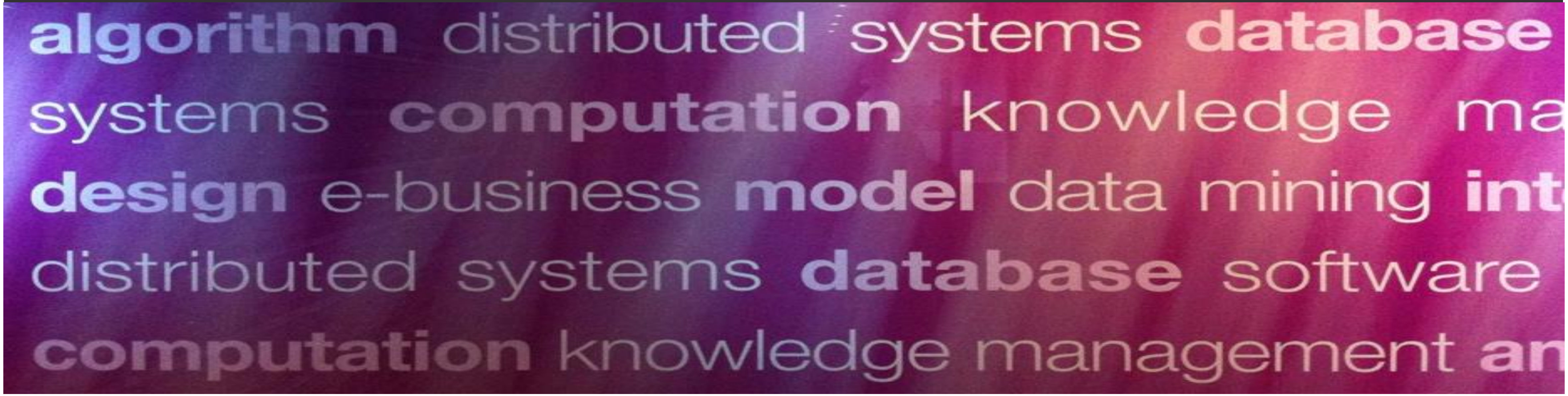
MONASH University

Information Technology

FIT3143 - LECTURE WEEK 9b

PARALLEL ALGORITHM DESIGN – MATRIX MULTIPLICATION USING FOX & CANNON WITH VIRTUAL TOPOLOGIES & COLLECTIVE COMMUNICATIONS



Topic Overview

- Quick recap of the matrix multiplication algorithm
- Fox algorithm for parallel matrix multiplication
- Cannon algorithm for parallel matrix multiplication

Learning outcome(s) related to this topic

- Design and develop parallel algorithms for various parallel computing architectures (LO3)

These slides and enclosed sample code files were prepared by Shageenderan Sapai, PhD postgraduate student, Monash University.

Quick recap of the matrix multiplication algorithm

Matrix multiplication

Assume we have 2 n x n matrices, A and B, and we want to find C such that C = A x B

1	4	8
2	5	6
1	5	6

A

X

7	3	4
2	9	6
7	1	9

B

=

71	47	100
66	57	92
59	54	88

C

$$c_{ij} = \left(a_i, b_j^T\right) = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, \; 0 \le i < m, \; 0 \le j < l$$

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} 1 & 4 & 8 \\ 2 & 5 & 6 \\ 1 & 5 & 6 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 7 & 3 & 4 \\ 2 & 9 & 6 \\ 7 & 1 & 9 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 71 & & \\ & & \\ & & \end{bmatrix}}$$

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} 1 & 4 & 8 \\ 2 & 5 & 6 \\ 1 & 5 & 6 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 7 & 3 & 4 \\ 2 & 9 & 6 \\ 7 & 1 & 9 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 71 & 47 & \\ & & \\ & & \end{bmatrix}}$$

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} 1 & 4 & 8 \\ 2 & 5 & 6 \\ 1 & 5 & 6 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 7 & 3 & 4 \\ 2 & 9 & 6 \\ 7 & 1 & 9 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 71 & 47 & 100 \\ & & \\ & & \end{bmatrix}}$$

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} 1 & 4 & 8 \\ 2 & 5 & 6 \\ 1 & 5 & 6 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 7 & 3 & 4 \\ 2 & 9 & 6 \\ 7 & 1 & 9 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 71 & 47 & 100 \\ 66 & & \\ & & \end{bmatrix}}$$

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} 1 & 4 & 8 \\ 2 & 5 & 6 \\ 1 & 5 & 6 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 7 & 3 & 4 \\ 2 & 9 & 6 \\ 7 & 1 & 9 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 71 & 47 & 100 \\ 66 & 57 & 92 \\ 59 & 54 & 88 \end{bmatrix}}$$

Serial Algorithm

```
A and B nxn matrices =>  C=AxB

void Serial_mat_mult(mt_A,mt_B,mt_C,int n)
{
  int i,j,k;
for (i=0;i<n;i++){
    for (j=0;j<n;j++){
       C[i][j]=0.0;
       for (k=0;k<n;k++)
           C[i][j]= C[i][j]+A[i][k]*B[k][j];
   }
}
```

Time Complexity O(n^3) – very slow

Parallel matrix multiplication algorithm - Fox

Parallel Matrix Multiplication Algorithm - General

```
A and B nxn matrices =>  C=AxB
```

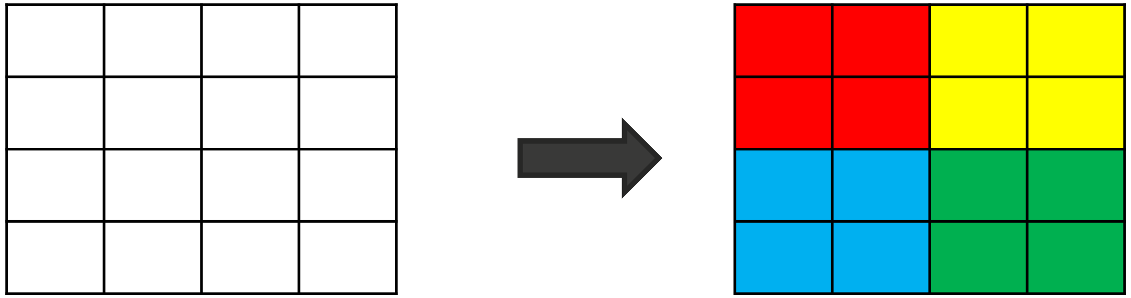
- First, Matrix A & B are partitioned so that each process works on a specific submatrix of A & B.
- Each process does a serial matrix multiplication on their local submatrices of A & B and the results are stored in an accumulating sum
- Local submatrices are then communicated around to calculate the final result
- There are different algorithms for how submatrices are communicated, mainly, **Fox algorithm** and **Cannon algorithm**

Parallel Algorithm – Data Partitioning

- Parallel matrix multiplication requires data partitioning – common methods are Block-Row Partitioning or Block-Column Partitioning.
- Another method is grid or checkerboard partitioning – which is used in Fox and Cannon algorithm
- Say we have *p* number of processes and a square Matrix, M, of size nxn and let q = sqrt(*p*)
- M is partitioned into *p* square blocks (sub matrices) $M_{i,j}$ where 0 <= i, j < q of size n/qxn/q

