Unit-04

PAGE NO:
DATE / /

Analytical models of parallel programs.

- Understanding parallel system
- Whenever we are talking about a parallel system it is a combination of parallel system algo and parallel architecture. → This is most important part of overhead.
- A sequential algo is usually evaluated in terms of its execution time, expressed as a function of the size of its input.
- The parallel runtime of a program depend on the input size, but also on the number of processing elements used and communication parameters of the machine.
- Therefore, execution time of parallel algo = input size + number of processing elements.

* **Sources of overhead in parallel programs** -
  What are the difficulties occur in parallel programs.
- If we increase the number of processors then can we get execute the program faster.
- The answer is number practically in parallel program this is easy possible due to number of overhead associated with parallelism.
- In addition to performing computation (i.e. computation would be performed by serial program for solving some problem), communication, idling and contention cause degradation in performance.

unication is such thing.

A typical execution profile of a parallel program is In this execution profile of a hypothetical parallel program executing on eight processing elements.

profile indicates time spend on - or what are the diff overhead when we are using parallel programming

1) Inter process communication
2) Iding
3) Express computation.

Inter process communication -

The processing elements in any parallal system need to communicate with each other for exchange of control and specially inform data.

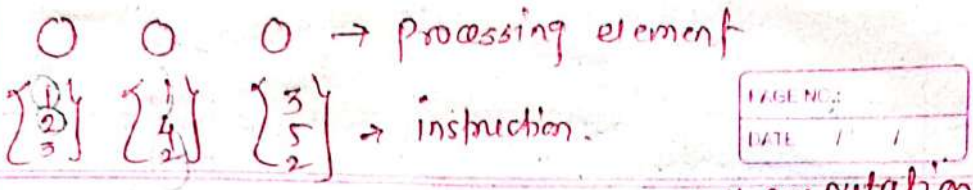
the major source of parallel processing overhead is time spend to interact and communication data betn processing elements.

If there is data dependency exists in problem and the number of processing elements working on the same data then it requires interprocess communication., so even if two processing elements are independent still they cannot work in parallel.

Iding (load imbalance) - → Non-uniform distribution of task.

processing elements in parallel system may become idle due to many reasons such as load imbalanced, synchronization and presence of serial components in a program.

In dynamic task generation, it is very difficult to predict the size of the subtasks assigned to processors in advance.



- Express computation → unnessary computation
- this is computation not performed by the serial version. this might be because of serial algo is difficult to parallelize, or that some computation are repeated across processors to minimize communication.

- Express computation = (computation performed by parallel program) - (computation for best serial algo).

- In other words
  Express = parallel - best serial.

- so we try to generate a parallel program that works like a best serial program but because of different factors parallel code takes more than the best serial program.

☆ Performance Metrics for parallel systems

The performance of parallel algo are analysed to determined the best algo and the system on which it can be executed most efficiently.

- there are number of metrics used based on the analysis of peeformance of parallel programs.
  1) execution time
  2) speedup
  3) Total parallel overhead
  4) efficiency
  5) cost.

PAGE NO:
DATE / /

1) Execution time – given ~~time~~ task required how many time for execution in system.

- the serial runtime (Ts) - the time elapse betn the start of the execution and the end of the execution by the sequential computer is serial runtime.

- Parallel runtime (Tp) - It is time that elapes from the the moment the first processor starts to the moment the last processor finishes its execution.

- Let Tall be the total time collectively spent by all the processing elements
  we have

$$T_{all} = p \times T_p$$

- observe that Tall - Tp is total time spend by all combined processors in non-useful work. this is called the total overhead.
  The overhead function is given by

$$T_o = pT_p - T_s$$

2) Total overhead - Non-useful Work (Wasted).

- The total overhead of a parallel system is defined as the total time collectively spent by all the processing elements over and above that required by the fastest known sequential algo for solving the same problem on a single processing elements.

- consider there are p number of processing units, then the total time spent in solving a problem summed over all processing elements is $p * T_p$.

- Ts unit of this time are spent performing useful work, and the remainder is overhead.

- function overhead (T_o) is given by

$$T_o = p * T_p * T_s$$

3) Speedup -

While deal with parallel computations we always ask a question that, what benefit we get from parallelism? After all what we want program execution should be completed at a shorter time, so we need to compute the speedup.

- In other word we can say that How much faster your code run in parallel over serial execution.
- It evaluates the benefit of solving a problem in parallel.

