Parallel Algorithm

RAM (Random Access Machine)

- **Unbounded** number of local memory cells
- Each memory cell can hold an integer of **unbounded** size
- Instruction set included –simple operations, data operations, comparator, branches
- All operations take **unit time**
- **Time complexity** = number of instructions executed
- **Space complexity** = number of memory cells used

PRAM (Parallel Random Access Machine)

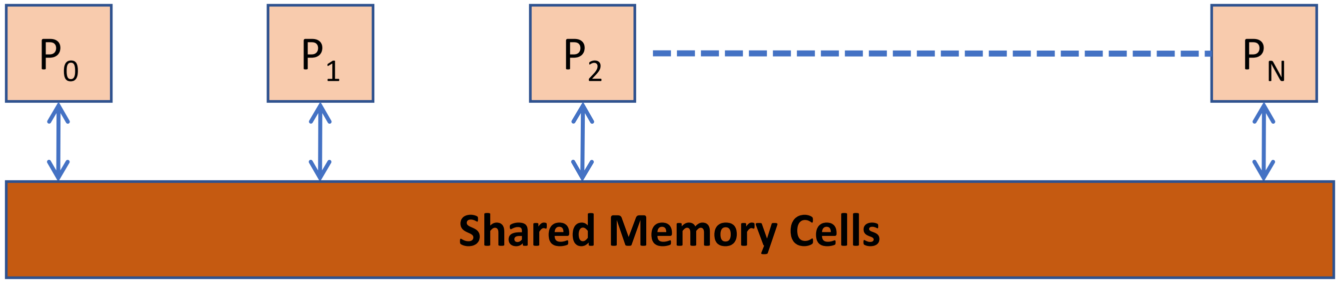
- Definition:
  - Is an abstract machine for designing the algorithms applicable to parallel computers
  - M’ is a system <M, X, Y, A> of infinitely many
    - RAM’s M1, M2, …, each M_i is called a processor of M’. All the processors are assumed to be identical. Each has ability to recognize its own index i
    - Input cells X(1), X(2),…,
    - Output cells Y(1), Y(2),…,
    - Shared memory cells A(1), A(2),…,

PRAM (Parallel RAM)

- Unbounded collection of RAM processors P_0, P_1, …,
- Each processor has unbounded registers
- Unbounded collection of share memory cells
- All processors can access all memory cells in unit time
- All communication via shared memory

PRAM (Parallel RAM)

- Some subset of the processors can remain idle



- Two or more processors may read simultaneously from the same cell
- A **write conflict** occurs when two or more processors try to write simultaneously into the same cell

Share Memory Access Conflicts

- PRAM are classified based on their Read/Write abilities (realistic and useful)
  - Exclusive Read(ER) : all processors can simultaneously read from distinct memory locations
  - Exclusive Write(EW) : all processors can simultaneously write to distinct memory locations
  - Concurrent Read(CR) : all processors can simultaneously read from any memory location
  - Concurrent Write(CW) : all processors can write to any memory location
  - EREW, CREW, CRCW

Concurrent Write (CW)

- What value gets written finally?
  - Priority CW: processors have priority based on which value is decided, the highest priority is allowed to complete WRITE
  - Common CW: all processors are allowed to complete WRITE ***iff*** all the values to be written are equal.
  - Arbitrary/Random CW: one randomly chosen processor is allowed to complete WRITE
  - Combimimg CW: a functiom may map multiple values into a single value. Function may be max, min, sum, multiply etc