Say p = 4 and n = 4

Each submatrix = n/q = 4/sqrt(4) = 2



M

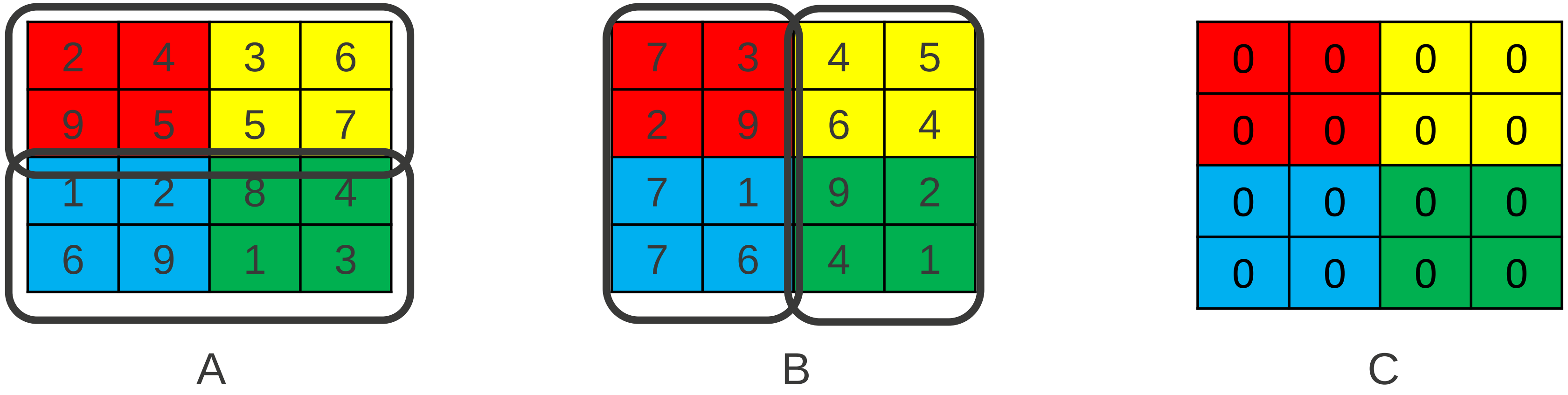
$M_{0,0}$	Rank 0
$M_{0,1}$	Rank 1
$M_{1,0}$	Rank 2
$M_{1,1}$	Rank 3

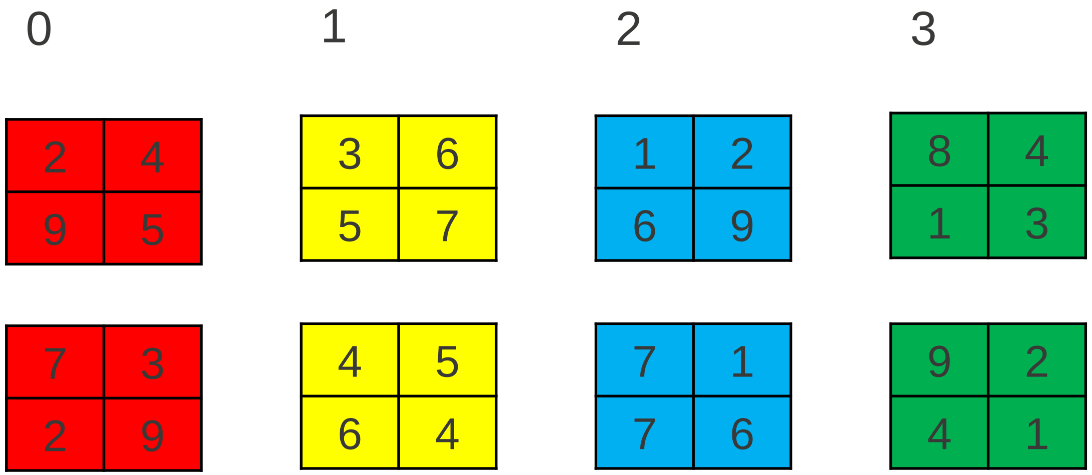
Fox Algorithm

- Matrix A & B are **grid partitioned** into submatrices $A_{i,j}$ and $B_{i,j}$
- In each step, a one-to-all broadcast of the rows of matrix A and a single-step circular upwards shift of columns in matrix B is done

- Loop q times
  1. Broadcast diagonal block $A_{i,i}$ to all processors in the same row group
  2. Multiply block of received $A_{i,i}$ with local block $B_{i,j}$ and add to $C_{i,j}$
  3. Send (shift) entire local block B up within the same column group
  4. Select block $A_{i,(j+1) \bmod q}$ (where $A_{i,j}$ is the block broadcast in the previous step) and broadcast to all processors in the same row group.
  5. Go to 2.
- Gather all

Data Partitioning – Fox Algorithm





Groups	Processors
Row Group	[0,1] [2, 3]
Column Group	[0,2] [1,3]

Loop q times

- Broadcast diagonal block $A_{i,i}$ to all processors in the same row group

A ($A_{0,0}$ is the top-left 2×2 block)