- It is the ratio of time taken to solve a problem on single processing element to the time required to solve the same problem on a parallel computer with 'p' identical processing elements.

$$S = T_s / T_p$$

eg. calculating speedup for adding 'n' number using 'n' processing elements.

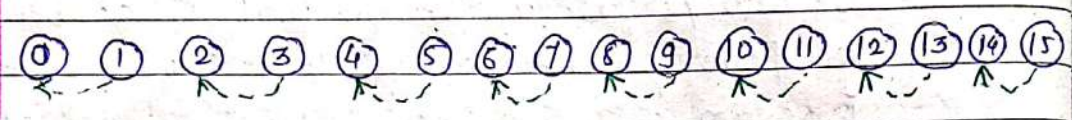
- Initially each processing element is assigned one of the numbers to be added and at the end of the computation, one the processing elements stores the sum of all the numbers.

- Assuming that 'n' is power of two, we can perform this operation in $\log(n)$ steps by propagating partial sum up a logical binary tree of processing elements.

PAGE NO.:
DATE / /

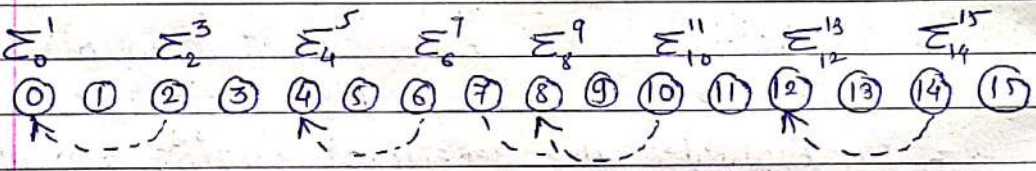
Here procedure for n=16
processing elements are labeled from 0 to 15

- 114, the 16 numbers to be added are labeled from 0 to 15. The sum of the numbers with consecutive labels from i to j is denoted by E_i^j.

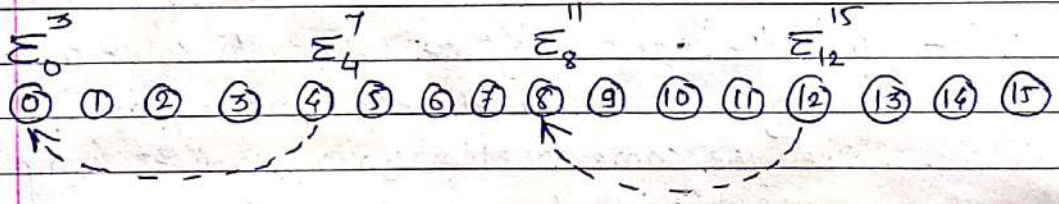


① first communication step

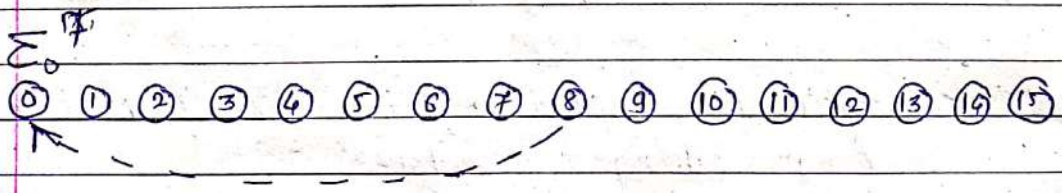
holding the addn of 0 & 1



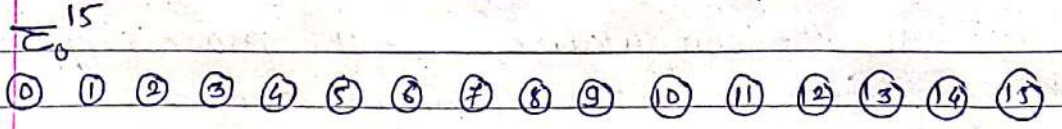
② second communication step



③ Third communication step



④ Fourth communication step



⑤ Accumulation of the sum at processing element 0 after the final communication

PAGE NO.:
DATE / /

- If an addition takes constant time, Tc and communication of a single word takes time Ts+Tw we have the parallel time

$T_P = \Theta(\log n)$

we know that $T_S = \Theta(n)$

speedup s is given by

$$\boxed{S = \Theta(n/\log n)}$$

= Speedup bounds -

eg. Adding 16 numbers

serial time = T_S = 16 steps = 16 unit ie

$T_S = \Theta(n)$ ie $N = 16$

Parallel time = T_P = 4 steps = 4 unit

$T_P = \Theta(\log n) = \log 16 = \log n$

$= \log 16 = 4$

$$\boxed{\text{speedup} = \Theta\left(\frac{n}{\log n}\right)} \Rightarrow \frac{16}{4} = \underline{4}$$

your parallel program is 4 times faster than serial.