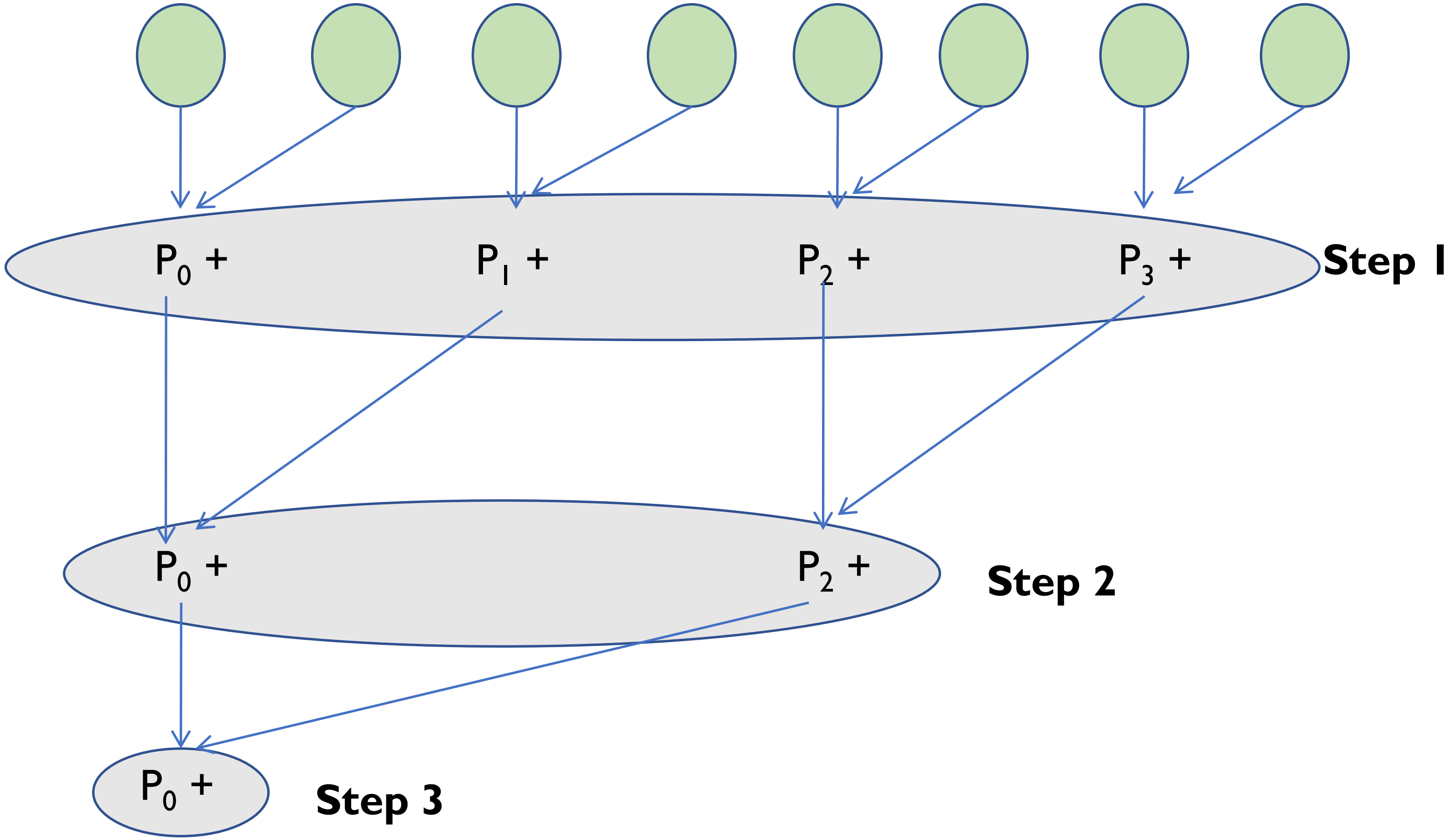
Strengths of PRAM

- PRAM is attractive and important model for designers of parallel algorithms Why?
  - It is **natural**: the number of operations executed per one cycle on p processors is at most p
  - It is **strong**: any processor can read/write any shared memory cell in unit time
  - It is **simple**: it abstracts from any communication or synchronization overhead, which makes the complexity and correctness of PRAM algorithm easier
  - It can be used as a **benchmark**: If a problem has no feasible/efficient solution on PRAM, it has no feasible/efficient solution for any parallel machine

An initial example

- How do you add N numbers residing in memory location A[0, 1, …, N]
- Serial Algorithm = O(N)
- PRAM Algorithm using N processors P_0, P_1, P_2, …, P_N ?

PRAM Algorithm (Parallel Addition)



- Program in P(i)
- L=n
- Repeat
- L=L/2
- If(i<L) then begin
- read A[2i] from SM
- Read A[2i+1] from SM
- Compute sum=A[2i]+A[2i+1]
- store in A[i]
- Until(L=1)

PRAM Algorithm (Parallel Addition)

- Log (n) steps = time needed
- n / 2 processors needed
- Speed-up = n / log(n)
- Efficiency = 1 / log(n)
- Applicable for other operations
  - +, *, <, >, etc.

Another algorithm to find sum

- Program in P(i)
- L=n
- Repeat
- L=L/2
- If(i<L) then begin
- read A[i] from SM
- Read A[i+L] from SM
- Compute sum of (A[i]and A[i+L])
- store in A[i]
- Until(L=1)

Program to find maximum of N nos

- Program in   P(i)
- L=n
- Repeat
- L=L/2
- If(i<L) then begin
- read A[i] from SM
- Read A[i+L] from SM
- Compute Max(A[i],A[i+L])
- store in A[i]
- Until(L=1)

Program to find minimum of N nos

- Program in   P(i)
- L=n
- Repeat
- L=L/2
- If(i<L) then begin
- read A[i] from SM
- Read A[i+L] from SM
- Compute Min(A[i],A[i+L])
- store in A[i]
- Until(L=1)

Program to find Product of N nos

- Program in P(i)
- L=n
- Repeat
- L=L/2
- If(i<L) then begin
- read A[i] from SM
- Read A[i+L] from SM
- Compute Product(A[i],A[i+L])
- store in A[i]
- Until(L=1)

Program to find sum of N nos using M processors

- Program in   P(I)
- sum=0

- For(J=I; J<N; J=J+M)
- Sum=sum+A[J]
- A[I]= sum
- L=M
- Repeat
- L=L/2
- If(i<L) then begin
-     read A[i] from SM
-     Read A[i+L] from SM
-     Compute sum(A[i],A[i+L])
-     store in A[i]
- Until(L=1)

Program to find sum of N nos using M processors-another way of writing

- Program in P(I)
- PAR for (I= 0;I< M,I++)
- {
- sum=0
-     For(J=I; J<N; J=J+M)
-         Sum=sum+A[J]
- A[I]= sum
- }
- L=M
- Repeat
- L=L/2
- If(i<L) then begin
-     read A[i] from SM
-     Read A[i+L] from SM
-     Compute sum(A[i],A[i+L])
-     store in A[i]
- Until(L=1)

Program to find sum of N nos using M processors-another way of writing

- Program in P(I)
- PAR for (I= 0;I< M,I++)
- {
- sum=0
- K= N/M
-                 For(J=K*I; J<(I+1)*K; J=J+1)
-                    Sum=sum+A[J]
- A[I]= sum
- }
- L=M
- Repeat
- L=L/2
- If(i<L) then begin
-         read A[i] from SM
-         Read A[i+L] from SM
-         Compute sum(A[i],A[i+L])
-         store in A[i]
- Until(L=1)

Matrix multiplication using n Processors on CRCW PRAM

- For (i=0;i<n; i++)
- {
- for (j=0;j<n; j++)
- { C[i][j]=0
- PAR for(k=0; k<n;k++)
- {
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]=A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

Matrix multiplication using n Processors on CREW PRAM

- For (i=0;i<n; i++)
- {
- for (j=0;j<n; j++)
- { PAR for(k=0; k<n;k++)
- {C[i][j]=0
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]=C[i][j]+A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

Matrix multiplication using nxn Processors on CRCW PRAM

- PAR For (i=0;i<n; i++)
- {
- PAR for (j=0;j<n; j++)
- { C[i][j]=0
- for(k=0; k<n;k++)
- {
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]=A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