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

B

7	3	4	5
2	9	6	4
7	1	9	2
7	6	4	1

C

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0

$A_{0,0}$

2	4
9	5

$A_{i,i}$

0	0
0	0

$B_{0,0}$

7	3
2	9

1

$A_{0,1}$

3	6
5	7

$A_{i,i}$

0	0
0	0

$B_{0,1}$

4	5
6	4

2

$A_{1,0}$

1	2
6	9

$A_{i,i}$

0	0
0	0

$B_{1,0}$

7	1
7	6

3

$A_{1,1}$

8	4
1	3

$A_{i,i}$

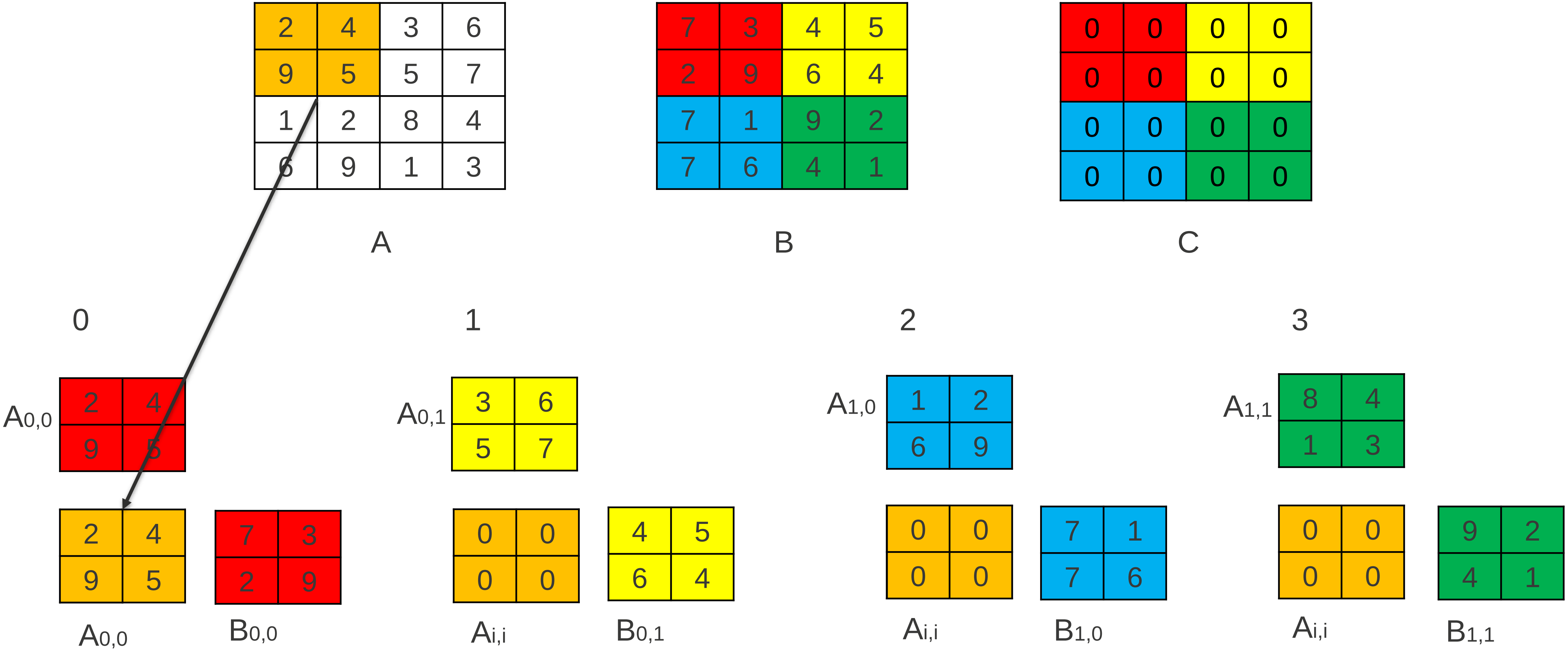
0	0
0	0

$B_{1,1}$

9	2
4	1

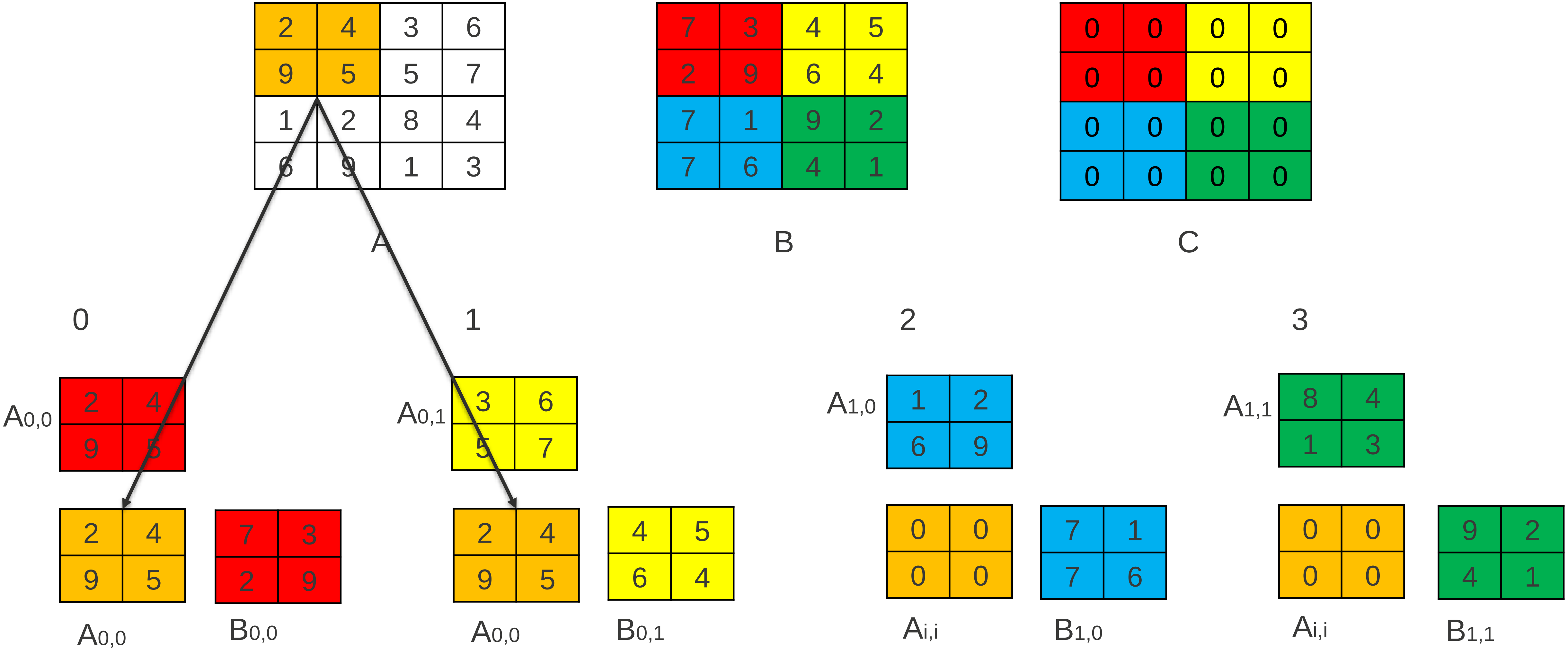
Loop q times

- Broadcast diagonal block $A_{i,i}$ to all processors in the same row group