PAGE NO.:
DATE / /

4) Efficiency -

parallel system contain 'p' processing element can deliver a speedup equal to p. but in practically this is not achieved, Why because while executing a parallel algorithm, the processing elements cannot loyal 100% of their time to the computation of the algo.

- So, Efficiency is a measure of the fraction of time for which a processing element (processor) is engaged.

- Therefore efficiency is defined as ratio of speedup to number of processing elements

ie.

$$E = \frac{S}{P} \quad \text{---------} (1)$$

lowest efficiency = 0, highest efficiency = 1.

eg. Adding n numbers on 'n' processing element

$$S = \Theta\left(\frac{n}{\log n}\right) \qquad P = n$$

and the efficiency is given by

$$E = \frac{\Theta\left(\frac{n}{\log n}\right)}{n}$$

$$E = \Theta\left(\frac{1}{\log n}\right)$$

PAGE NO.:
DATE / /

5) Cost of parallel system -

→ The cost of solving problem on parallel system is a product of parallel runtime and number of processing element used.
ie (P × Tp).

- So, cost reflect the sum of time that each processing elements spends solving the problem.

- Cost is also called product processor time W and calculated as product of parallel runtime & number of processing elements used.

ie $W = Tp * P$

- A cost-optimal system is also known as p*TP -optimal system.

- $$E = \frac{(Ts)/(Tp)}{P}$$

$$E = \frac{Ts}{P * TP}$$

cost (P*TP) is sometime referred to as work or processor time product.

PAGE NO.:
DATE / /

* Minimum Execution time and minimum cost-optimal Execution time.

- We are interested in how fast the problem can be solved or what the minimum possible execution time of a parallel algorithm.

- As we increase the number of processing element for a given problem size, either the parallel runtime continue to decreases,

- We can determine the minimum parallel runtime T_p^{min} given W. by differentiating the expression for Tp. With respect to p and equating the derivation to zero.

- The number of processing element for which TP is minimum is determined by eq's

$$\frac{d}{dp} TP = 0$$

Let Po is the value of number of processing element, value of T_p^{min} (minimum parallel runtime) can be determined by substituting po for in expression for Tp.

eg consider minimum execution time for adding n. numbers

we know that, the parallel runtime for problem of adding n numbers on 'p' processing element is

$$TP = \frac{n}{p} + 2 \log p.$$

After equating the derivation with respect to p of the right side to zero

DATE / /

$$-\frac{n}{p^2} + \frac{2}{p} = 0$$

$$-n + 2p = 0$$

substitute $p = n/2$ we get

$$T_p^{min} = 2 \log n$$

for this example, the processor time product for $p = p_0$ ie $\Theta(n \log n)$, which is higher than $\Theta(n)$ serial complexity of the problem.

- So, parallel system is not cost-optimal for value of p.

* Minimum cost optimal execution time –

- Let T_p^{min} be the minimum time in which a problem can be solved by cost-optimal parallel system.
- Let $T_p^{cost-opt}$ be the minimum cost-optimal parallel system.
- From the previous point, the isoefficiency function $f(p)$ of parallel system is $\Theta(p \log p)$
- In other word for cost optimaly $p = \Theta(f^{-1}(W))$

For cost optimal system

$$T_p = \Theta(W/p)$$

therefore represented by eqn

$$T_p^{cost-opt} = \Omega\left(\frac{W}{f^{-1}(W)}\right)$$

PAGE NO.:
DATE / /

⑧ Dense Matrix algorithms-

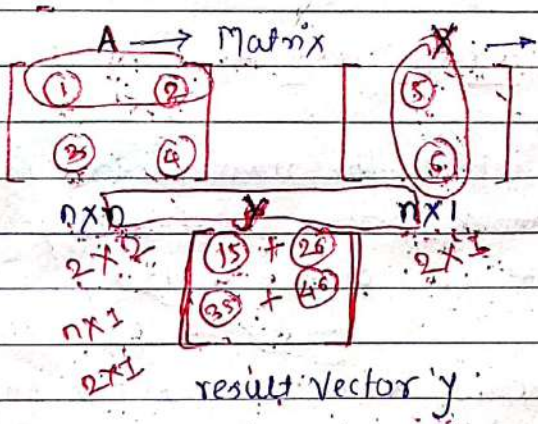
- Dense or full matrices that have no or few known usable zero entries, sparse matrix is a matrix in which most of the elements are zero.

