Your Roll No.....

CEN-704

B.Tech.(Computer Engineering), VII Semester Examination 2018

Parallel and Distributed Computing

Paper No: CEN-704

Time:-03 Hours

Max. Marks:-60

Write your Roll No. on top immediately on receipt of question paper.

Attempt all questions by attempting any two parts from each question. All questions carry equal marks.

CO1

Q1(a)
Three enhancements with the following speed ups are proposed for a new architecture: Speedup1=30 , speedup2= 20 , and peedup3 =15. Only one enhancement is usable at a time. Assume the enhancements can be used 25% , 35% and 10% of the time for enhancements 1 2 and 3 respectively. For what fraction of the reduced execution time is no enhancement in use? If only two enhancements are to be used then which two enhancements should be used? 06

Q1(b)
(i).An application program is executed on a nine computer cluster. A benchmark program takes time T on this cluster. Further 25% of T is time in which the application is run simultaneously on all nine computers. The remaining time, the application has to run on a single computer. Calculate the effective speedup under the aforementioned condition as compared to executing the program on a single computer. Also calculate the percentage of code that has been parallelized in the preceding program. 03

(ii) Let a be the percentagee of program code that can be executed simultaneously by n computers in a cluster, each computer using a different set of parameters or initial conditions. Assume that the remaining code must be executed simultaneously by a single processor. Each processor has an execution rate of x MIPS. Determine an expression for the effective MIPS rate when using the system for exclusive execution of this program in terms of a, n and x. 03

Q1
(c). Consider a computer which can execute a program in two operational modes; regular mode versus enhanced mode with a probability distribution of {A, 1-A} respectively. If A varies between a and b and 0<=a<=1, derive an expression for the average speedup factor using harmonic mean concept. 06

CO3

CO2

Q2(a) Describe Branch prediction techniques with help of suitable examples and diagrams. 06

Q2(b)(i) Consider the following

N= no. of instructions to be executed
M= No. of segments in pipeline
P= probability that a given instruction is a unconditional branch instruction
Q= probability that a given instruction is a conditional branch instruction
R= probability that a given conditional branch instruction will cause branching.

Calculate the followings

i. Speedup
ii. Throughput
iii. efficiency
iv. Average no. of Instructions executed per Instruction cycle. 04

Q2(b)

(ii)Derive an expression for optimal number of stages in a pipeline. 02

Q2(c) Three functional pipelines f1, f2 and f3 are characterized by the following reservation tables. 06

f1:

	1	2	3	4
S1	X			
S2		X		
S3			X	X

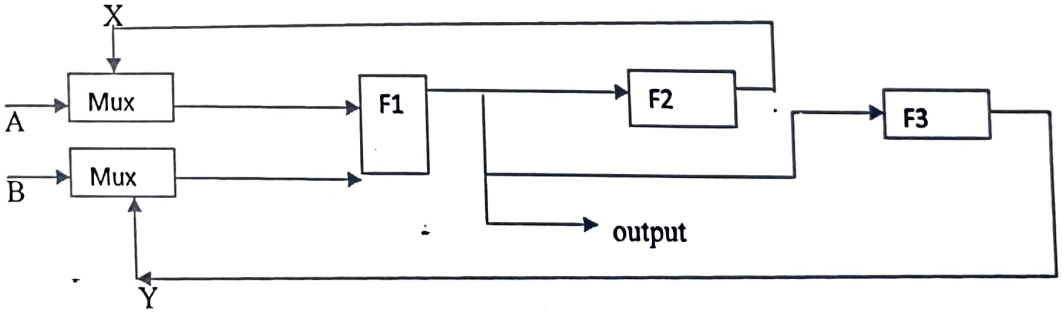
f2:

	1	2	3	4
T1	X			X
T2		X		
T3			X	

f3:

	1	2	3	4
U1	X		X	
U2				X
U3		X		

Using these three pipelines a composite pipeline is formed as below.



Each task going through this composite pipeline in the following order: f1 first, f2 and f3 next, f1 again and then the output is obtained. The dual multiplexer selects a pair of inputs (A,B) or (X,Y) and feeds them into the input of f1.

(i) Draw the state transition diagram for collision free scheduling
(ii)List all simple cycle and greedy cycle
(iii)Find MAL
reservation table.

CO3

Q3 (a) Design and verify SIMD algorithm for NXN Matrix multiplication with computational complexity as $O(N^2)$. 06

Q3 (b) What is PRAM model? Design and verify PRAM Algorithm for EREW NXN matrix multiplication with computational complexity of O(N). 06

Q3(c) Parallelize Quick Sort Algorithm. Compute its speed up with respect to its sequential algorithm. 06

CO4

Q4(a). What is MPI program? Write an MPI program for computing the values of N! 06

Q4(b) What do you mean by Open Mp program ? Write an open MP program to compute the value of PI 06

Q4(c) Explain the following terms (i) GPU (ii) CUDA. Explain how a CUDA program is written with help of suitable example. Revise 06

CO5

Q5. Describe in details any two of the following with the help of suitable examples and diagrams.

(a)Mobile Agent

(b) Object request Broker

(c) Matrix Logical clock 2x6=12

NAME: Shubham Gupta

PROGRAM NAME: B. Tech. (Computer Engineering)

SEMESTER: 7th SEM.

EXAMINATION ROLL NO.: 17BCS017

UNIQUE PAPER CODE: CEN-704

PAPER TITLE: PARALLEL & DISTRIBUTION COMPUTING

DATE OF EXAM: 9th FEBRUARY 2020

TIME OF EXAM: 10:00AM To 1:00PM

Shubham Gupta
17BCS017

Q-1] - According to question,

i). Iteration :

0 : $L_2 + L_4$

1 : $L_2 + 2L_4$

2 : $L_2 + 3L_4$

⋮ ⋮

2047 : $L_2 + 2048L_4$

Total : $2048L_2 + (1+2+3+4 \ldots\ldots 2048)\, L_4$

$= 2048L_2 + 2098176L_4$

Since L_2 and L_4 take 2 machine cycles.

Total time on uniprocessor system =

$(2048 + 2098176) * 2$

$= \underline{4200448}$ machine cycles.

ii) Processor :

1 : $I = 1-64 \Rightarrow (L_2+L_4) + (L_2+2L_4) + \ldots\ldots (L_2+64L_4)$

2 : $I = 65-128 \Rightarrow (L_2+65L_4) + (L_2+66L_4) + \ldots\ldots (L_2+128L_4)$

⋮ ⋮ ⋮

32 : $I = 1985-2048 \Rightarrow (L_2+1985L_4) + (L_2+1986L_4) + \ldots\ldots (L_2+2048L_4)$

→ Processors 33-64 aren't used.

→ Since loop goes till 2048 and each processor handles 64 iterations. Hence 32 processor * 64 iterations = 2048. So only 32 processors are used.

→ The overall execution time is determined by processor 32.

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

$64L_2 + (1985L_4 + 1986L_4 \ldots\ldots 2048L_4)$
$\Rightarrow 64L_2 + 129056L_4$
Total duration = $(64 + 129056) * 2 = 258240$ machine cycles
Speed up = $4200448/258240 = 16.2656$.

iii) In order to get a balanced load, the i loops have to be distributed such that each processor executes equal machine cycles

Processor 1: i = 1 ..... 16 and i = 2033 to 2048
Processor 2: i = 17 ..... 32 and i = 2017 to 2032
⋮ ⋮ ⋮ ⋮ ⋮
Processor 64: i = 1009 to 1024 and i = 1025 to 1040.

If each processor is given iterations from beginning as well as from last, balancing may be done.
~~Divide the iteration in size/chunk size of~~
The modified program is:-

```
PAR for (L=1, L<=64, L++)
{   for (I=(L-1)*32+1, I<=L*32, I++)
    {  SUM[I]=0
       for(J=1, J<=I, J++)
        SUM[I] = SUM[I]+I
    }
for (I=(64-L)*32+1, I<=(64-L+1)*32, I++)
        {
          SUM[I]=0
            for(J=1, J<=I, J++)
              SUM[I]=SUM[I]+I
        }
}
```

Shubham Gupta
17BCS017

DATE: __/__/__
PAGE

iv) Total Execution time = $32L_2 + 32784L_4$

$= (32 + 32784) * 2$

$= 65,632$ machine cycles

$$\text{Speedup} = \frac{4200448}{65632} = 64$$

(the load is perfectly balanced over all 64 processors)

NAME: Shubham Gupta

PROGRAM NAME: B. Tech. (Computer Engineering)

SEMESTER: 7th SEM.

EXAMINATION ROLL NO.: 17BCS017

UNIQUE PAPER CODE: CEN-764

PAPER TITLE: PARALLEL & DISTRIBUTION COMPUTING

DATE OF EXAM: 9th FEBRUARY 2020

TIME OF EXAM: 10:00AM To 1:00PM

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

Q-2] - a) Branch prediction is used to overcome the fetch limitation imposed by control hazards in order to expose instruction level parallelism (ILP), the key ingredient to pipelined and superscalar architectures that mask instruction execution latencies.

→ Performance = {(accuracy, cost of misprediction)}

→ Branch arrive much faster when multiple instructions are issued per clock.

→ Use Amdahl's law.