Matrix multiplication using nxn Processors on CRCW PRAM

```
• For ( i=0;i<n; i++)
•       {
•            PAR for (j=0;j<n; j++)
•                    { PAR for(k=0; k<n;k++)
•                        {
•                          Read A[i][k]
•                          Read B[k][j]
•                          Compute C[i][j]=A[i][k]*B[k][j]
•                           store in C[i][j]
•                          }
•                  }
• }
```

Matrix multiplication using nxn Processors on CREW PRAM

- PAR For (i=0;i<n; i++)
- {
- PAR for (j=0;j<n; j++)
- { for(k=0; k<n;k++)
- {
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]= C[i][j]+A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

Matrix multiplication using nxnxn Processors on CRCW PRAM

- PAR For (i=0;i<n; i++)
- {
- PAR for (j=0;j<n; j++)
- { PAR for(k=0; k<n;k++)
- {
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]=A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

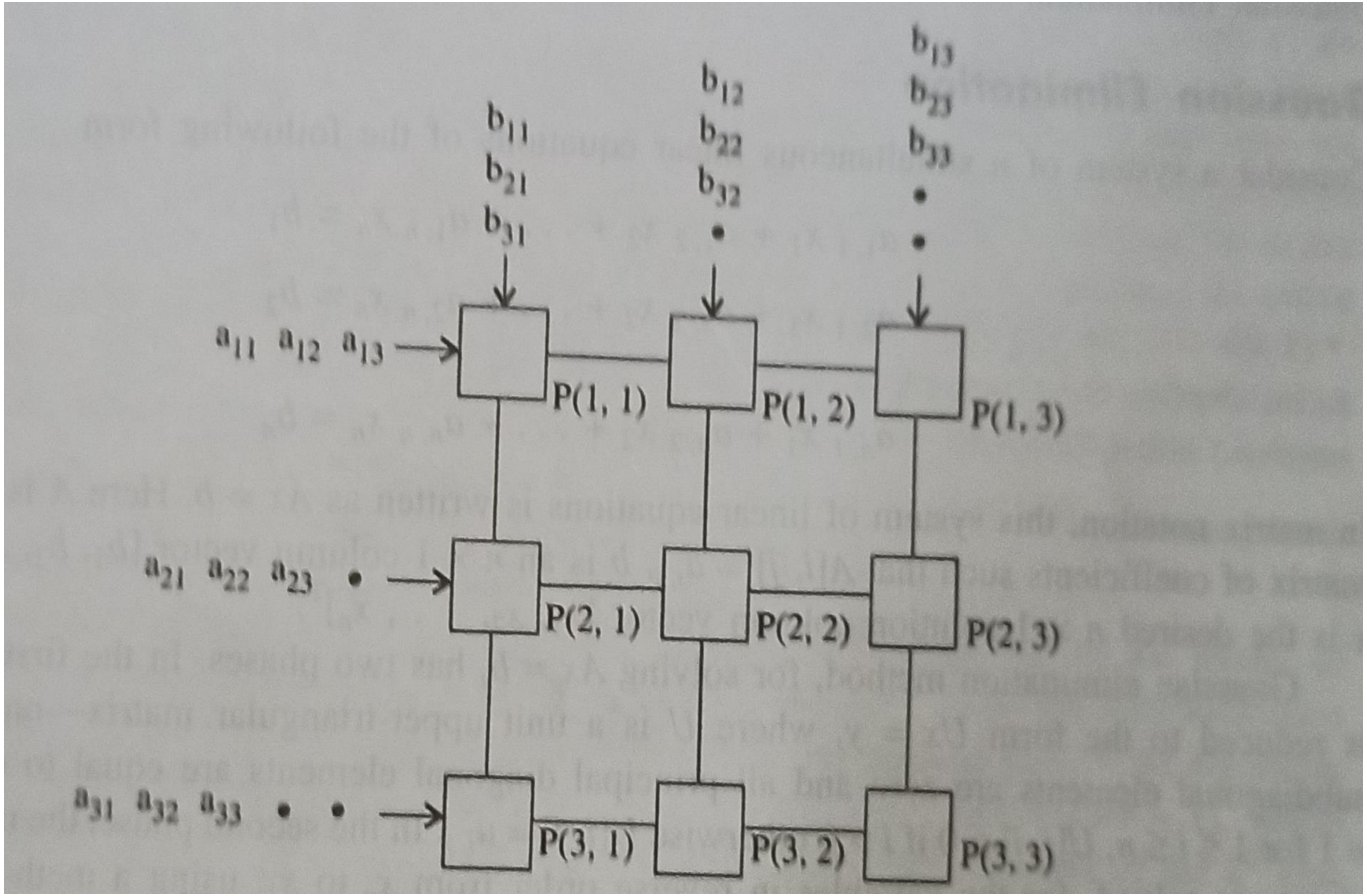
Matrix multiplication using nxnxn Processors on CREW PRAM

- PAR For (i=0;i<n; i++)
- {
- PAR for (j=0;j<n; j++)
- { PAR for(k=0; k<n;k++)
- {C[i][j]=0
- Read A[i][k]
- Read B[k][j]
- Compute C[i][j]=C[i][j]+A[i][k]*B[k][j]
- store in C[i][j]
- }
- }
- }

Matrix multiplication using nxn Processors on EREW PRAM

- PAR For (i=0;i<n; i++)
- {
- PAR for (j=0;j<n; j++)
- { C[i][j]=0
- for(k=0; k<n;k++)
- {
- Ik=(i+j+k)mod n + 1
- Read A[i][Ik]
- Read B[Ik][j]
- Compute C[i][j]=C[i][j]+A[IK][k]*B[Ik][j]
- store in C[i][j]
- }
- }
- }