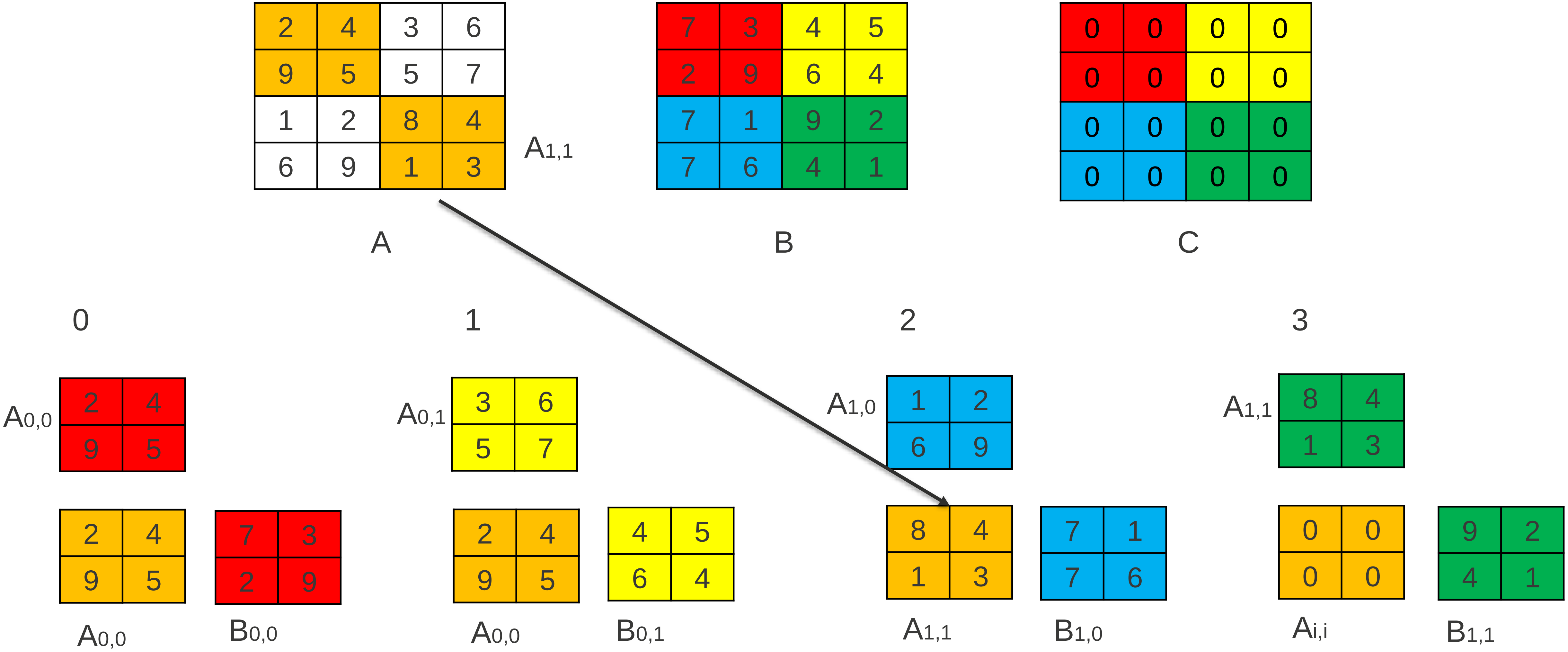
Loop q times

- Broadcast diagonal block $A_{i,i}$ to all processors in the same row group



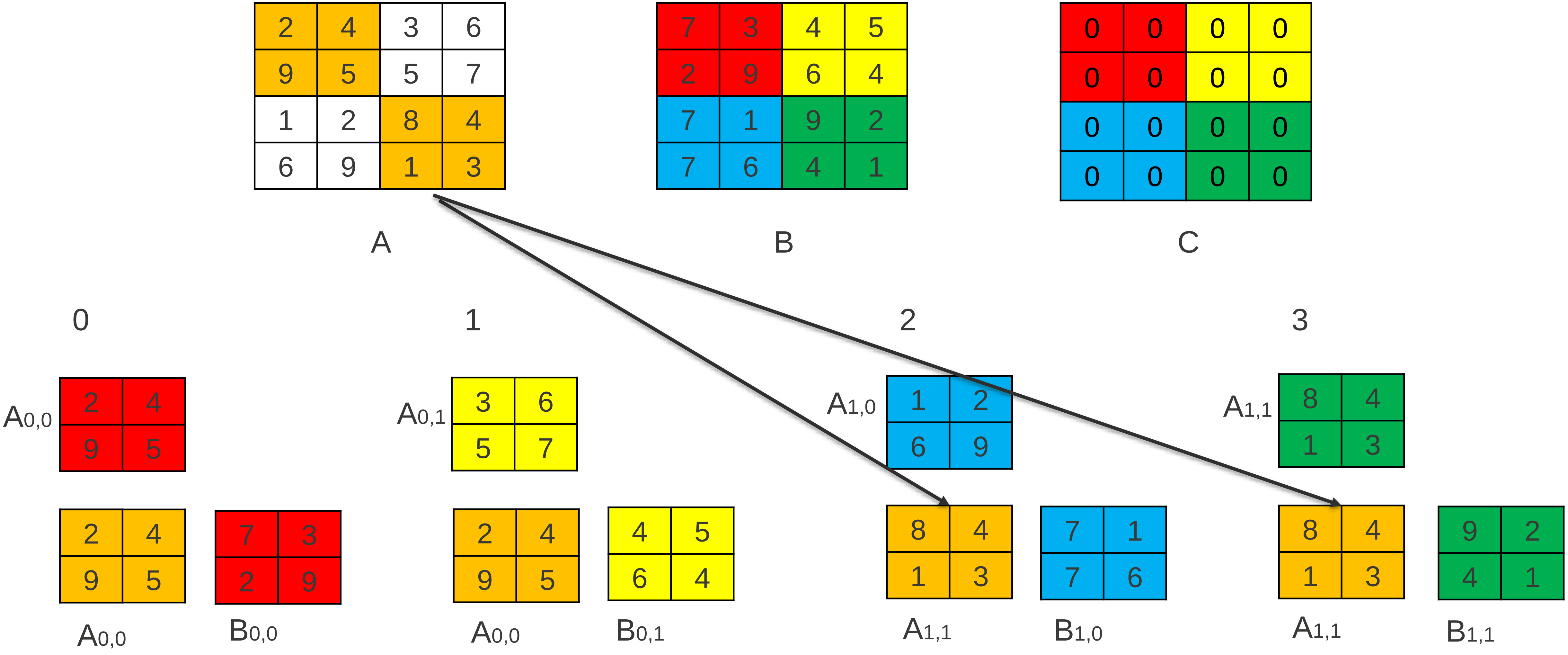
Loop q times

- Broadcast diagonal block $A_{i,i}$ to all processors in the same row group



Loop q times

- Broadcast diagonal block $A_{i,i}$ to all processors in the same row group



Loop q times

- Multiply block of received A with resident block B and add to C

A

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

B

7	3	4	5
2	9	6	4
7	1	9	2
7	6	4	1

C

22	42	32	26
73	72	66	65
84	32	88	20