→ Methods:

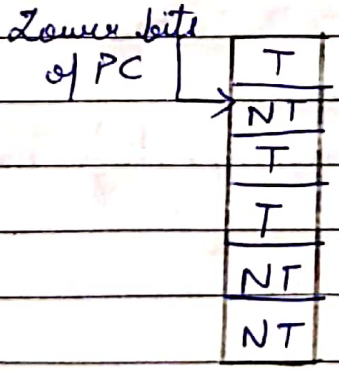
- Branch history table (1 or more bits)
- Correlating branches
- Branch target buffer

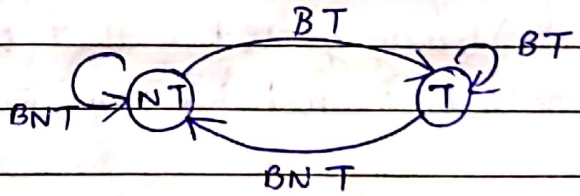
1) 1 Bit Branch Predictor

→ Lower bits of PC address index table of 1-bit values.

→ Entry says whether or not branch taken last time

→ No address check.





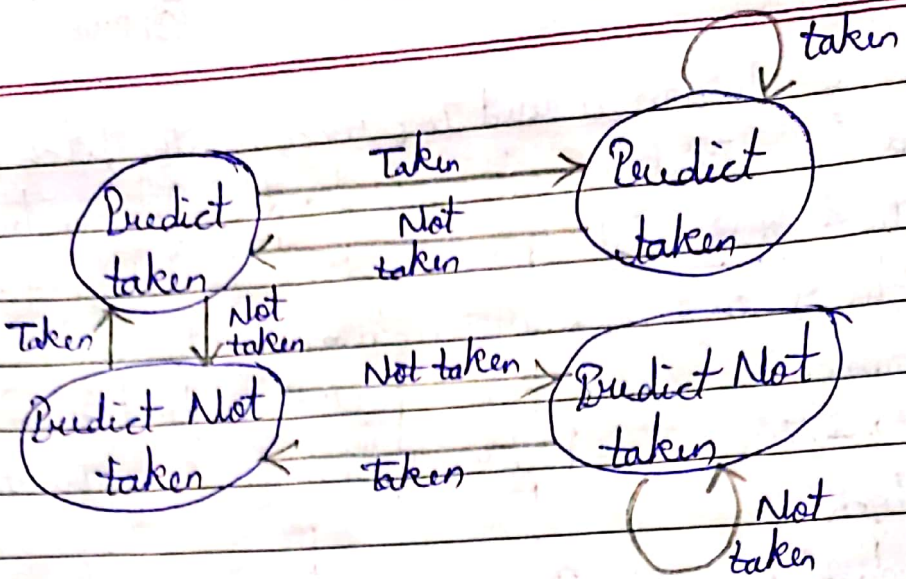
→ Problem: In a loop 1-bit BHT will cause at least two mispredictions

- First time through loop on next time through code, when it predicts exit instead of looping
- End of loop case, when it exits instead of looping as before.

2) 2-Bit Branch Predictor

Shubham Gupta
17BCS017

DATE: __/__/
PAGE



→ When we get misprediction twice, the prediction changes.

2-bit predictor:
→ counter can hold values b/w 0 & $2^n - 1$.
→ predict taken when values >= $2^n - 1$ half of new value.
→ The counter is incremented on each taken branch and decremented on each not taken branch.

Misprediction:
→ Wrong guess for that branch.
→ Got branch history of wrong branch from index table.

3) Correlated Branch Prediction:

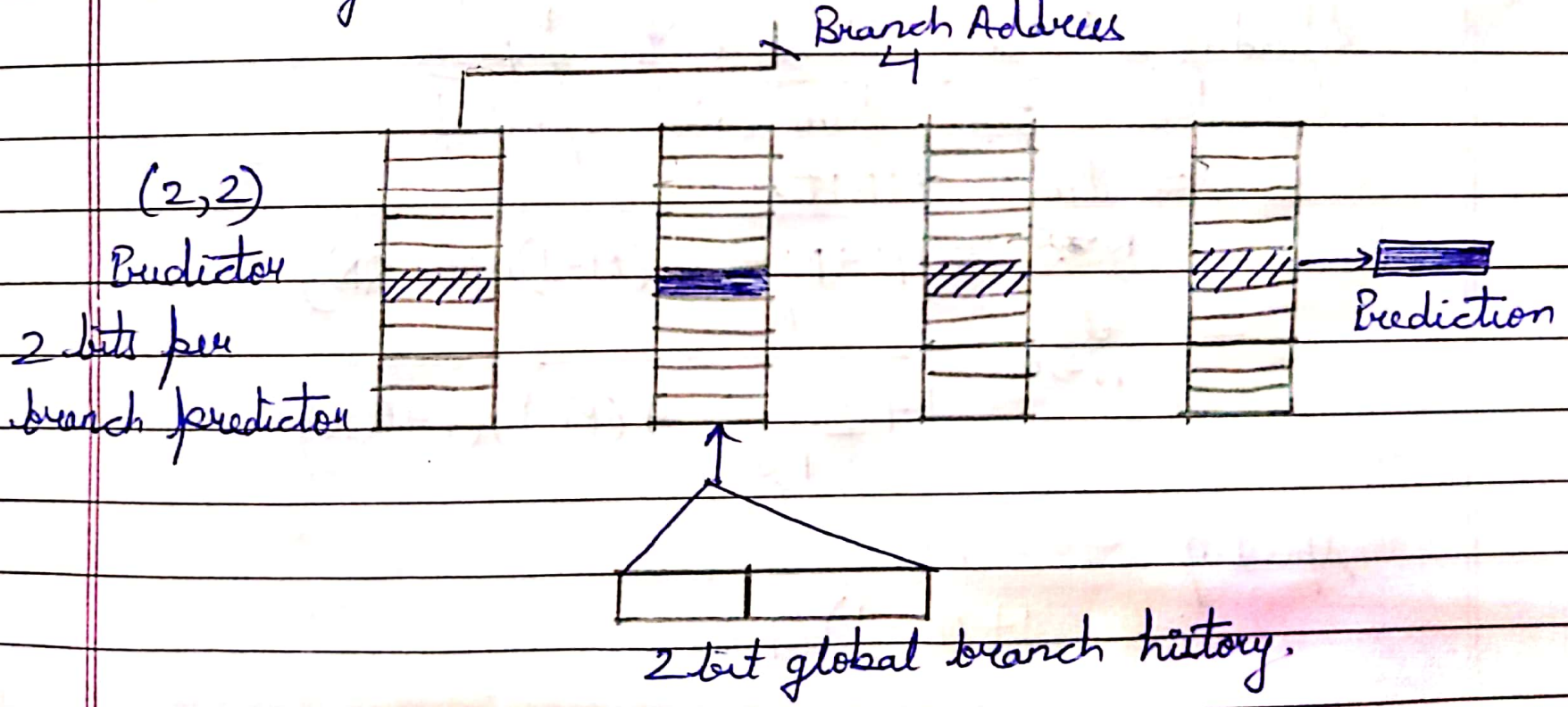
Record in most recently executed branches as taken as not taken and use that pattern to select the proper n-bit branch history table.

Global branch history – m-bit shift register keeping T/NT status of last m branches.

Shubham Gupta
17BCS017

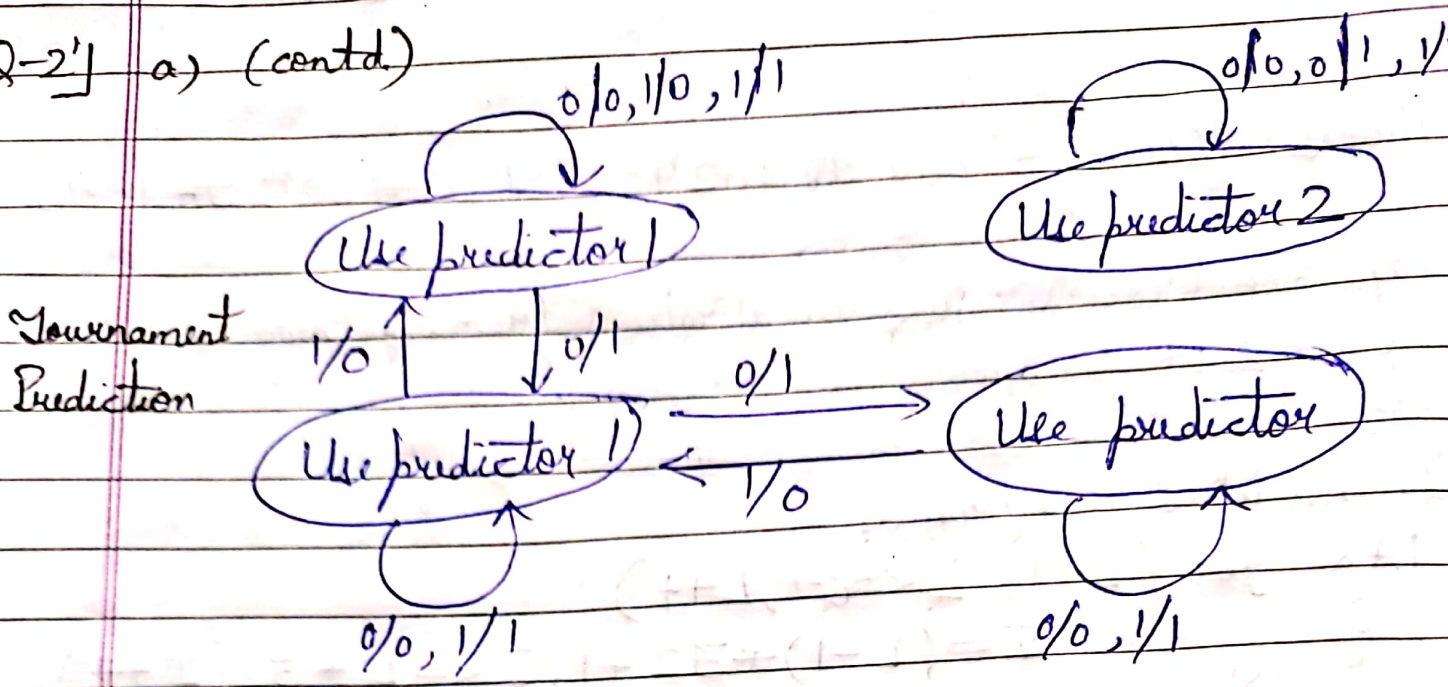
DATE: __/__/____
PAGE _____

Correlating Branch



4) Tournament Branch Predictor:
It is multilevel predictor. 1 based on local information and other on local and they are combined with a selector.