Matrix Multiplication on Mesh with NxN processors



Parallel Algorithm

```
for i := 1 to n do in parallel
    for j := 1 to n do in parallel
        c_{i,j} := 0
        while P_{i,j} receives two inputs a and b do
            c_{i,j} := c_{i,j} + a * b
            if i < n then send b to P_{i+1, j}
            end if
            if j < n then send a to P_{i, j+1}
            end if
        end while
    end for
end for
```

Complexity

- Processor P(I,j) receives its input after i-1+j-1 steps from the beginning of computation
- After getting the input P(I,j) takes n steps to Compute C(I,j)
- So C(I,j) is computed in i-1+j-1 +n steps
- So complexity of algorithm is O(i-1+j-1 +n)=O(n-1+n-1+n)=O(n)

Matrix Multiplication on SIMD machines

For $j = 1$ **to** n **Do**

Par for $k = 1$ **to** n **Do**

$c_{ik} = 0$ (*vector load*)

For $j = 1$ **to** n **Do**

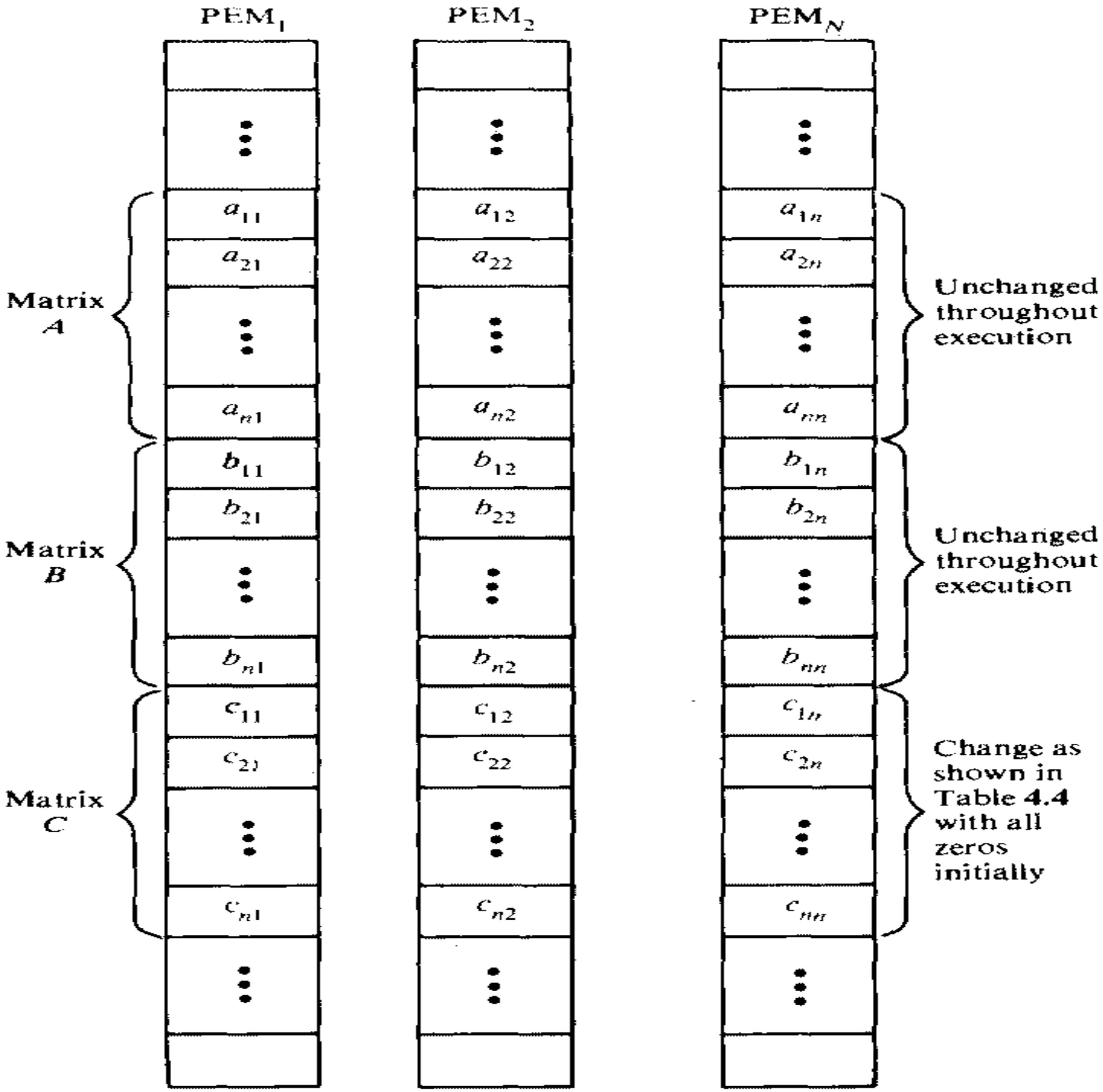
Par for $k = 1$ **to** n **Do**

$c_{ik} = c_{ik} + a_{ij} \cdot b_{jk}$ (*vector multiply*)