- parallel computations involving matrices and vectors due to their regular structure, used for data decomposition.

- Typical algo rely on input, output and intermediate data decomposition.

- Most algo use one-dimensional (1D) and two dimensional (2D) block, cyclic & block cyclic partitionings.

A) Matrix-Vector Multiplication -




```
MV_ (A, X, Y)
{
for (I=0 to n-1)
{                    n
Y[i] = 0
for (j=0 to n-1)
{                    n²
Y[i] = Y[i] + A[i][j]
                     * V[j]
}
}
}
```

- Multiply a dense nxn matrix A and nx1 vector x to result of this multiplication vector 'y'

- compute
  $y = y + A * x$, where A is a dense matrix,
- the sequential algo requires n^2 multiplication and addition.

Assuming that multiplication and addition pair take unit time, the sequential run time is

$$W = n^2$$

- The different partitioning schemes such as a row-wise 1D, column wise-1D and 2D can be used and parallel formulation of matrix and vector multiplication can be performed.

- Row-wise 1D partitioning -

- The $n \times n$ matrix is partitioned among p processors, with each processor storing n/p complete row of the matrix.

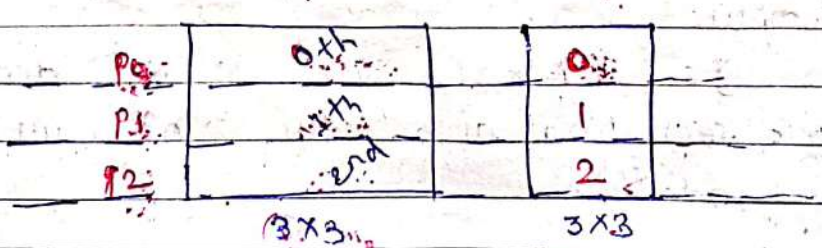
- $n \times 1$ vector x is distributed such that each process own n/p of its elements.

-A) One row per process -

- The case in which the $n \times n$ matrix is partitioned among n processes so that each process stores one complete row of the matrix

- The $n \times 1$ vector x is distributed such that process owns of its elements.

- The initial distribution of the matrix and the vector for row-wise block 1D-partitioning.



PAGE NO.:
DATE : / /

(A) (X)

one row per process

P0	0th		0
P1	1st		1
P2	2nd		2

- Here we have first row of matrix A to process P0 and also 0th element of vector x is given to process P0.
- In the same format, we have assigned 1st element of matrix A to process P1 and 1st element of vector to process P1 and so, on.
- process P0 initially own x[i] and A[i,0]---A[i,n-1] and is responsible for computing y[i].
- Vector x is multiplied with each row of the matrix, so every process needs the entire vector.
- Since, each process start with only element of x, an all to all broadcast is required to distribution all the elements to all the process.

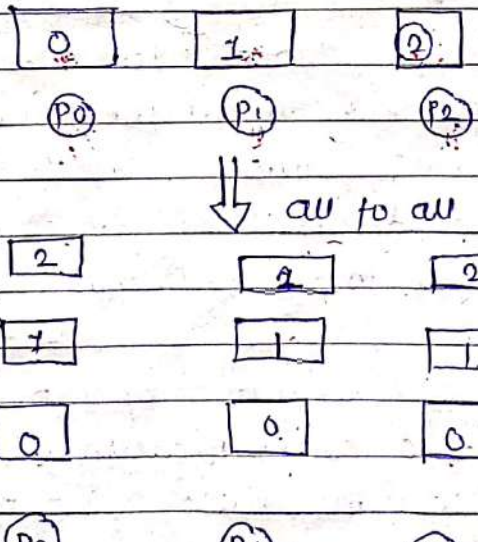
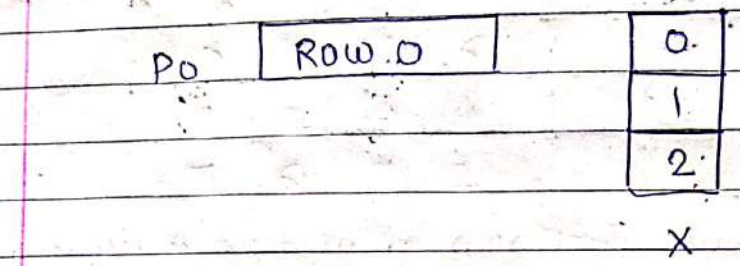
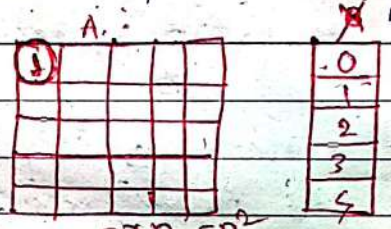