Q-2] a) (contd.)



Branch prediction schemes are of 2 types:

i) Static B.P. 2) Dynamic B.P.

↓

These techniques are very simple to analyse & require low cost and energy to execute instructions because it don't require any history table of instruction as well as h/w components.

⇒ Main scheme of static branch prediction:-

Single direction prediction	Backward taken forward not taken	program based prediction	profile based pred.

GOOD WRITE

b) According to question,

No. of Instructions = N

No. of segments in pipeline = M

Probability of unconditional branch = P

Probability of conditional branch = Q

Probability that branch is taken = R

i) No. of unconditional branches = NP

No. of conditional branches = NQ

No. of condl. branches taken = NQR

Total branches instruction = NP + NQR

= N(P + QR)

Total time = (M + N − 1) × t + extra branch cycles

= (M + N − 1) t + (M − 1) × N (P + QR)

= t [M + N − 1 + (M − 1) × N × (P + Q,R)]

GOOD WRITE

(Shubham Gupta, 17BCS017)

Shubham Gupta
17BCS017

DATE: / PAGE

Total time w/o pipelining = MNt

$$\text{Speedup} = \frac{\text{Exec. time with pipeline}}{\text{Exec. time with speedup}}$$

$$= \lim_{N\to\infty} \frac{MNt}{t\left[M+N-1+(M-1)N(P+QR)\right]}$$

$$= \lim_{N\to\infty} \frac{M}{1+\frac{(N-1)}{N}+(M-1)(P+QR)}$$

$$\text{Speedup} = \frac{M}{1+(M-1)(P+QR)}$$

ii) $$\text{Throughput} = \frac{\text{No. of instructions}}{\text{Total time}}$$

$$= \lim_{N\to\infty} \frac{N}{(M+N-1+N(M-1)(P+QR))t}$$

$$= \frac{1}{(1+(M-1)(P+QR))t}$$

$$= \frac{f}{1+(M-1)(P+QR)}$$

where f is the frequency of the clock

iii) $$\text{Efficiency} = \frac{\text{Speedup}}{\text{No. of stages}}$$

$$= \frac{M}{1+(M-1)(P+QR)} \times \frac{1}{M}$$

$$= \frac{1}{1+(M-1)(P+QR)}$$

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

iv) Average Instructions executed per cycle = Throughput X Clock time

$$= \frac{f}{1+(M-1)(P+QR)} \times t$$

$$= \frac{1}{1+(M-1)(P+QR)}.$$

NAME: Shubham Gupta

PROGRAM NAME: B. Tech. (Computer Engineering)

SEMESTER: 7th SEM.

EXAMINATION ROLL NO.: 17BCS017

UNIQUE PAPER CODE: CEN-704

PAPER TITLE: PARALLEL & DISTRIBUTION COMPUTING

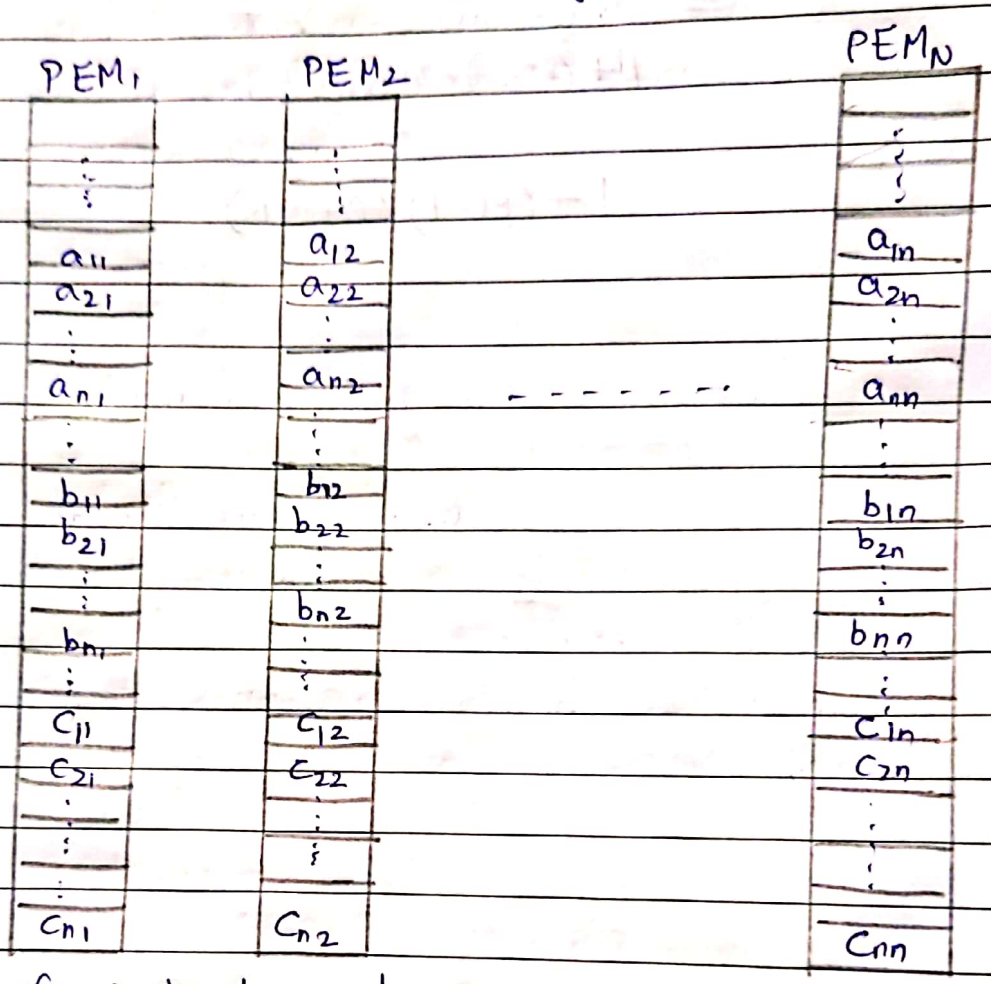
DATE OF EXAM: 9th FEBRUARY 2020

TIME OF EXAM: 10:00AM To 1:00PM

Shubham Gupta
17BCS017

DATE: __/__/__
PAGE

Q-3.1 – a) Matrix Multiplication on SIMD vector/Array processor in $O(n^2)$ time complexity (∵ n processors)




```
for i=1 to n do
    for k=1 to n do in parallel
        c[i,k] = 0
    end for
    for j=1 to n do
        for k=1 to n do in parallel
            c[i,k] = c[i,k] + a[i,j] * b[j,k]
        end for
    end for
end for
```

Time Complexity: $O(n^2)$.

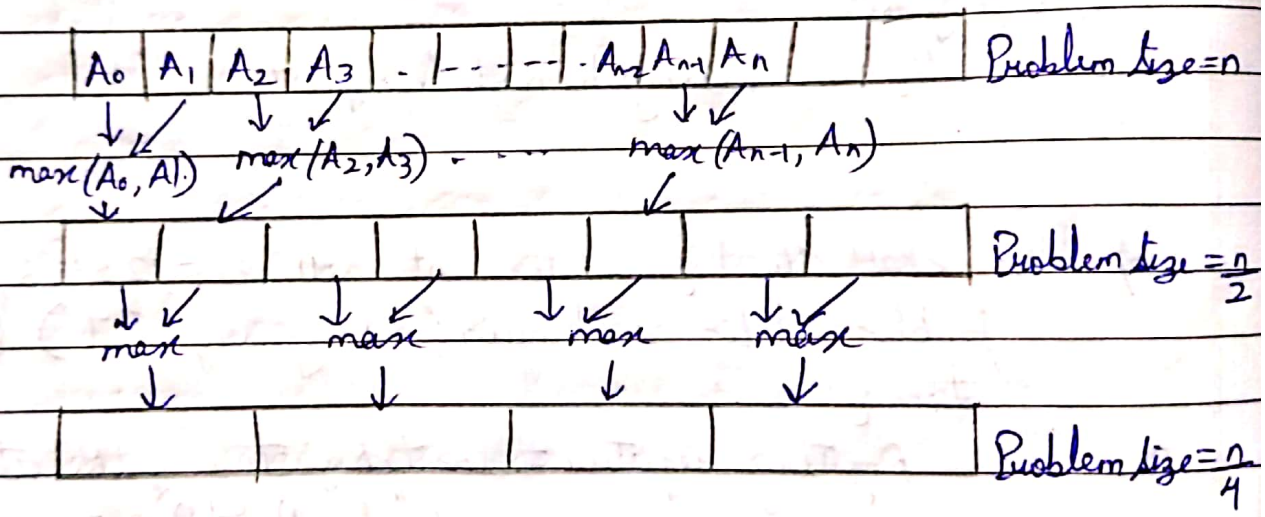
Shubham Gupta
17BCS017

DATE: __/__/__
PAGE _____

Q-3. | b) PRAM (Parallel Random Access Machine)
→ abstract machine for designing the algorithms applicable to parallel computers.