End of j loop

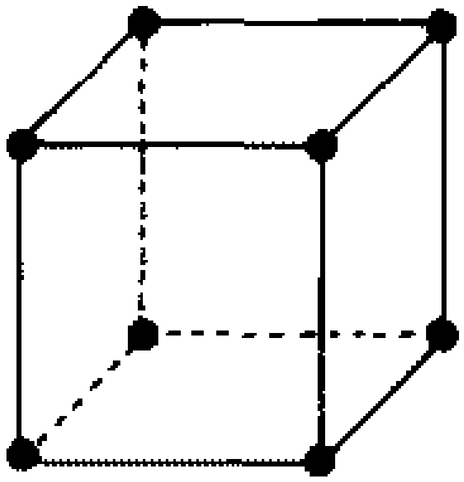
End of i loop

Memory allocation for matrix multiplication

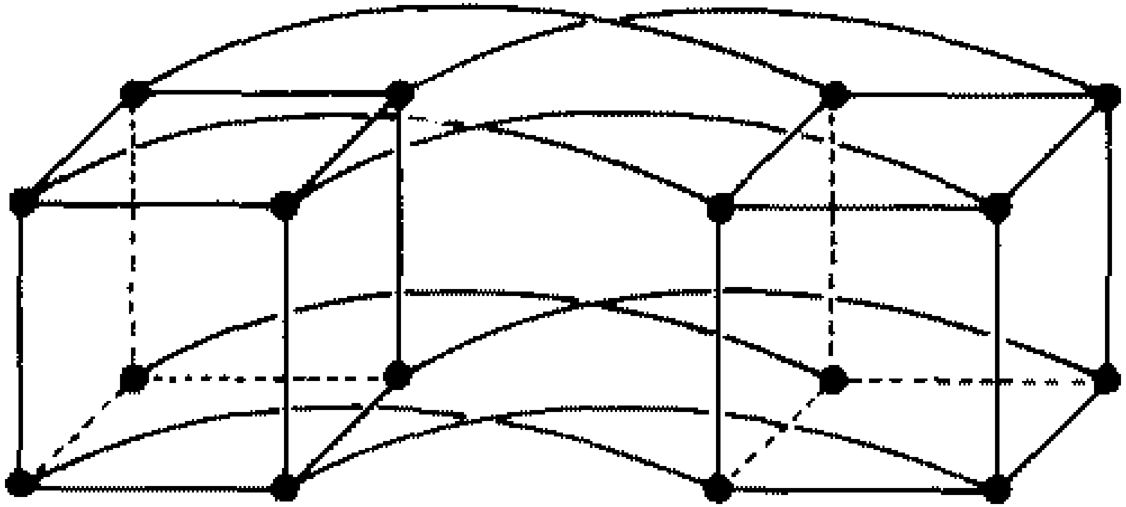


Successive content of C array in the memory

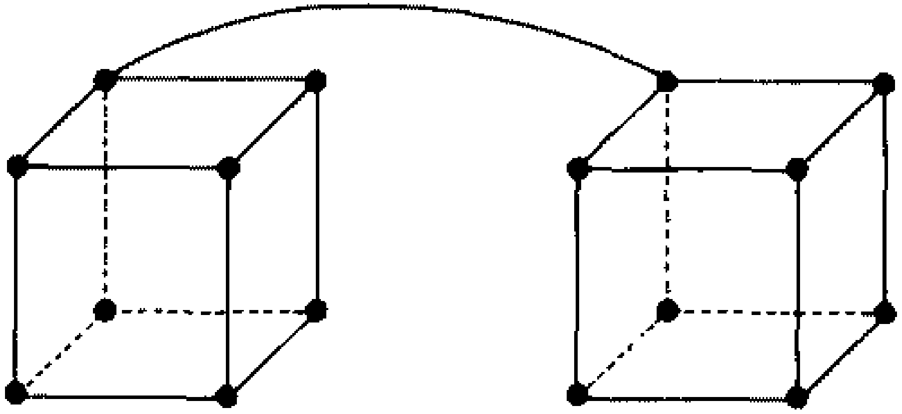
Outer loop i	Inner loop j	Parallel SIMD operations on $k = 1, 2, \ldots, n$			
		$c_{i1} \leftarrow c_{i1} + a_{ij} \times b_{j1}$	$c_{i2} \leftarrow c_{i2} + a_{ij} \times b_{j2}$	$\cdots$	$c_{in} \leftarrow c_{in} + a_{ij} \times b_{jn}$
1	1	$c_{11} \leftarrow c_{11} + a_{11} \times b_{11}$	$c_{12} \leftarrow c_{12} + a_{11} \times b_{12}$		$c_{1n} \leftarrow c_{1n} + a_{11} \times b_{1n}$
	2	$c_{11} \leftarrow c_{11} + a_{12} \times b_{21}$	$c_{12} \leftarrow c_{12} + a_{12} \times b_{22}$		$c_{1n} \leftarrow c_{2n} + a_{12} \times b_{2n}$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
	n	$c_{11} \leftarrow c_{11} + a_{1n} \times b_{n1}$	$c_{12} \leftarrow c_{12} + a_{1n} \times b_{n2}$		$c_{1n} \leftarrow c_{1n} + a_{1n} \times b_{nn}$
2	1	$c_{21} \leftarrow c_{21} + a_{21} \times b_{11}$	$c_{22} \leftarrow c_{22} + a_{21} \times b_{12}$		$c_{2n} \leftarrow c_{2n} + a_{21} \times b_{1n}$
	2	$c_{21} \leftarrow c_{21} + a_{22} \times b_{21}$	$c_{22} \leftarrow c_{22} + a_{22} \times b_{22}$		$c_{2n} \leftarrow c_{2n} + a_{22} \times b_{2n}$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
	n	$c_{21} \leftarrow c_{21} + a_{2n} \times b_{n1}$	$c_{22} \leftarrow c_{22} + a_{2n} \times b_{n2}$		$c_{2n} \leftarrow c_{2n} + a_{2n} \times b_{2n}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	1	$c_{n1} \leftarrow c_{n1} + a_{n1} \times b_{11}$	$c_{n2} \leftarrow c_{n2} + a_{n1} \times b_{12}$		$c_{nn} \leftarrow c_{nn} + a_{n1} \times b_{1n}$
	2	$c_{n1} \leftarrow c_{n1} + a_{n2} \times b_{21}$	$c_{n2} \leftarrow c_{n2} + a_{n2} \times b_{22}$		$c_{nn} \leftarrow c_{nn} + a_{n2} \times b_{2n}$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
	n	$c_{nn} \leftarrow c_{nn} + a_{nn} \times b_{n1}$	$c_{n2} \leftarrow c_{n2} + a_{nn} \times b_{n2}$		$c_{nn} \leftarrow c_{nn} + a_{nn} \times b_{nn}$
Local memory		PEM_1	PEM_2	$\cdots$	PEM_n