fig. after all to all broadcast

- so, every process will have row and vector element



- process time product - whatever the cost generated
- process product cost is $\Theta(n^2)$
- this time is same as sequential partitioning, so, we can conclude that 1D partitioning is cost optimal.

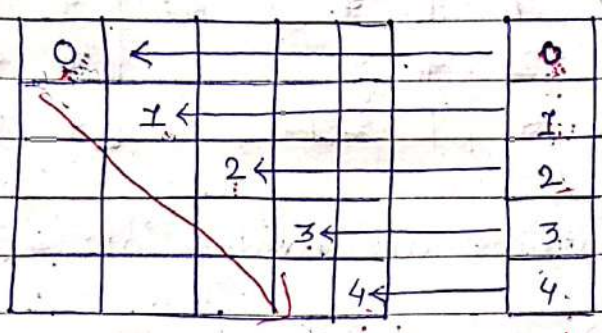


- 2D- partitioning -
- In 2D- partitioning - $p = n^2$ (one element per process)
- The $n \times 1$ vector x is distributed only in the last column of 'n' processors
- We must align the vector with matrix appropriately
- The first communication step for 2D partitioning aligns the vector x along the principal diagonal of the vector.
- The second step, the vector elements from each diagonal process to all the processes in the corresponding column.
- finally - the result vector is computed by performing an all to one reduction along the columns.

- three basic communication operations are used.
  1) one to one communication
  2) one to all broadcast
  3) All to one reduction.

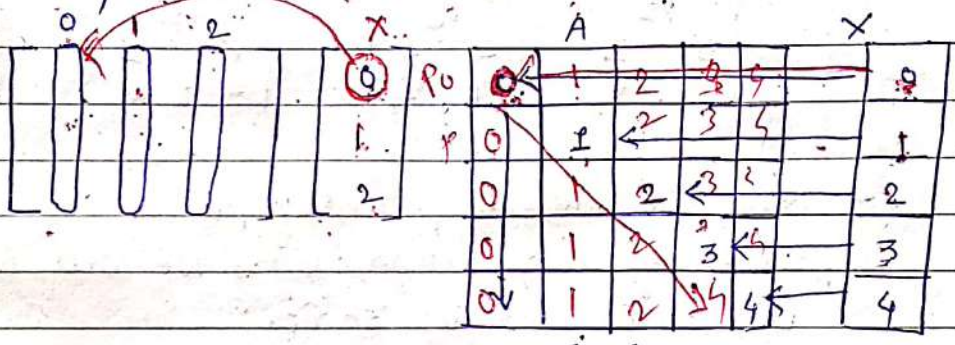
PAGE NO.:
DATE / /

1) one to one communication - It is use to align the vector along the main diagonal



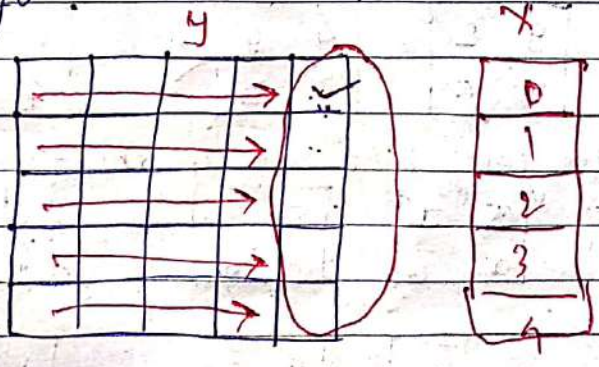
↳ After computation

2) one to all broadcast of each vector element among the n processes of each column.