28	19	21	5

0

$A_{0,0}$

2	4
9	5

2	4
9	5

x

7	3
2	9

$A_{0,0}$ $B_{0,0}$

1

$A_{0,1}$

3	6
5	7

2	4
9	5

x

4	5
6	4

$A_{0,0}$ $B_{0,1}$

2

$A_{1,0}$

1	2
6	9

8	4
1	3

x

7	1
7	6

$A_{1,1}$ $B_{1,0}$

3

$A_{1,1}$

8	4
1	3

8	4
1	3

x

9	2
4	1

$A_{1,1}$ $B_{1,1}$

Loop q times

- Multiply block of received A with resident block B and add to C

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

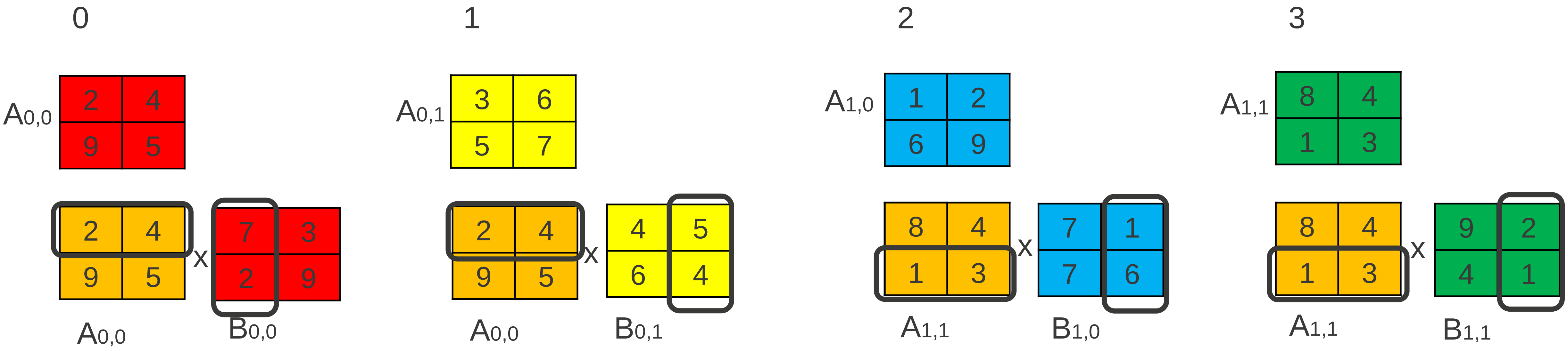
A

7	3	4	5
2	9	6	4
7	1	9	2
7	6	4	1

B

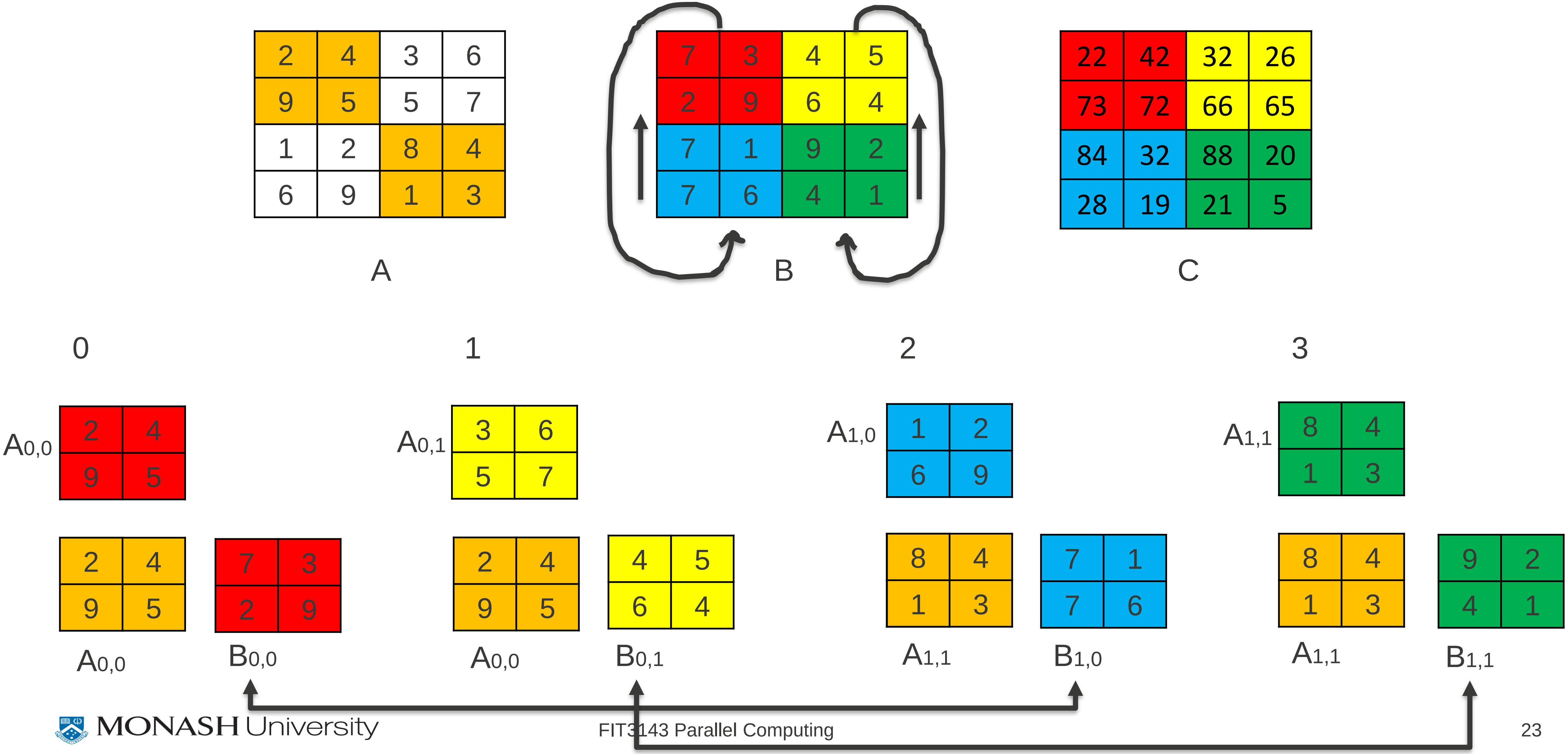
22	42	32	26
73	72	66	65
84	32	88	20
28	19	21	5