(a) A 3 cube

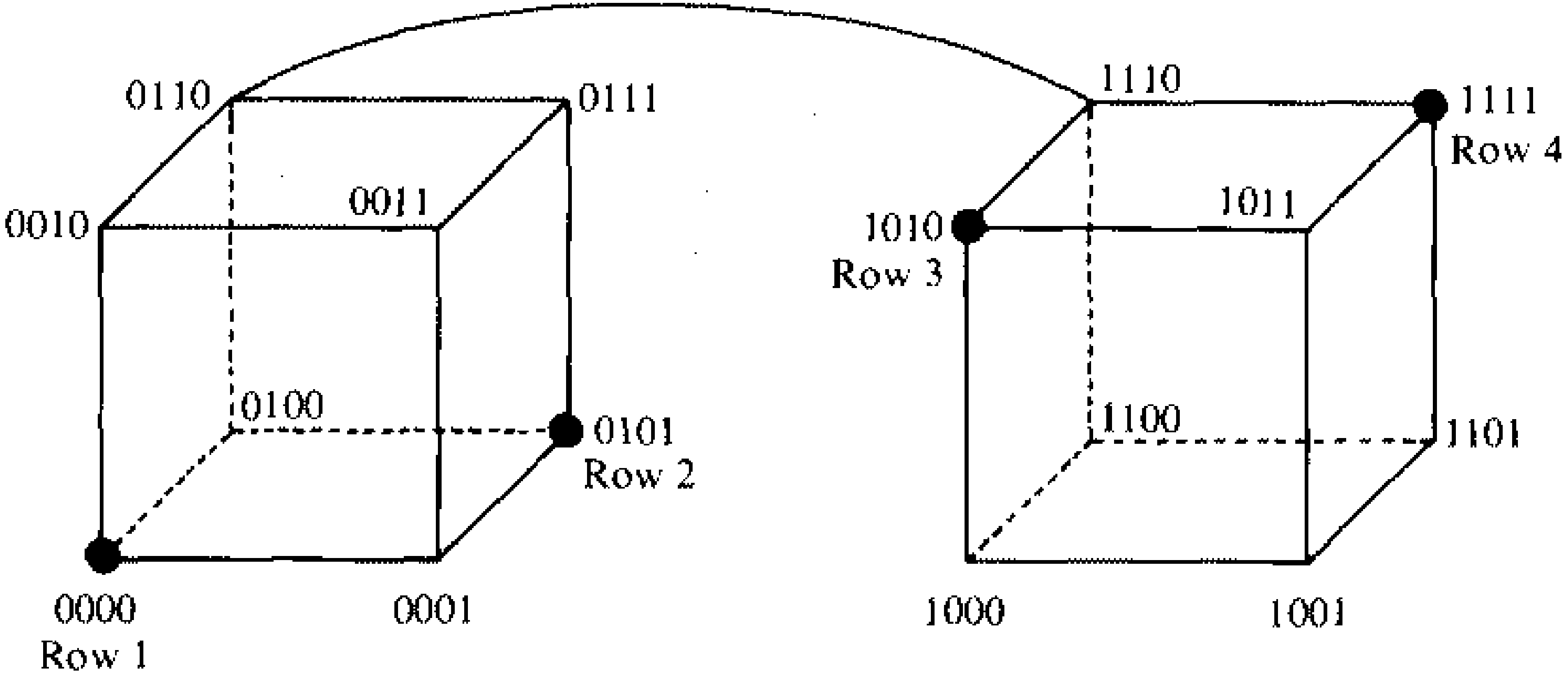


(b) A 4 cube formed from two 3 cubes



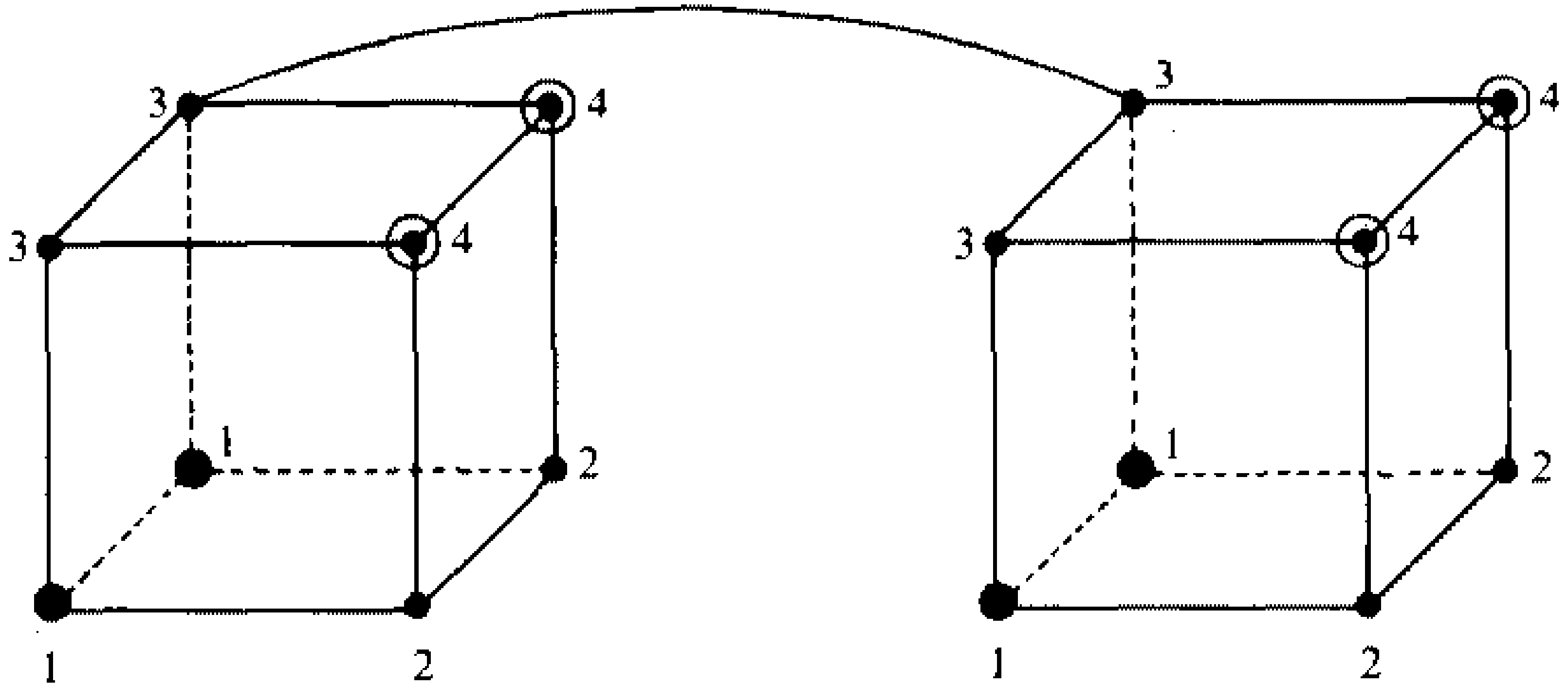
(c) The 4 cube showing only one of eight fourth-dimension connections