→ M' is a system $<M, X, Y, A>$ of infinitely many RAMs $M_1, M_2 \ldots$ 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) ....



Since the problem size is decreasing in half every clock cycle. The time taken is determined by height of the above process = log n.
Hence time complexity = O(log n)

Algo:
Program in P(i)
L = n
Repeat

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

if (i<L) then begin
read A[i] from shared memory
Read A[i+1] from shared memory
Compute Max(A[i], A[i+L])
Store in A[i][2]
i = i+2
L = L/2
Do until (L=1)

NAME: Shubham Gupta

PROGRAM NAME: B. Tech. (Computer Engineering)

SEMESTER: 7th SEM.

EXAMINATION ROLL NO.: 17BCS017

UNIQUE PAPER CODE: CEN-704

PAPER TITLE: PARALLEL & DISTRIBUTION COMPUTING

DATE OF EXAM: 9th FEBRUARY 2020

TIME OF EXAM: 10:00AM To 1:00PM

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

Q-4.)- a) Message Passing Interface (MPI) is a communication protocol for parallel programming. MPI is specifically used to allow applications to run in parallel across a number of separate computers connected by a network. It is full featured and designed to provide access to advanced parallel hardware for end users, tool developers, library writers etc.

MPI program for computing the factorial of an integer N:-

```
#include<stdio.h>
#include<mpi.h>
int main(int argc, char **argv)
{
   int myid, numprocs, i, n, lm, j, mod;
   int fact, rslt=1;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   int unused attribute((unused));
   if (myid==0)
   { printf("Enter the number to find the factorial\n");
     unused = scanf("%d", &n);
   }
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
   lm = n/numprocs;
   mod = n%numprocs;
   for(i=myid*lm+1; i<=myid*lm+lm; i++)
     {
      rslt = rslt*i;
      if(mod!=0)
      {
```

GOOD WRITE

```
if (myid == numprocs-1)
{
for(j=i; j<=(myid+1)*lmtmod; j++)
{
  rult = rult * j;
printf("The multiplication from %d is %d\n", myid+1, rult);
}
}
}
MPI_Reduce(&rslt, &fact, 1, MPI_INT, MPI_PROD, 0, MPI_COMM_
if (myid==0)                                        WORLD);
{ printf("\n The factorial of the Given number %d is %d", n, fact);
}
MPI_Finalize();
return 0;
}
```

On execution based on available processor, the number will be divided among different processors and will carry a local multiplication at different processors and the products are accumulated at master node using MPI_Reduce.

c) CUDA:→ It is a parallel computing platform and an API model that was developed by Nvidia. Using CUDA, one can utilize the power of Nvidia GPUs to perform general computing tasks, such as multiplying matrices and performing other linear algebra operations, instead of just doing graphical calculations. Using CUDA, developers can now harness the potential of the GPU for general purpose computing (GPGPU). It is Compute Unified Device Architecture.

Shubham Gupta
17BCS017

DATE: __/__/__
PAGE

CUDA C program for matrix multiplication :-

```
#include<stdio.h>
#include<math.h>
#define TILE_WIDTH 2
/* matrix multiplication kernels */
//non shared
__global__ void
MatrixMul(float *Md, float *Nd, float *Pd, const int
          WIDTH)
{  //calculate thread id
   unsigned int col = TILE_WIDTH * blockIdx.x +
                      threadIdx.x;
   unsigned int row = TILE_WIDTH * blockIdx.y +
                      threadIdx.y;
   for(int k=0; k<WIDTH; k++)
   { Pd[row*WIDTH+col] += Md[row*WIDTH+k]
                        * Nd[k*WIDTH+col];
   }
}
//shared
__global__ void
MatrixMulSh(float *Md, float *Nd, float *Pd, const
            int WIDTH)
{
// taking shared array to break the matrix in Tile Width
   and fetch them in that array per element
__shared__ float Mds[TILE_WIDTH]
 [TILE_WIDTH];
__shared__ float Nds[TILE_WIDTH]
 [TILE_WIDTH];
```

Shubham Gupta
17BCS017

SPP DATE: __/__/____
PAGE _____

```
//calculate thread id
  unsigned int col =
TILE_WIDTH * blockIdx.x + threadIdx.x;
  unsigned int row = TILE_WIDTH * blockIdx.y + threadIdx.y;
for(int m=0; m<WIDTH/TILE_WIDTH; m++)
      //m indicate number of phase
{ Mds[threadIdx.y][threadIdx.x] =
   Md[row*WIDTH + (m*TILE_WIDTH + threadIdx.x)];
  Nds[threadIdx.y][threadIdx.y] =
  Nd[(m*TILE_WIDTH + threadIdx.y)*WIDTH + col];
   __syncthreads(); // for synchronizing the threads

//Do for tile
 for(int k=0; k<TILE_WIDTH; k++)
  Pd[row*WIDTH + col] += Mds[threadIdx.x][K] * Nds[K]
                                          [threadIdx.y];

 __syncthreads();
 }
}

//main routine
int main()
{ const int WIDTH=6;
  float array1_h[WIDTH][WIDTH], array2_h[WIDTH][WIDTH],
     result_array_h[WIDTH][WIDTH], M_result_array_h[WIDTH]
                                          [WIDTH];
  float *array1_d, *array2_d, *result_array_d, *M_result
                                          _array_d;
  //device array
  int i,j;
//input is host array
  for(i=0; i<WIDTH; i++)
     {
```

```
    for (j=0; j<WIDTH; j++)
    { array1_h[i][j] = 1;
      array2_h[i][j] = 2;
    }
}
//copy host array to device array;

cudaMemcpy(array1_d, array1_h, WIDTH*WIDTH*sizeof(int),
           cudaMemcpyHostToDevice);
cudaMemcpy(array2_d, array2_h, WIDTH*WIDTH*sizeof(int)
           ,cudaMemcpyHostToDevice);

//allocating memory for resultant device array
cudaMalloc((void**)&result_array_d, WIDTH*
           WIDTH*sizeof(int));
cudaMalloc((void**)&M_result_array_d, WIDTH*
           WIDTH*sizeof(int));

//calling kernel
dim3 dimGrid(WIDTH/TILE_WIDTH, WIDTH/TILE_WIDTH, 1);
dim3 dimBlock(TILE_WIDTH, TILE_WIDTH, 1);

// Change if 0 to if 1 for running non shared code and make
   if 0 for shared memory code
#if 0
MatrixMul<<<dimGrid, dimBlock>>>(array1_d,
   array2_d, M_result_array_d, WIDTH);
#endif
#if 1
MatrixMulSh<<<dimGrid, dimBlock>>>(array1_d,
   array2_d, M_result_array_d, WIDTH);
```

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

```
#endif
// copy back result_array_d to result_array_h
cudaMemcpy(M_result_array_h, M_result_array_d, WIDTH*WIDTH
           * sizeof(int), cudaMemcpyDeviceToHost);
// print the result array
for(i=0; i<WIDTH; i++)
  { for(j=0; j<WIDTH; j++)
    { printf("%d", M_result_array_h[i][j]);
    }
  printf("\n");
}
system("pause");
}
```

NAME: Shubham Gupta

PROGRAM NAME: B. Tech. (Computer Engineering)

SEMESTER: 7th SEM.

EXAMINATION ROLL NO.: 17BCS017

UNIQUE PAPER CODE: CEN-704

PAPER TITLE: PARALLEL & DISTRIBUTION COMPUTING

DATE OF EXAM: 9th FEBRUARY 2020

TIME OF EXAM: 10:00AM To 1:00PM

Shubham Gupta
1713CS017

DATE: __/__/____
PAGE _____

Q-5] a) Mobile Agent

A mobile agent is a transportable program or object. In this paradigm, an agent is launched from an originating host. The agent then travels autonomously from host to host according to an itinerary that it carries. At each stop, the agent accesses the necessary resources or services and performs the necessary task to accomplish its mission.