C



Loop q times

1. Send (shift) entire local block B up within the same column group



Loop q times

- Send (shift) **entire** resident block B up one step

A

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

B

7	1	9	2
7	6	4	1
7	3	4	5
2	9	6	4

C

22	42	32	26
73	72	66	65
84	32	88	20
28	19	21	5

0

$A_{0,0}$

2	4
9	5

$A_{0,0}$

2	4
9	5

$B_{0,0}$

7	1
7	6

1

$A_{0,1}$

3	6
5	7

$A_{0,0}$

2	4
9	5

$B_{0,1}$

9	2
4	1

2

$A_{1,0}$

1	2
6	9

$A_{1,1}$

8	4
1	3

$B_{1,0}$

7	3
2	9

3

$A_{1,1}$

8	4
1	3

$A_{1,1}$

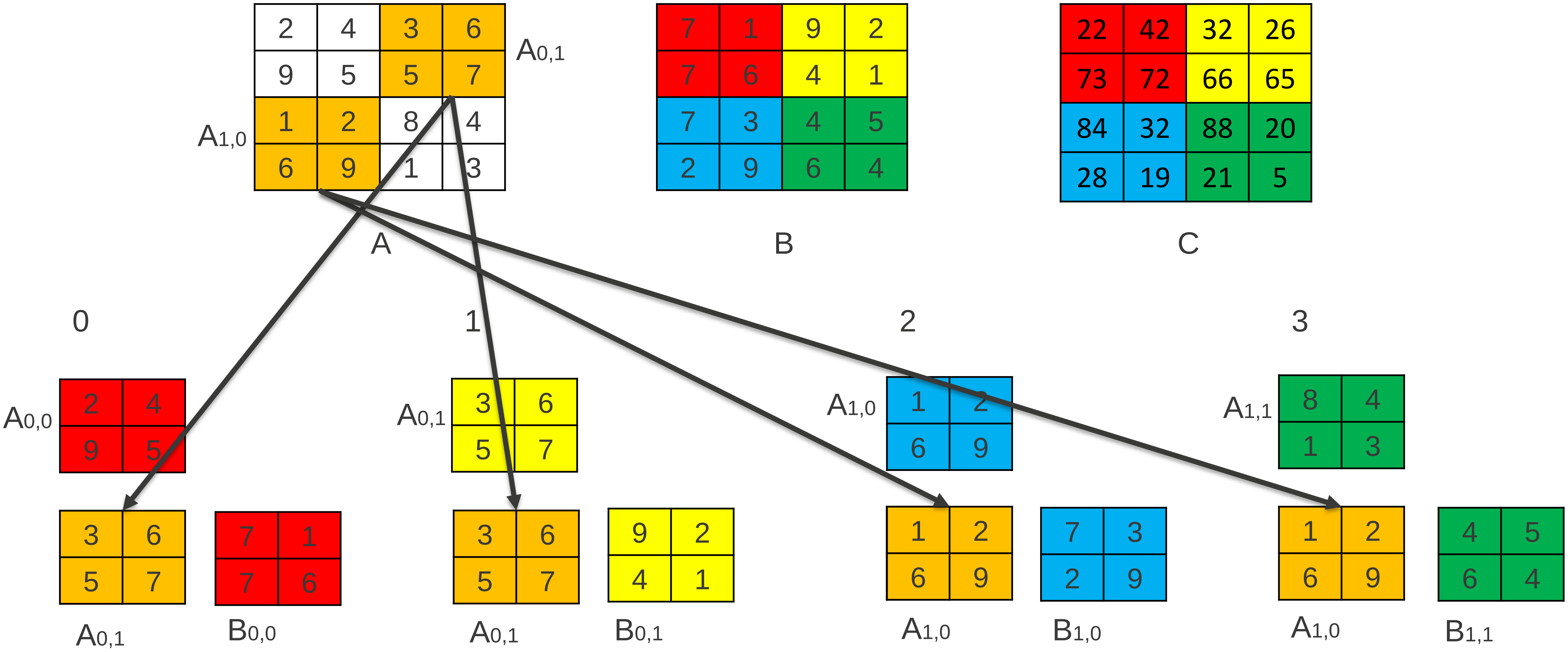
8	4
1	3

$B_{1,1}$

4	5
6	4

Loop q times

- Broadcast $A_{i,(j+1) \bmod q}$ to all processors in the same row group



Loop q times

- Multiply block of received A with resident block B and add to C

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

A

7	1	9	2
7	6	4	1
7	3	4	5
2	9	6	4

B

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

0

$A_{0,0}$

2	4
9	5

3	6
5	7

x

7	1
7	6

$A_{0,1}$ $B_{0,0}$

1

$A_{0,1}$

3	6
5	7

3	6
5	7

x

9	2
4	1

$A_{0,1}$ $B_{0,1}$

2

$A_{1,0}$

1	2
6	9

1	2
6	9

x

7	3
2	9

$A_{1,0}$ $B_{1,0}$

3

$A_{1,1}$

8	4
1	3

1	2
6	9

x

4	5
6	4

$A_{1,0}$ $B_{1,1}$

Loop q times

- Multiply block of received A with resident block B and add to C

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

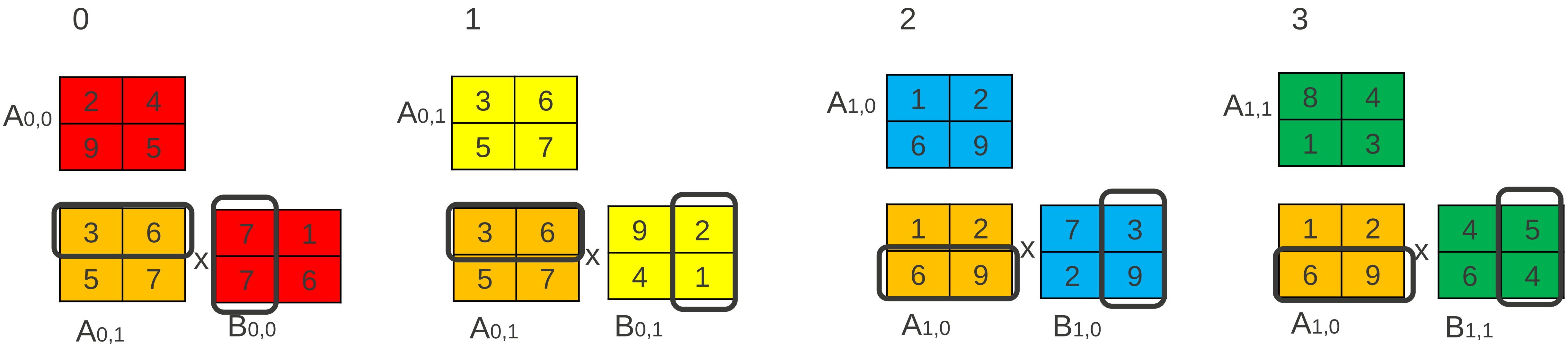
A

7	1	9	2
7	6	4	1
7	3	4	5
2	9	6	4

B

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C



- Already looped q(2) times, so now we must:
- Gather All

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

Gathered results on Process 0

Fox Algorithm

- Tends to be faster than simple parallel matrix multiplication for large values of n
- Implemented in MPI by creating row and column communicators which enable easy communication of A & B blocks
- Although dealing with 2D matrices, in coding implementation these are represented using 1D arrays
- Difficult to adapt for non-square matrices i.e. A (mxn) x B (nxm)
- Has high communication overhead because at each step we are sending (relatively) large blocks of local A & B.

Parallel matrix multiplication algorithm - Cannon

Cannon Algorithm

- Matrix A & B are **grid partitioned** into submatrices $A_{i,j}$ and $B_{i,j}$
- Data is moved incrementally in q-1 phases (ring broadcast algorithm)

- Each process skews matrix $A_{i,j}$ and $B_{i,j}$ to align elements by shifting $A_{i,j}$ by i rows to the left and shifting $B_{i,j}$ by j columns up.
- Multiply local block A with local block B and add to local C
- Loop q-1 times
  1. Shift elements (each A_i row by 1 unit and each B_j column by 1 unit)
  2. Multiply local block A with local block B and add to local C
- Gather all

Data Partitioning – Cannon Algorithm

A

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

B

7	3	4	5
2	9	6	4
7	1	9	2
7	6	4	1

C

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0

2	4
9	5

7	3
2	9

1

3	6
5	7

4	5
6	4

2

1	2
6	9

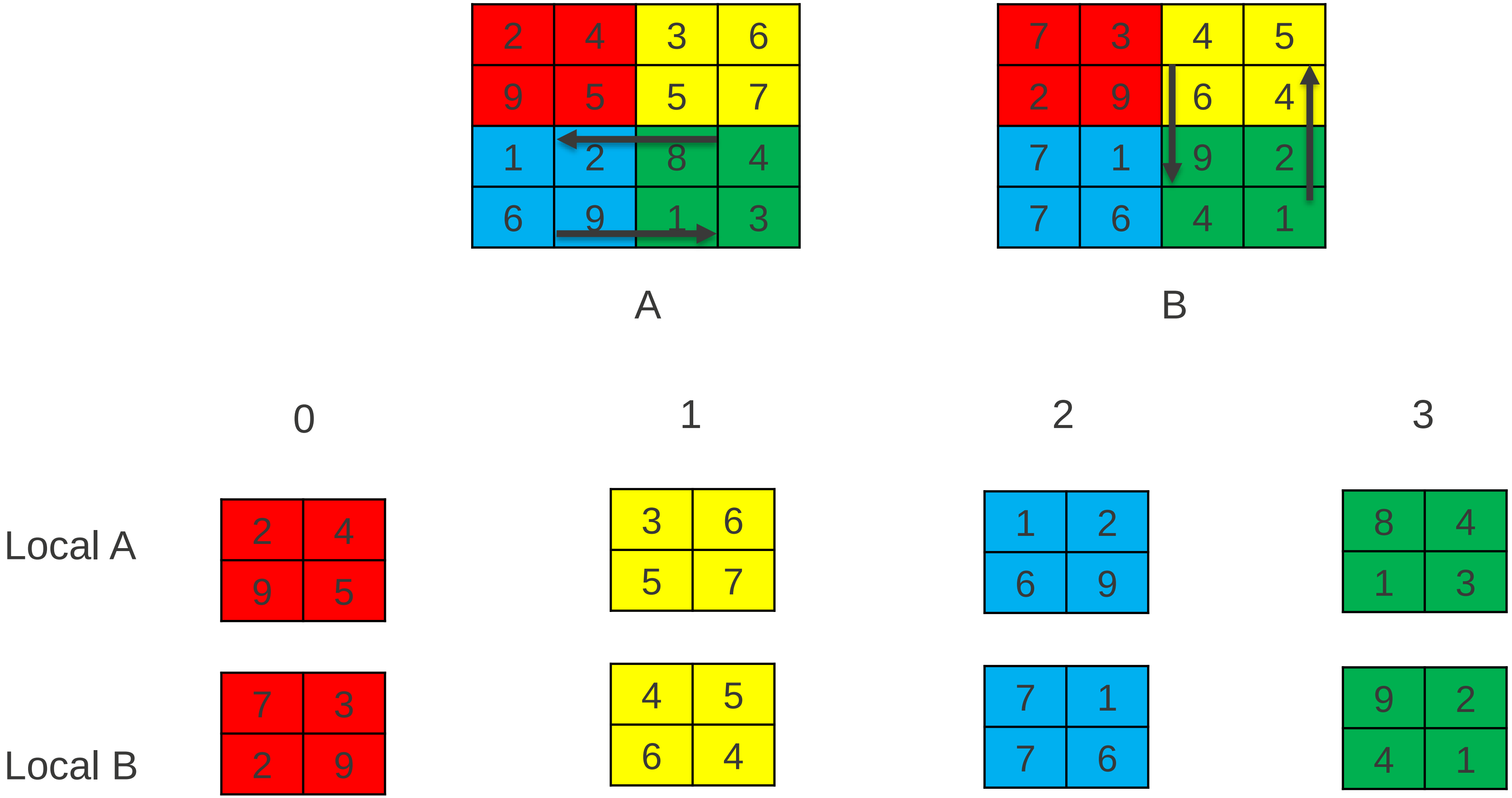
7	1
7	6

3

8	4
1	3

9	2
4	1

Skew matrix A and B to align elements by shifting $A_{i,j}$ by i rows to the left and shift $B_{i,j}$ by j columns up.