Initial distribution of rows of A



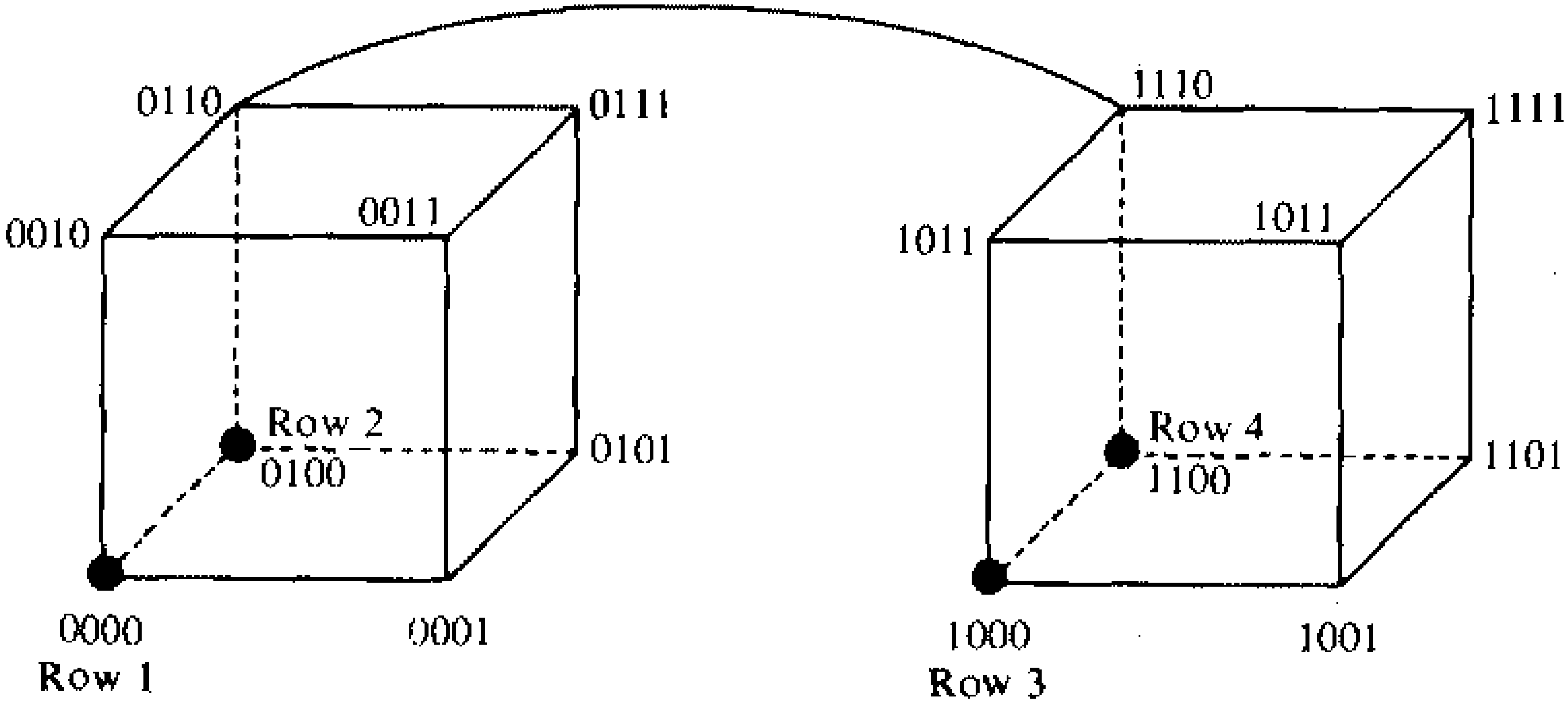
(*a*) Initial distribution of rows of *A*