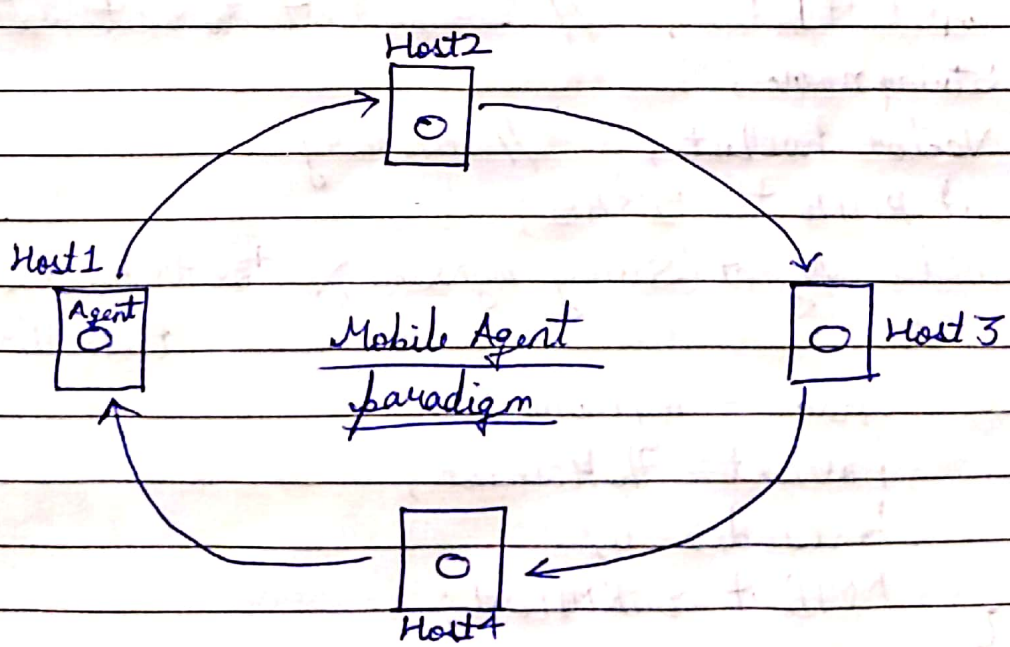
Agent carries following data items:

→ Identifying information: Information that allows the agent to be identified.

→ Itinerary: A list of addresses of the host that the agent is to visit.

→ Logic code to form its task.

Examples: Mobile Agent systems include Concordia system and Aglet system.



Program for Mobile Agent:-

→ AgentInterface.java

Shubham Gupta
17BCS017

DATE: __/__/__
PAGE

```
// An interface for a transportable object representing
// a mobile agent
import java.io.Serializable;
public interface AgentInterface extends Serializable {
  void execute();
}
```

→ Agent.java

```
// An implementation of a mobile agent
import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
public class Agent implements AgentInterface {
  int hostIndex; // which host to visit next
  String name;
  Vector hostList;  // itinerary
  int RMIPort = 12345;
  public Agent(String myName, Vector theHostList, int
                                        theRMIPort){
    name = myName;
    hostList = theHostList;
    hostIndex = 0;
    RMIPort = theRMIPort;
  }

// This method defines the tasks that the mobile agent is to
// perform once it has arrived at a server
```

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

```
public void execute(){
    String thisHost, nextHost;
    Sleep(2);
    System.out.println("Mobile Agent is here");
    thisHost = (String) hostList.elementAt(hostIndex);
    hostIndex++;
    if (hostIndex < hostList.size()) {
        // if there is another host to visit
        nextHost = (String) hostList.elementAt(hostIndex);
        sleep(5);
        try {
            // Locate the RMI registry on the next host
        Registry registry = LocateRegistry.getRegistry("localhost",
                                                        RMIPort);
ServerInterface h = (ServerInterface) registry.lookup(nextHost);
        System.out.println("Lookup for" + nextHost + "at" +
                           thisHost + "completed");
        }
        catch(Exception e) {
        system.out.println("Exception in Agent execute"+e);
            }
    }
    else {
        sleep(5);
        system.out.println("Agent has come home");
        sleep(5);
    }
}
```

→ ServerInterface.java

Shubham Gupta
17BCS017

DATE: __/__/__
PAGE

```
// Agent server interface file
import java.rmi.*;
public interface ServerInterface extends Remote {
    public void receive(Agent h)
        throws java.rmi.RemoteException;
}
```

→ Server.java

```
// An implementation of an agent server
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.Registry;
import java.rmi.registry.locateRegistry;
import java.net.*;
import java.io.*;
public class Server extends UnicastRemoteObject
    implements ServerInterface{
        static Int RMIPORT=12345;
    public Server() throws RemoteException {
            super();
    }
public void receive(Agent h) throws RemoteException{
    Sleep(3)
    system.out.println("** Agent" + h.name + "arrive");
    h.execute();
}
public static void main(String args[]){
    InputStreamReader is=new InputStreamReader
                            (system.in);
    BufferedReader br = new BufferedReader(is);
    String s;
```

Shubham Gupta
17BCS017

DATE: __/__/____
PAGE _____

```
String myName = "Server" + args[0];
try{
    System.setSecurityManager(new RMISecurityManager());
    Server h = new Server();
Registry registry = locateRegistry.getRegistry(RMIPort);
registry.rebind(myName, h);
System.out.println("Agent" + my name + "ready");
    }
catch (RemoteException re) {
    System.out.println("Exception in Agent server" + te);
        }
```

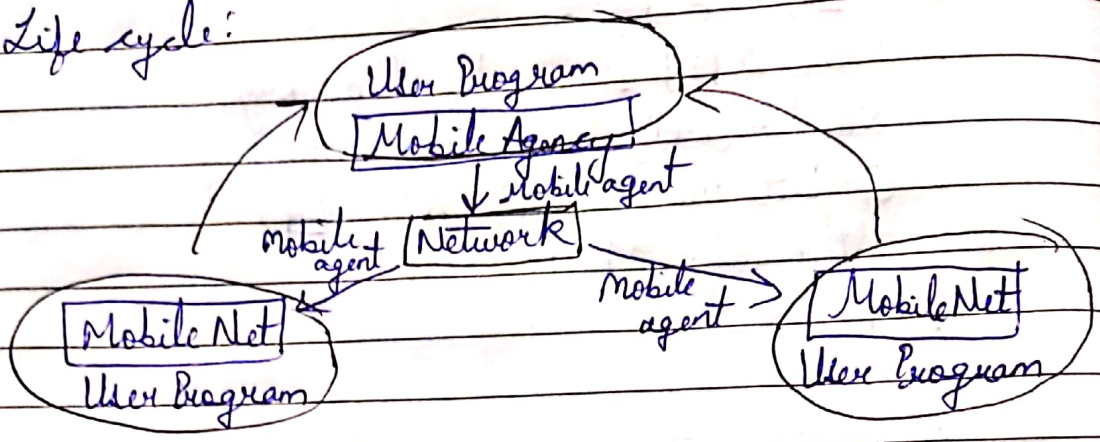
→ Client.java

```
    // for launching the mobile agent
    .
    .
    .
public class Client{ static int RMIPort = 12345;
    public static void main(String args[]) {
        try {
            Registry registry = locateRegistry.getRegistry
                                ("localhost", RMIPort);
        ServerInterface h = (ServerInterface) registry.lookup("Server");
    s.o.p("lookup for server1 completed");
    vector hotlist = new Vector();
    hotlist.addElement("server1");
    hotlist.addElement("server2");
        "       "       "   ("server3");
    Agent a = new Agent("007", hostList, RMIPort);
    h.receive(a);
}
```

Shubham Gupta
17BCS017

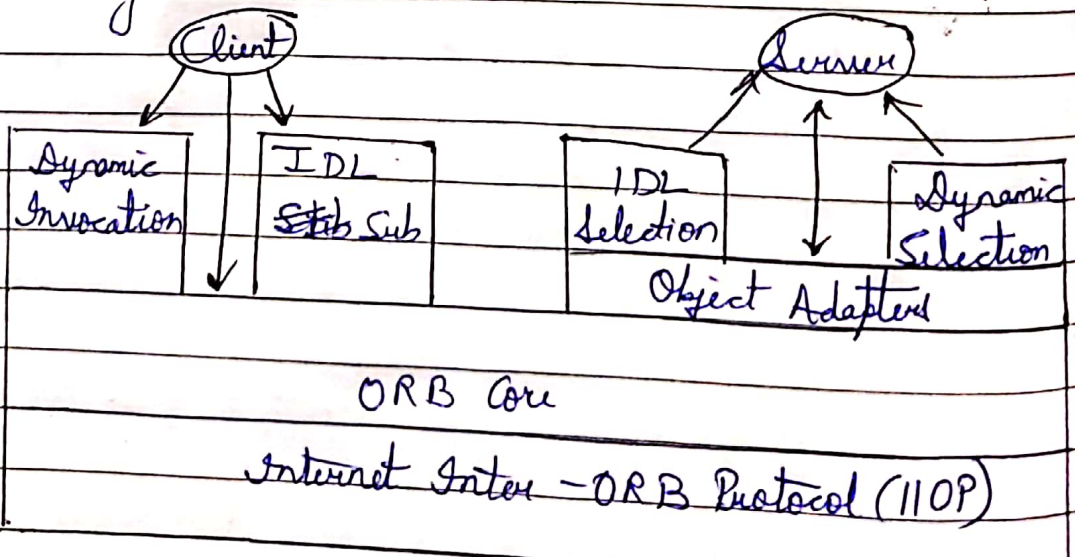
DATE: __/__/__
PAGE

Life cycle:



b) Object Request Broker:

It is the programming that acts as a "broker" in between a client request for a service from a distributed object or component and the completion of that request. Having ORB support in a network means that a client program can request a service without having to understand where the server is in a distributed network or exactly what the interface to the server program looks like. Components can find out about each other and exchange interface info. as they are running.