3) All to one reduction -

- Whatever results are generated in a row by a particular process in a given element that will be reduced in the last row.
- perform the addition operation with all the last row elements and this will give us the resultant vector.
- cost complexity is $O(n^2 \log n)$. it is not a cost optimal algo. $\Theta(n^2)$



PAGE NO.
DATE / /

* Matrix - Matrix Multiplication -

$$A = \begin{bmatrix} 2 & 1 & 5 & 3 \\ 0 & 7 & 1 & 6 \\ 9 & 2 & 4 & 4 \\ 3 & 6 & 7 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 4 & 5 & 6 & 5 \\ 1 & 9 & +8 & -8 \\ 4 & 0 & -8 & 5 \end{bmatrix}$$

- These matrices are divided into 4 square block

P0		P1		P0		P1	
2	1	5	3	6	1	2	3
0	7	1	6	4	5	6	5

P2		P3		P2		P3	
9	2	4	4	1	9	8	-8
3	6	7	2	4	0	-8	5

- Now, we have 4 processors P0, P1, P2, P3 before doing any computation, first we will implement Left shift and up shift.

- Apply Left-shift on 1st matrix ie A and apply up shift on 2nd matrix ie B

- Left-shift - Apply row wise from bottom to up
- up shift - Apply column wise from right to left

P0		P1		P0		P1	
2	1	5	3	6	1	2	3
0	7	1	6	4	5	6	5

P2		P3		P2		P3	
9	2	4	4	1	9	8	-8
3	6	7	2	4	0	-8	5

After Left shift and up shift

P0		P1			P0		P1		
0	7	1	6	[16 7 28 35]	4	5	-8	5	[16 -25 -40 22]

P2		P3		P2		P3	
4	4	9	2	1	9	8	3
7	2	3	6	4	0	6	5

PAGE NO.:
DATE / /

- Multiplication should be performed with the same index processes. ie A·P0 x B·P0

$$C_0 = \begin{bmatrix} 16 & 7 \\ 28 & 35 \end{bmatrix} \quad \begin{bmatrix} 16 & -25 \\ -40 & 22 \end{bmatrix} = C_1$$

$$C_2 = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix} \quad \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix} = C_3$$

- We have to perform these shifting by
  $\sqrt{p}$ time (p number of processors)
  $\sqrt{4} = 2$
  so, we need to perform shifting operation two times

- Now, in 2nd iteration, As shifting is bottom to up from right to left from previous step:

P0	P1	P0	P1
5 3	2 1	1 9	8 -8
1 6	0 7	4 0	-8 5

P2	P3	P2	P3
4 4	9 2	6 1	2 3
7 2	3 6	4 5	6 5

~~So, after shift with P0 and P1 and up shift P~~

P0	P1	P0	P1
2 1	5 3	6 1	8 -8
0 7	1 6	4 5	-8 5

P2	P3	P2	P3
4 4	9 2	1 9	2 3
7 2	3 6	4 0	6 5

So after shift with P0 and P1, and up shift P3 x P1

P0	P1	P0	P1
5 3	2 1	1 9	2 3
1 6	0 7	4 0	6 5

P2	P3	P2	P3
9 2	4 4	6 1	8 -8
3 6	7 2	4 5	-8 5

$$C0 = \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix} \qquad C1 = \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix}$$

$$C2 = \begin{bmatrix} 62 & 19 \\ 42 & 33 \end{bmatrix} \qquad C3 = \begin{bmatrix} 0 & -12 \\ 40 & -46 \end{bmatrix}$$

then add current c value and previous c value

$$C0 = \begin{bmatrix} 16 & 7 \\ 28 & 35 \end{bmatrix} + \begin{bmatrix} 17 & 45 \\ 25 & 9 \end{bmatrix} = \begin{bmatrix} 33 & 52 \\ 53 & 44 \end{bmatrix}$$

$$C1 = \begin{bmatrix} 16 & -25 \\ -40 & 22 \end{bmatrix} + \begin{bmatrix} 10 & 11 \\ 42 & 35 \end{bmatrix} = \begin{bmatrix} 26 & -14 \\ 2 & 57 \end{bmatrix}$$

$$C2 = \begin{bmatrix} 20 & 36 \\ 15 & 63 \end{bmatrix} + \begin{bmatrix} 62 & 19 \\ 42 & 33 \end{bmatrix} = \begin{bmatrix} 82 & 55 \\ 57 & 96 \end{bmatrix}$$

$$C3 = \begin{bmatrix} 30 & 37 \\ 42 & 39 \end{bmatrix} + \begin{bmatrix} 0 & -12 \\ 40 & -46 \end{bmatrix} = \begin{bmatrix} 30 & 25 \\ 82 & -7 \end{bmatrix}$$