Skew matrix A and B to align elements by shifting A_i by i rows to the left and shift B_i by i columns up.

2	4	3	6
9	5	5	7
1	2	8	4
6	9	1	3

A

7	3	4	5
2	9	6	4
7	1	9	2
7	6	4	1

B

0

Local A

2	4
9	5

Local B

7	3
2	9

1

Local A

3	6
5	7

Local B

9	2
4	1

2

Local A

8	4
1	3

Local B

7	1
7	6

3

Local A

1	2
6	9

Local B

4	5
6	4

Skew matrix A and B to align elements by shifting A_i by i rows to the left and shift B_i by i columns up.

A

2	4	3	6
9	5	5	7
8	4	1	2
1	3	6	9

B

7	3	9	2
2	9	4	1
7	1	4	5
7	6	6	4

0

Local A

2	4
9	5

Local B

7	3
2	9

1

Local A

3	6
5	7

Local B

9	2
4	1

2

Local A

8	4
1	3

Local B

7	1
7	6

3

Local A

1	2
6	9

Local B

4	5
6	4

Loop q-1 times

1. Multiply elements and add to accumulating sum

2	4	3	6
9	5	5	7
8	4	1	2
1	3	6	9

A

7	3	9	2
2	9	4	1
7	1	4	5
7	6	6	4

B

22	42	51	12
73	72	73	17
84	32	16	13
28	19	78	66

C

0

2	4
9	5

x

7	3
2	9

Local A Local B

1

3	6
5	7

x

9	2
4	1

Local A Local B

2

8	4
1	3

x

7	1
7	6

Local A Local B

3

1	2
6	9

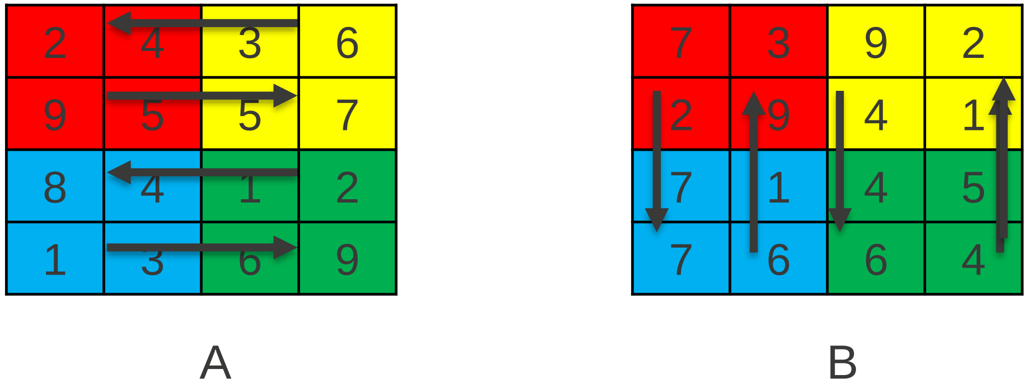
x

4	5
6	4

Local A Local B

Loop q1 times

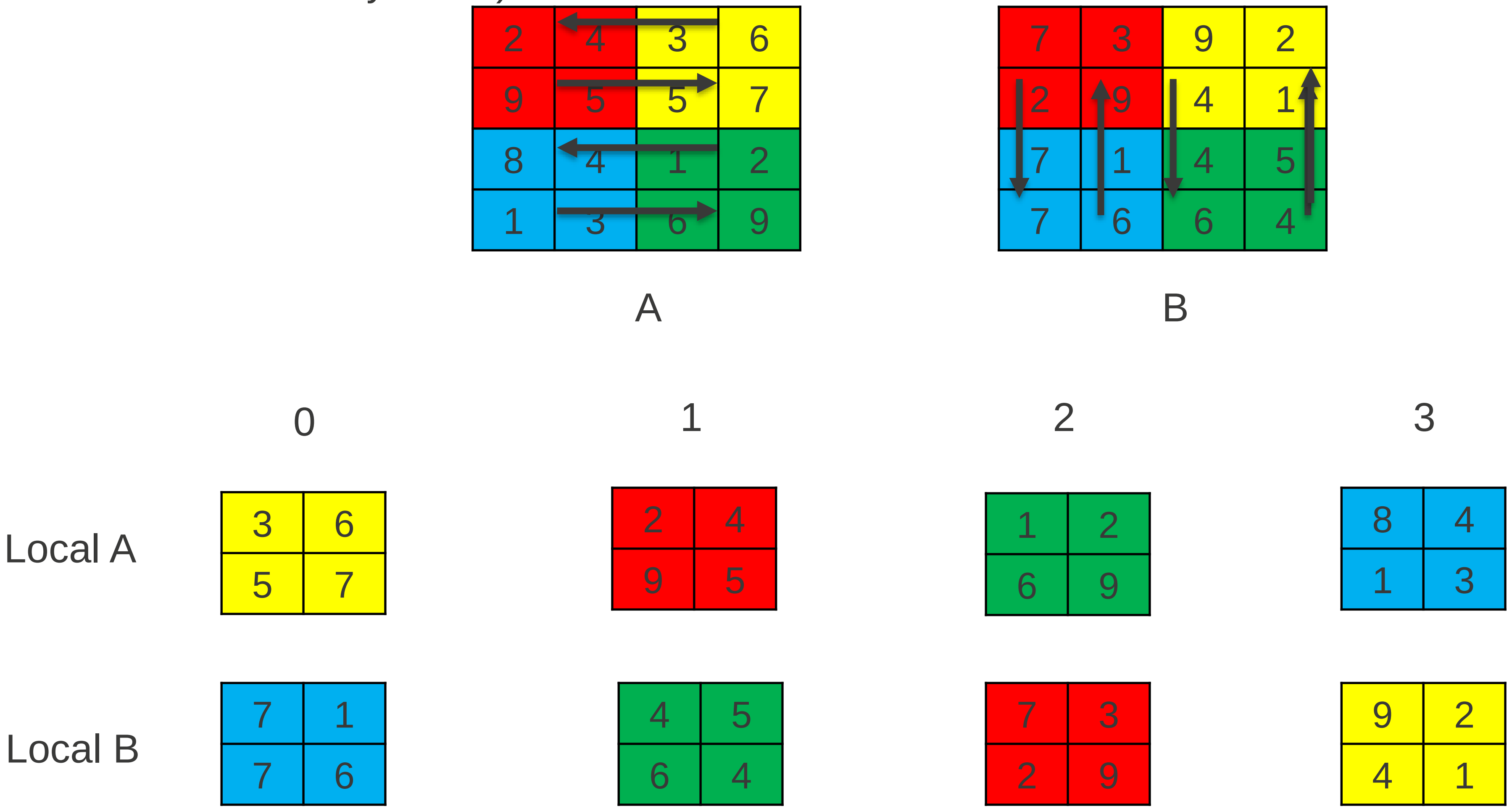
1. Shift elements (each A_i row by 1 unit and each B_i column by 1 unit)