Object Request Broker Architecture

GOOD WRITE

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)

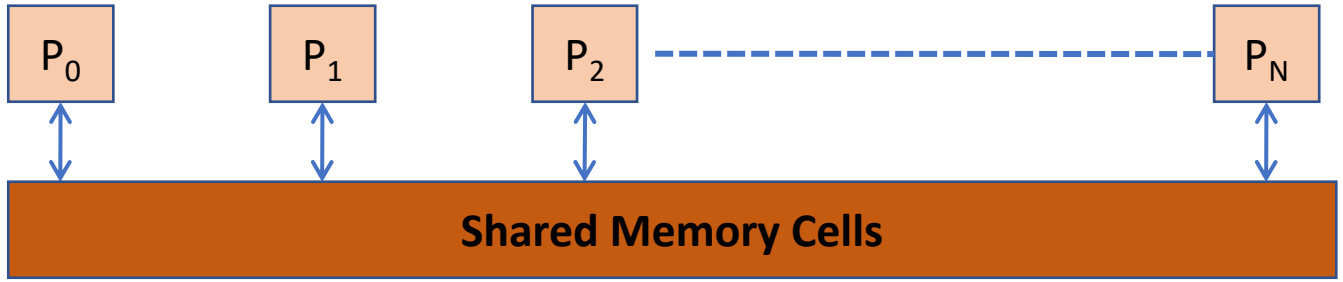
- Definition:
  - Is an abstract machine for designing the algorithms applicable to parallel computers
  - M' is a system <M, X, Y, A> of infinitely many
    - RAM's M1, M2, …, each M_i is called a processor of M'. All the processors are assumed to be identical. Each has ability to recognize its own 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

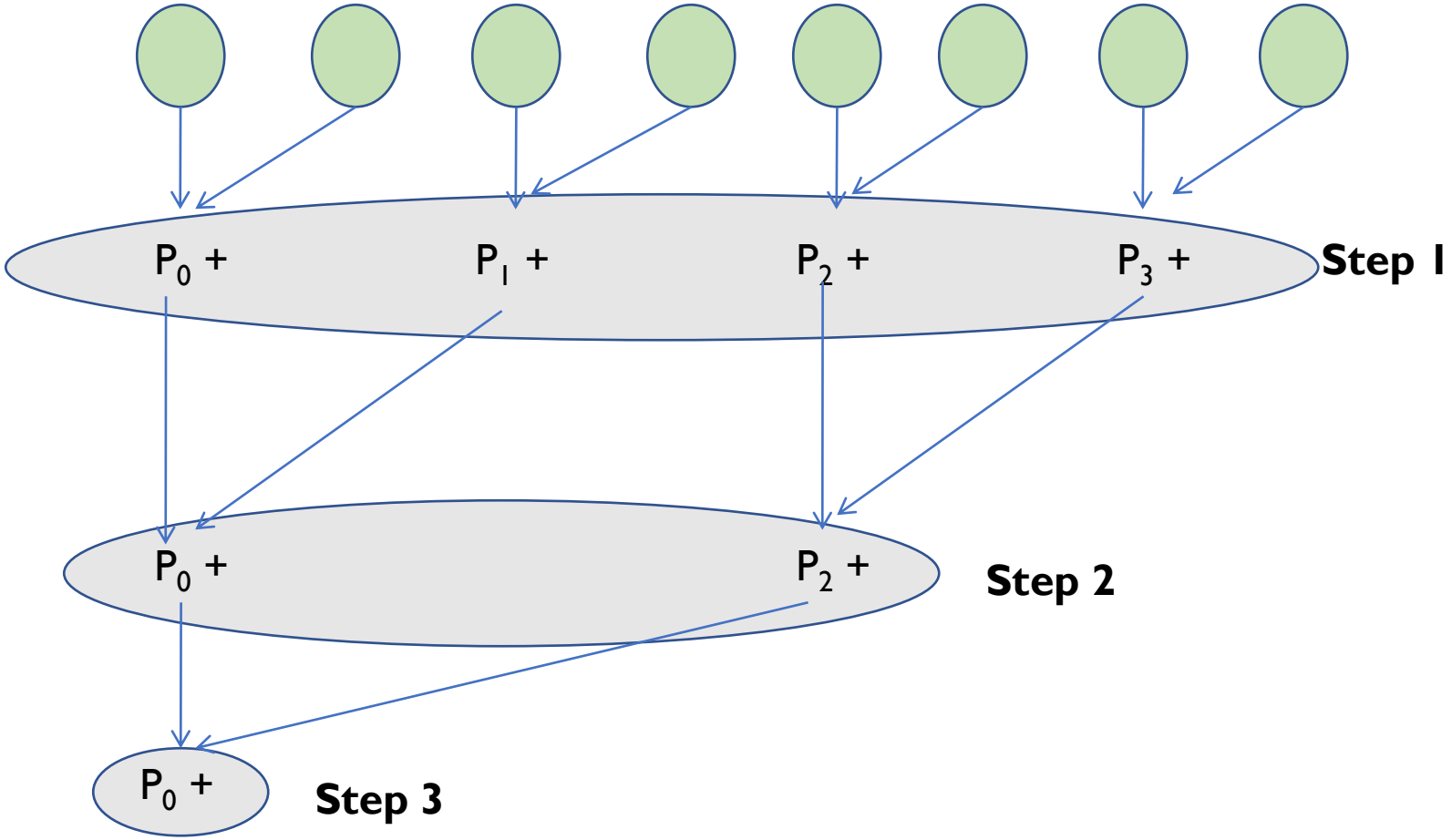
Strengths of PRAM

- 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]
                 }
            }
}
```

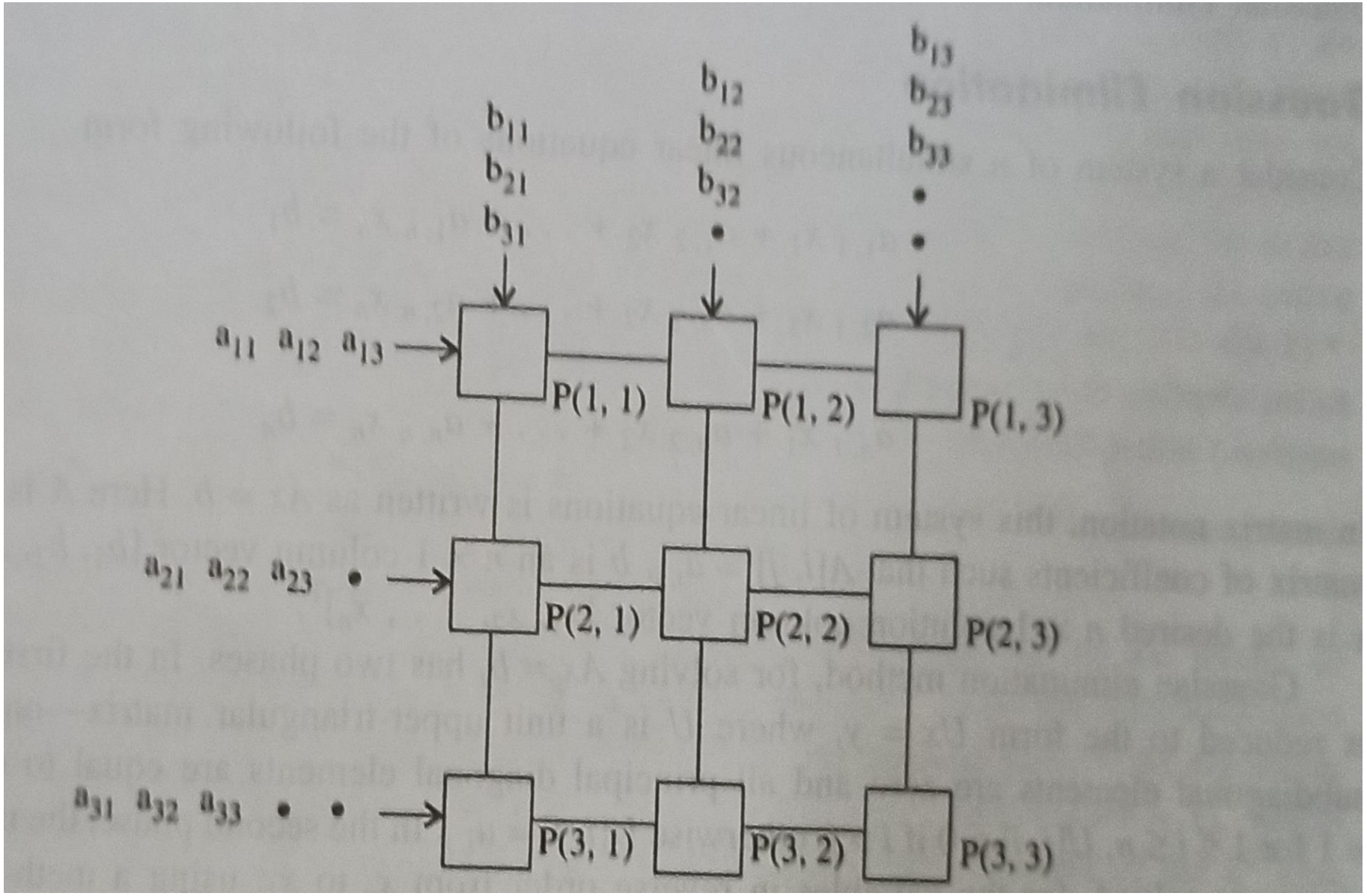
Matrix multiplication using nxnxn Processors on CREW PRAM