4-way broadcaste of rows of A



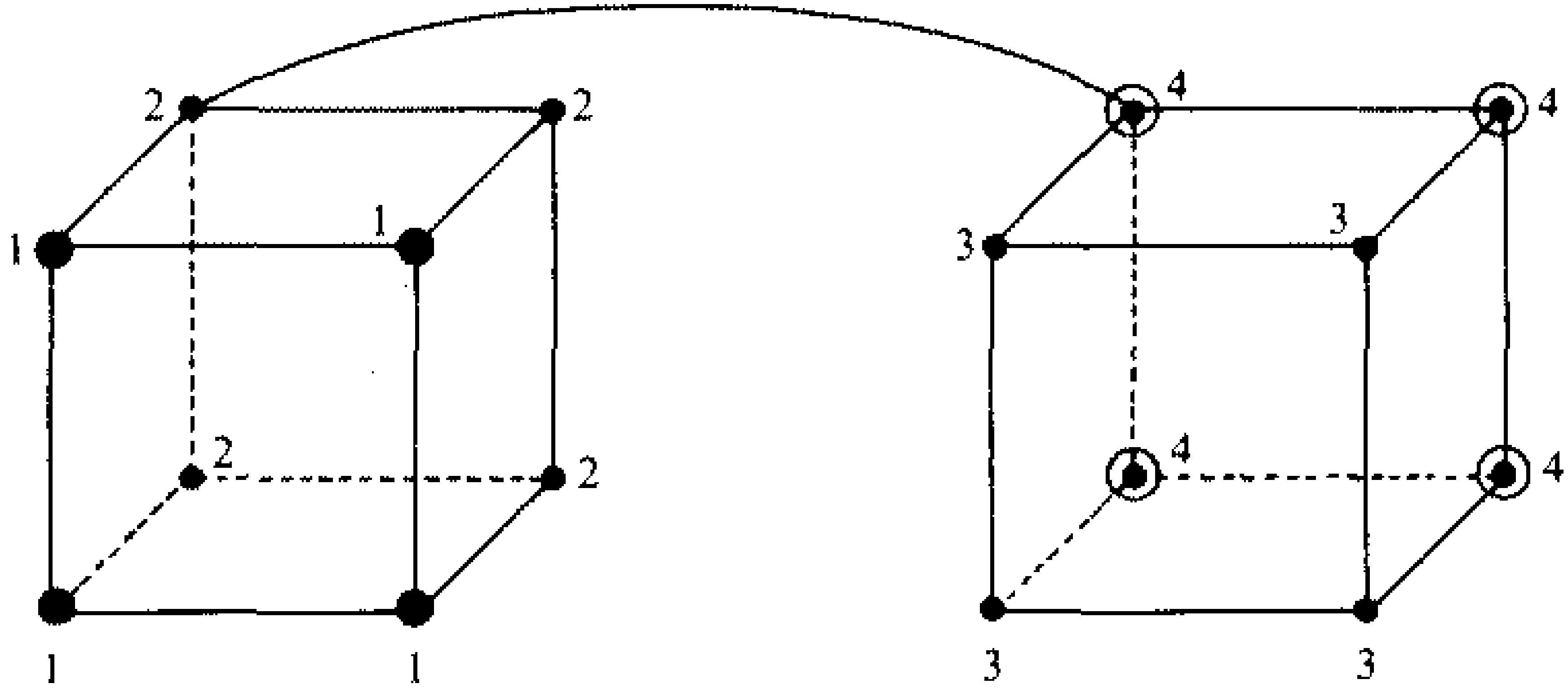
(b) **4-way** broadcase of rows of A

Initial Distribution of rows of B transpose



(c) Initial distribution of rows of B^t

4-way broadcast of rows of B transpose



(*d*) 4-way broadcase of rows of B^t

Let $(p_{2m-1}p_{2m-2}\cdots p_m p_{m-1}\cdots p_1 p_0)_2)$ be the PE address in the $2m$ cube. We can achieve the $O(n \log_2 n)$ compute time only if initially the matrix elements are favorably distributed in the PE vertices. The n rows of matrix A are distributed over n distinct PEs whose addresses satisfy the condition

$$p_{2m-1}p_{2m-1}\cdots p_m = p_{m-1}p_{m-2}\cdots p_0 \tag{5.25}$$

as demonstrated in Figure 5.20a for the initial distribution of four rows of the matrix A in a 4×4 matrix multiplication ($n = 4$, $m = 2$). The four rows of A are then broadcast over the fourth dimension and front to back edges, as marked by row numbers in Figure 5.20b.

Example 5.5: An $O(n \log_2 n)$ algorithm for matrix multiplication

1. Transpose B to form B^t over the m cubes $x_{2m-1} \cdots x_m 0 \cdots 0$ in $n \log_2 n$ steps (Figure 5.20c).
2. N-way broadcast each row of B^t to all PEs in the m cube

$$p_{2m-1} \cdots p_m x_{m-1} \cdots x_0$$

   in $n \log_2 n$ steps (Figure 5.20d).
3. N-way broadcast each row of A residing in PE $p_{2m-1} \cdots p_m p_{m-1} \cdots p_0$ to all PEs in the m cube $x_{2m-1} \cdots x_m p_{n-1} \cdots p_0$ in $n \log_2 n$ steps (Figure 5.20b). All the n rows can be broadcast in parallel.
4. Each PE now contains a row of A and a column of B and can form the inner product in $O(n)$ steps (Figure 5.21). The n elements of each result row can be brought together within the same PEs which initially held a row of A in $O(n)$ steps.

CREW Matrix multiplication

Procedure CREW Matrix Multiplication

```
for i:=1 to n do in parallel
    for j:=1 to n do in parallel
        c_{i,j} :=0;
        for k:=1 to n do
            c_{i,j} := c_{i,j} + a_{i,k} * b_{k,j};
        end for
    end for
end for
```

EREW Matrix multiplication

Procedure EREW Matrix Multiplication

```
for i:=1 to n do in parallel
    for j: = 1 to n do in parallel
        c_{i,j} :=0;
        for k := 1 to n do
            l_k := (i + j + k) mod n + 1;
            c_{i,j} := c_{i,j} + a_{i,l_k} * b_{l_k,j};
        end for
    end for
end for
```

CRCW Matrix multiplication

Procedure CRCW Matrix Multiplication

```
for i := 1 to n do in parallel
    for j := 1 to n do in parallel
        for s := 1 to n do in parallel
            c_{i,j} := 0
            c_{i,j} := a_{i,s} * b_{s,j}
        end for
    end for
end for
```