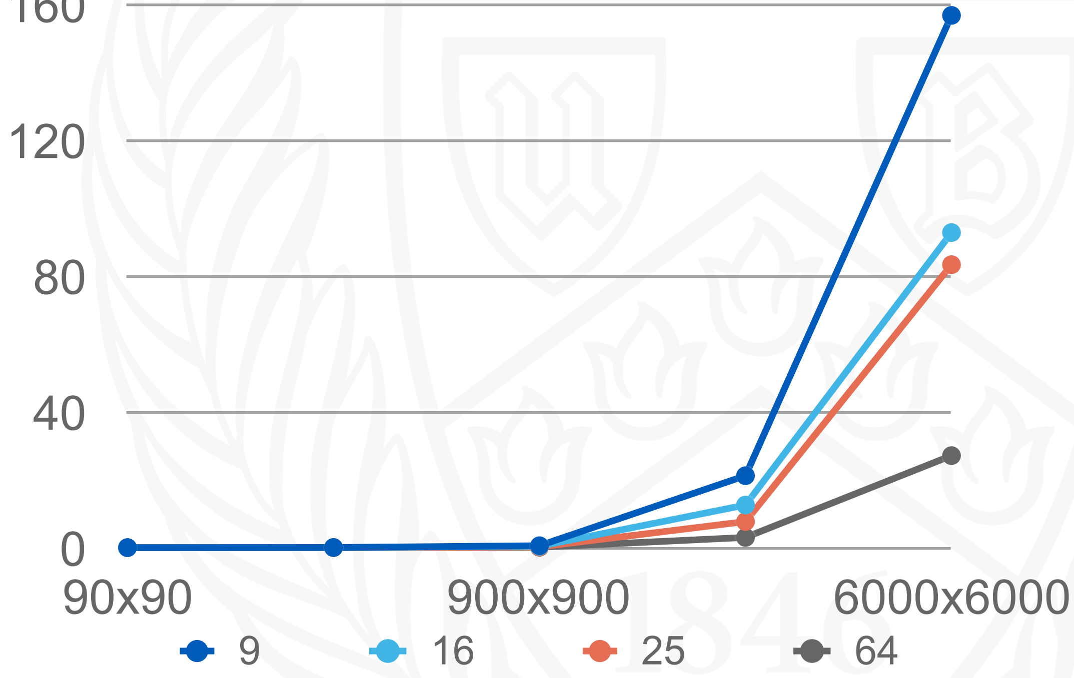
Matrix Size	No of processors	Time (secs)
3*3	9	0.000204
5*5	25	0.000791
7*7	49	0.002239
8*8	64	0.004439



Parallel Processing

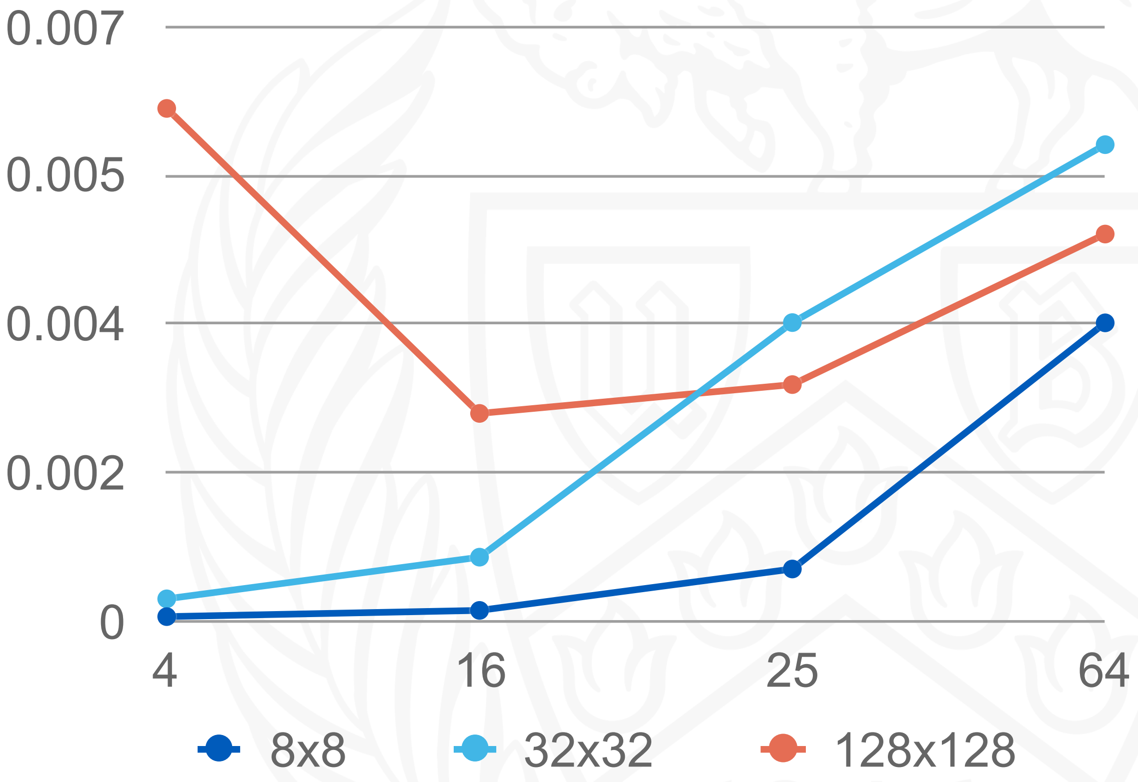
	9	16	25	64
90x90	0.001354	0.001467	0.001426	0.003325
300x300	0.039349	0.022482	0.016006	0.008147
900x900	0.564426	0.415001	0.207663	0.104493
3000x3000	21.176418	12.546824	7.677214	3.003271
6000x6000	156.690799	92.761089	83.312945	27.089127





Increasing the number of processors

	8x8	32x32	128x128	1000x1000
4	0.000061	0.000270	0.006036	1.924891
16	0.000134	0.000760	0.002450	0.830853
25	0.000621	0.003518	0.002789	0.190719
64	0.003515	0.005610	0.003559	0.057688



Learnings

- Understanding MPI and Parallel processing.
- Cannon’s Matrix Multiplication Algorithm
- Parallel processing and its effect on runtime.
- Understanding that just increasing the nodes won’t always reduce the runtime.

Next Steps

- Run single block matrix multiplication in parallel.
- Implementation using OpenMP.

THANK YOU