✱ Cannon's algorithm of matrix-matrix multiplication

- Cannon's algo is a memory efficient version of the simple algo for matrix-matrix multiplication.
- In cannon's algo, only shifting operation is used.
- The main advantage of the algo is that its storage requirement remain constant and are independent of the number of processors.
- 1 step

Initially the matrix A
Row 0 is unchanged
Row 1 is shifted 1 place left
Row 2 is shifted 2 place left
Row 3 is shifted 3 place left

initially the matrix B
column 0 is unchanged
column 1 is shifted 1 place up
column 2 is shifted 2 place up
column 3 is shifted 3 place up

eg

A					B					C			
A_{00}	A_{01}	A_{02}	A_{03}		B_{00}	B_{01}	B_{02}	B_{03}		C_{00}	C_{01}	C_{02}	C_{03}
A_{10}	A_{11}	A_{12}	A_{13}	×	B_{10}	B_{11}	B_{12}	B_{13}	=	C_{10}	C_{11}	C_{12}	C_{13}
A_{20}	A_{21}	A_{22}	A_{23}		B_{20}	B_{21}	B_{22}	B_{23}		C_{20}	C_{21}	C_{22}	C_{23}
A_{30}	A_{31}	A_{32}	A_{33}		B_{30}	B_{31}	B_{32}	B_{33}		C_{30}	C_{31}	C_{32}	C_{33}

- Now apply shifting operation
  - Row wise shifting with A matrix (Wrap around shifting)
  - column wise shifting with B matrix
  - Row wise shifting is in left direction and column wise shifting is in up direction.

-

00	01	02	03
11	12	13	10
22	23	20	21
33	30	31	32

00	11	22	33
10	21	32	03
20	31	02	13
30	01	12	23

A_{00}	A_{01}	A_{02}	A_{03}
B_{00}	B_{11}	B_{22}	B_{33}
A_{11}	A_{12}	A_{13}	A_{10}
B_{10}	B_{21}	B_{32}	B_{03}
A_{22}	A_{23}	A_{20}	A_{21}
B_{20}	B_{31}	B_{02}	B_{13}
A_{33}	A_{30}	A_{31}	A_{32}
B_{30}	B_{01}	B_{12}	B_{23}

— Now, one shift for each block (single step shifting)

A_{00}	A_{01}	A_{02}	A_{03}
B_{00}	B_{11}	B_{22}	B_{33}
A_{11}	A_{12}	A_{13}	A_{10}
B_{10}	B_{21}	B_{32}	B_{03}
A_{22}	A_{23}	A_{20}	A_{21}
B_{20}	B_{31}	B_{02}	B_{13}
A_{33}	A_{30}	A_{31}	A_{32}
B_{30}	B_{01}	B_{12}	B_{23}

→

01	02	03	00
10	21	32	03
12	13	10	11
20	31	02	13
23	20	21	22
30	01	12	23
30	31	32	33
00	11	22	33

Again apply one shift (row left, column-up)

02	03	00	01
20	31	02	13
13	10	11	12
30	01	12	23
20	21	22	23
00	11	22	33
31	32	33	30
10	21	32	03

→

03	00	01	02
30	01	12	23
10	11	12	13
00	11	22	33
21	22	23	20
10	21	32	03
32	33	30	31
20	31	02	13

apply (one shift)
row - left, column - up

$\sqrt{P} = \sqrt{16} = 04$

C =

A_{03}	A_{00}	A_{01}	A_{02}
B_{30}	B_{01}	B_{12}	B_{23}
A_{10}	A_{11}	A_{12}	A_{13}
B_{00}	B_{11}	B_{22}	B_{33}
A_{21}	A_{22}	A_{23}	A_{20}
B_{10}	B_{21}	B_{32}	B_{03}
A_{32}	A_{33}	A_{30}	A_{31}
B_{20}	B_{31}	B_{02}	B_{13}