```
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**

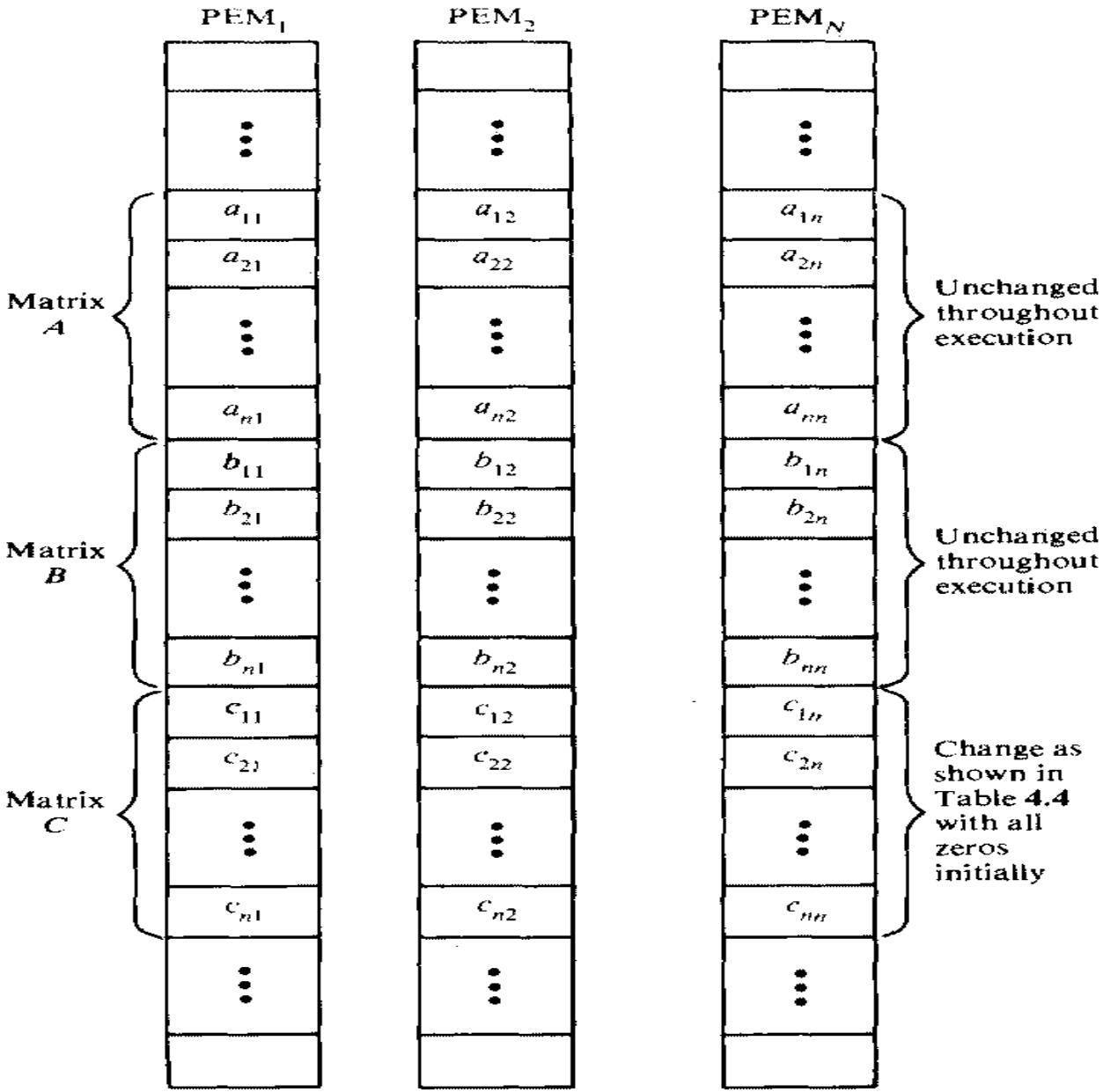
Par for $k = 1$ **to** n **Do**

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

End of j loop

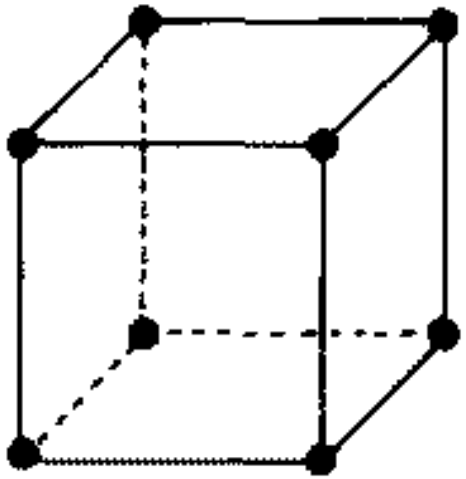
End of i loop

Memory allocation for matrix multiplication

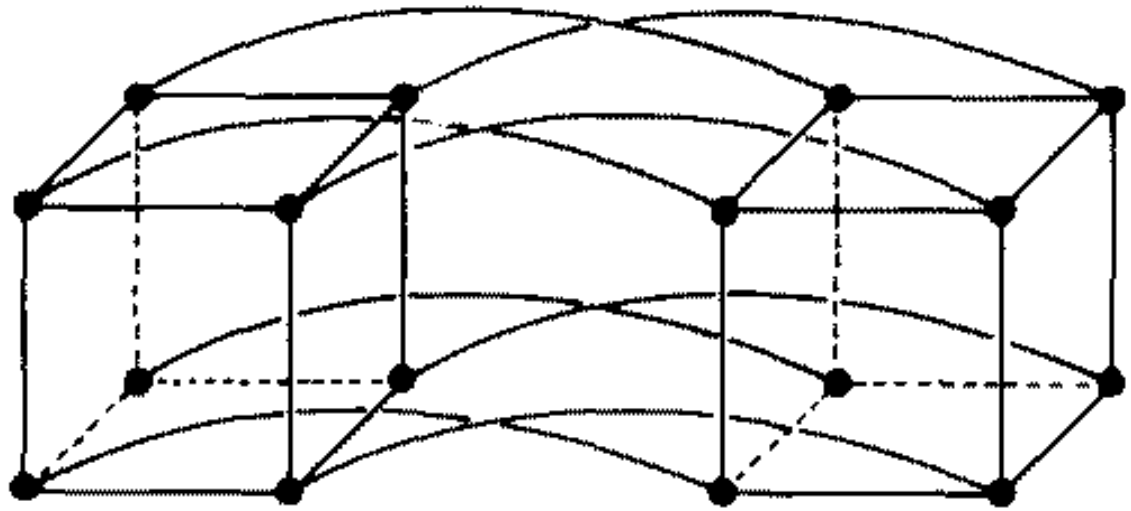


Successive content of C array in the memory

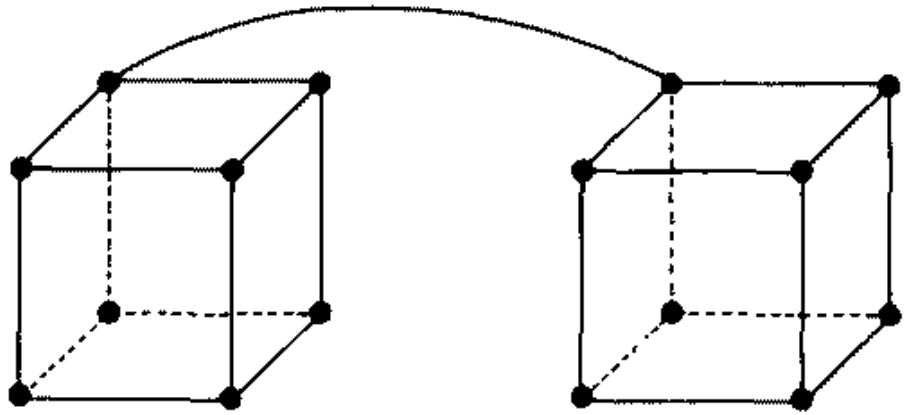
Outer loop i	Inner loop j	Parallel SIMD operations on $k = 1, 2, \ldots, n$			
		$c_{i1} \leftarrow c_{i1} + a_{ij} \times b_{j1}$	$c_{i2} \leftarrow c_{i2} + a_{ij} \times b_{j2}$	$\cdots$	$c_{in} \leftarrow c_{in} + a_{ij} \times b_{jn}$
1	1	$c_{11} \leftarrow c_{11} + a_{11} \times b_{11}$	$c_{12} \leftarrow c_{12} + a_{11} \times b_{12}$		$c_{1n} \leftarrow c_{1n} + a_{11} \times b_{1n}$
	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