	0		1		2		3	
Local A	2	4	3	6	8	4	1	2
	9	5	5	7	1	3	6	9
Local B	7	3	9	2	7	1	4	5
	2	9	4	1	7	6	6	4

Loop q-1 times

1. Shift elements (each A_i row by 1 unit and each B_i column by 1 unit)



Loop q-1 times

1. Shift elements (each A_i row by 1 unit and each B_i column by 1 unit)

3	6	2	4
5	7	9	5
1	2	8	4
6	9	1	3

A

7	1	4	5
7	6	6	4
7	3	9	2
2	9	4	1

B

	0	1	2	3
Local A	3 6 5 7	2 4 9 5	1 2 6 9	8 4 1 3
Local B	7 1 7 6	4 5 6 4	7 3 2 9	9 2 4 1

Loop q-1 times

1. Multiply local block A with local block B and add to local C

3	6	2	4
5	7	9	5
1	2	8	4
6	9	1	3

A

7	1	4	5
7	6	6	4
7	3	9	2
2	9	4	1

B

22	42	51	12
73	72	73	17
84	32	16	13
28	19	78	66

C

0

$$\begin{bmatrix} 3 & 6 \\ 5 & 7 \end{bmatrix} \times \begin{bmatrix} 7 & 1 \\ 7 & 6 \end{bmatrix}$$

Local A Local B

1

$$\begin{bmatrix} 2 & 4 \\ 9 & 5 \end{bmatrix} \times \begin{bmatrix} 4 & 5 \\ 6 & 4 \end{bmatrix}$$

Local A Local B

2

$$\begin{bmatrix} 1 & 2 \\ 6 & 9 \end{bmatrix} \times \begin{bmatrix} 7 & 3 \\ 2 & 9 \end{bmatrix}$$

Local A Local B

3

$$\begin{bmatrix} 8 & 4 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} 9 & 2 \\ 4 & 1 \end{bmatrix}$$

Local A Local B

Loop q-1 times

1. Multiply local block A with local block B and add to local C

3	6	2	4
5	7	9	5
1	2	8	4
6	9	1	3

A

7	1	4	5
7	6	6	4
7	3	9	2
2	9	4	1

B

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

0

3	6
5	7

x

7	1
7	6

Local A Local B

1

2	4
9	5

x

4	5
6	4

Local A Local B

2

1	2
6	9

x

7	3
2	9

Local A Local B

3

8	4
1	3

x

9	2
4	1

Local A Local B

- Already looped q-1(1) times, so now we must:
- Gather All

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

85	81	83	38
157	119	139	82
95	53	104	33
88	118	99	71

C

Gathered results on Process 0

Cannon Algorithm

- Tends to be faster than simple parallel matrix multiplication for large values of n
- Implemented in MPI by creating row and column communicators which enable easy shifting of data
- Although dealing with 2D matrices, in coding implementation these are represented using 1D arrays
- Difficult to adapt for non-square matrices i.e. A (mxn) x B (nxm)
- Has lower communication overhead than Fox algorithm because at each step we are sending less data than in Fox

Cannon Algorithm – Bigger Example

2	4	3	6	4	4
9	5	5	7	3	3
1	2	8	4	2	2
6	9	1	3	1	1
1	2	8	4	2	2
6	9	1	3	1	1

A

7	3	4	5	4	4
2	9	6	4	5	5
7	1	9	2	6	6
7	6	4	1	7	7
7	1	9	2	6	6
7	6	4	1	7	7

B

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

C

Skew

2	4	3	6	4	4
9	5	5	7	3	3
8	4	2	2	1	2
1	3	1	1	6	9
2	2	1	2	8	4
1	1	6	9	1	3

A

7	3	9	2	6	6
2	9	4	1	7	7
7	1	9	2	4	4
7	6	4	1	5	5
7	1	4	5	6	6
7	6	6	4	7	7

B

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

C

Multiply

2	4	3	6	4	4
9	5	5	7	3	3
8	4	2	2	1	2
1	3	1	1	6	9
2	2	1	2	8	4
1	1	6	9	1	3

A

7	3	9	2	6	6
2	9	4	1	7	7
7	1	9	2	4	4
7	6	4	1	5	5
7	1	4	5	6	6
7	6	6	4	7	7

B

22	42	51	12	52	52
73	72	73	17	39	39
84	32	26	6	14	14
28	19	13	3	69	69
28	14	16	13	76	76
14	7	78	66	27	27

C

Skew

3	6	4	4	2	4
5	7	3	3	9	5
2	2	1	2	8	4
1	1	6	9	1	3
1	2	8	4	2	2
6	9	1	3	1	1

A

7	1	9	2	4	4
7	6	4	1	5	5
7	1	4	5	6	6
7	6	6	4	7	7
7	3	9	2	6	6
2	9	4	1	7	7

B

22	42	51	12	52	52
73	72	73	17	39	39
84	32	26	6	14	14
28	19	13	3	69	69
28	14	16	13	76	76
14	7	78	66	27	27

C

Multiply

3	6	4	4	2	4
5	7	3	3	9	5
2	2	1	2	8	4
1	1	6	9	1	3
1	2	8	4	2	2
6	9	1	3	1	1

A

7	1	9	2	4	4
7	6	4	1	5	5
7	1	4	5	6	6
7	6	6	4	7	7
7	3	9	2	6	6
2	9	4	1	7	7

B

85	81	103	24	80	80
157	119	112	26	100	100
112	46	42	19	90	90
42	26	91	69	96	96
39	35	104	33	102	102
74	106	99	71	40	40

C

Skew

4	4	2	4	3	6
3	3	9	5	5	7
1	2	8	4	2	2
6	9	1	3	1	1
8	4	2	2	1	2
1	3	1	1	6	9

A

7	1	4	5	6	6
7	6	6	4	7	7
7	3	9	2	6	6
2	9	4	1	7	7
7	1	9	2	4	4
7	6	4	1	5	5

B

85	81	103	24	80	80
157	119	112	26	100	100
112	46	42	19	90	90
42	26	91	69	96	96
39	35	104	33	102	102
74	106	99	71	40	40

C

Multiply

4	4	2	4	3	6
3	3	9	5	5	7
1	2	8	4	2	2
6	9	1	3	1	1
8	4	2	2	1	2
1	3	1	1	6	9

A

7	1	4	5	6	6
7	6	6	4	7	7
7	3	9	2	6	6
2	9	4	1	7	7
7	1	9	2	4	4
7	6	4	1	5	5

B

141	109	135	50	140	140
199	140	178	91	179	179
123	67	130	39	116	116
102	125	112	74	109	109
123	67	130	39	116	116
102	125	112	74	109	109

C

- Already looped q-1(2) times, so now we must:
- Gather All

141	109	135	50	140	140
199	140	178	91	179	179
123	67	130	39	116	116
102	125	112	74	109	109
123	67	130	39	116	116
102	125	112	74	109	109

C

→

141	109	135	50	140	140
199	140	178	91	179	179
123	67	130	39	116	116
102	125	112	74	109	109
123	67	130	39	116	116
102	125	112	74	109	109

C

Gathered results on Process 0

C code files implementing fox and cannon parallel matrix multiplication with MPI

- Fox – Click here
- Cannon – Click here