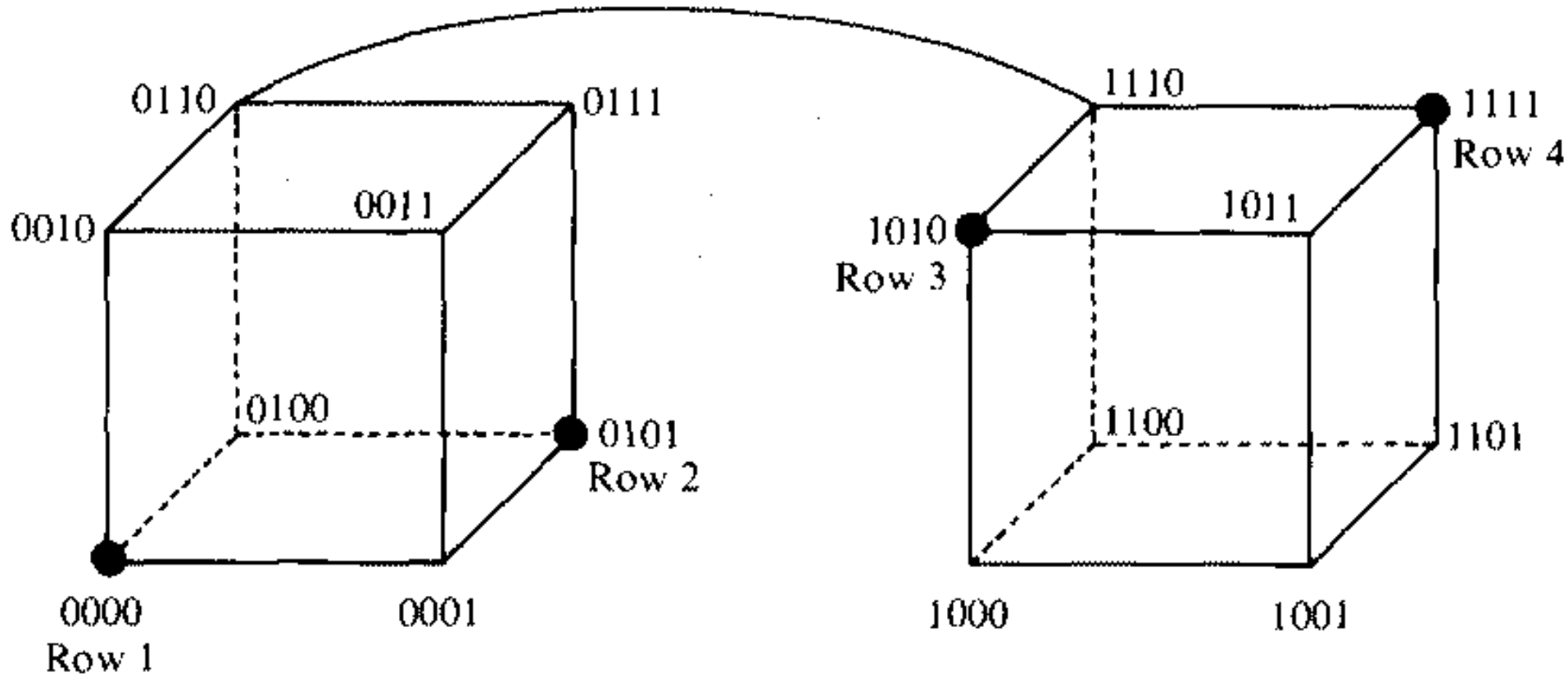


(*b*) A 4 cube formed from two 3 cubes



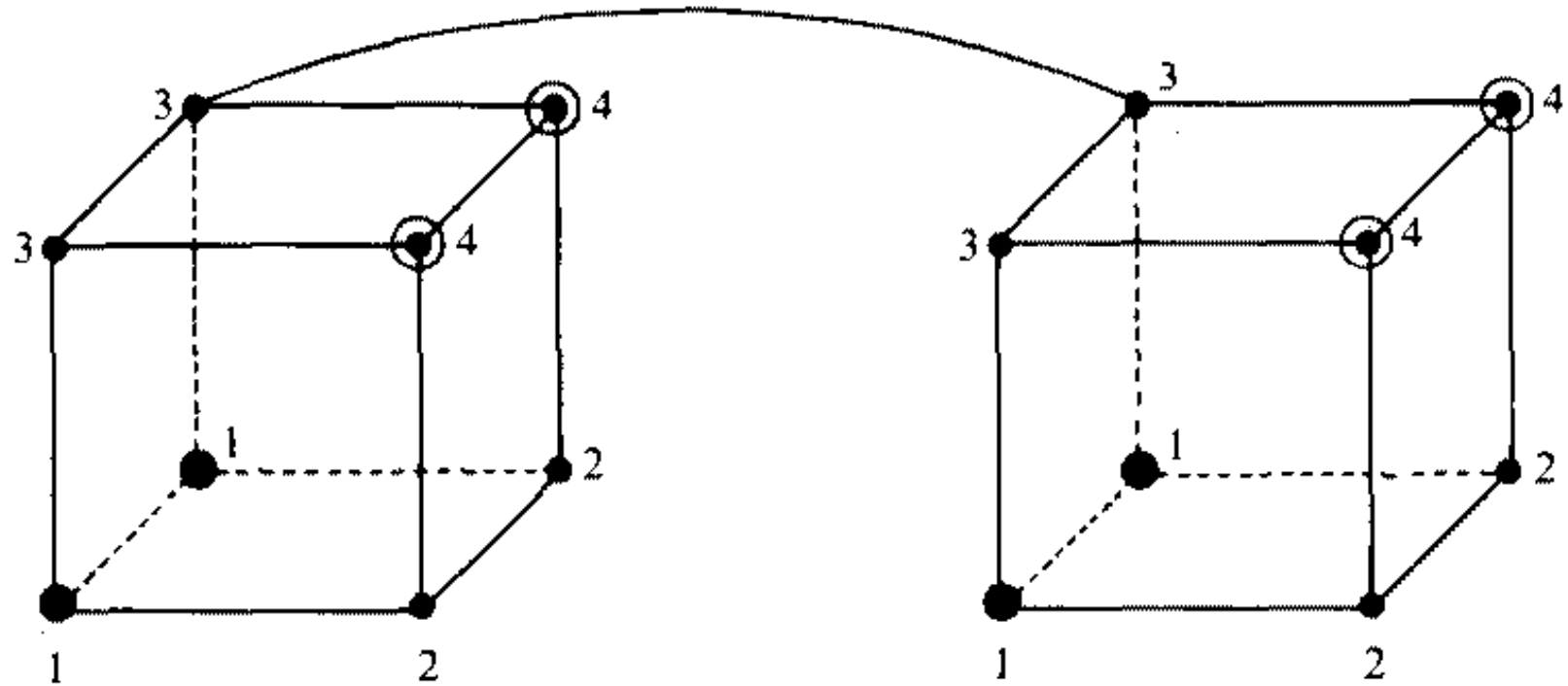
(c) The 4 cube showing only one of eight fourth-dimension connections.

Initial distribution of rows of A



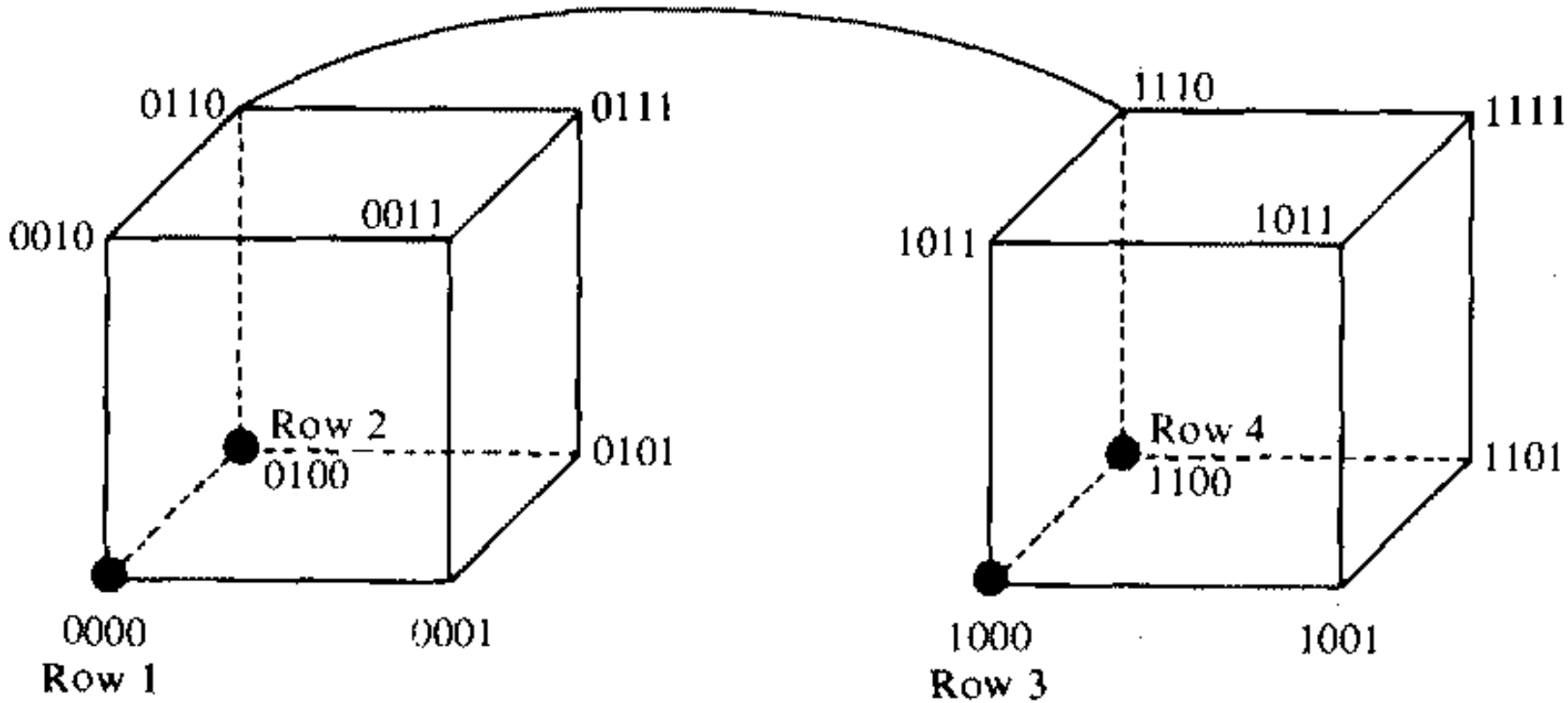
(*a*) Initial distribution of rows of *A*

4-way broadcaste of rows of A



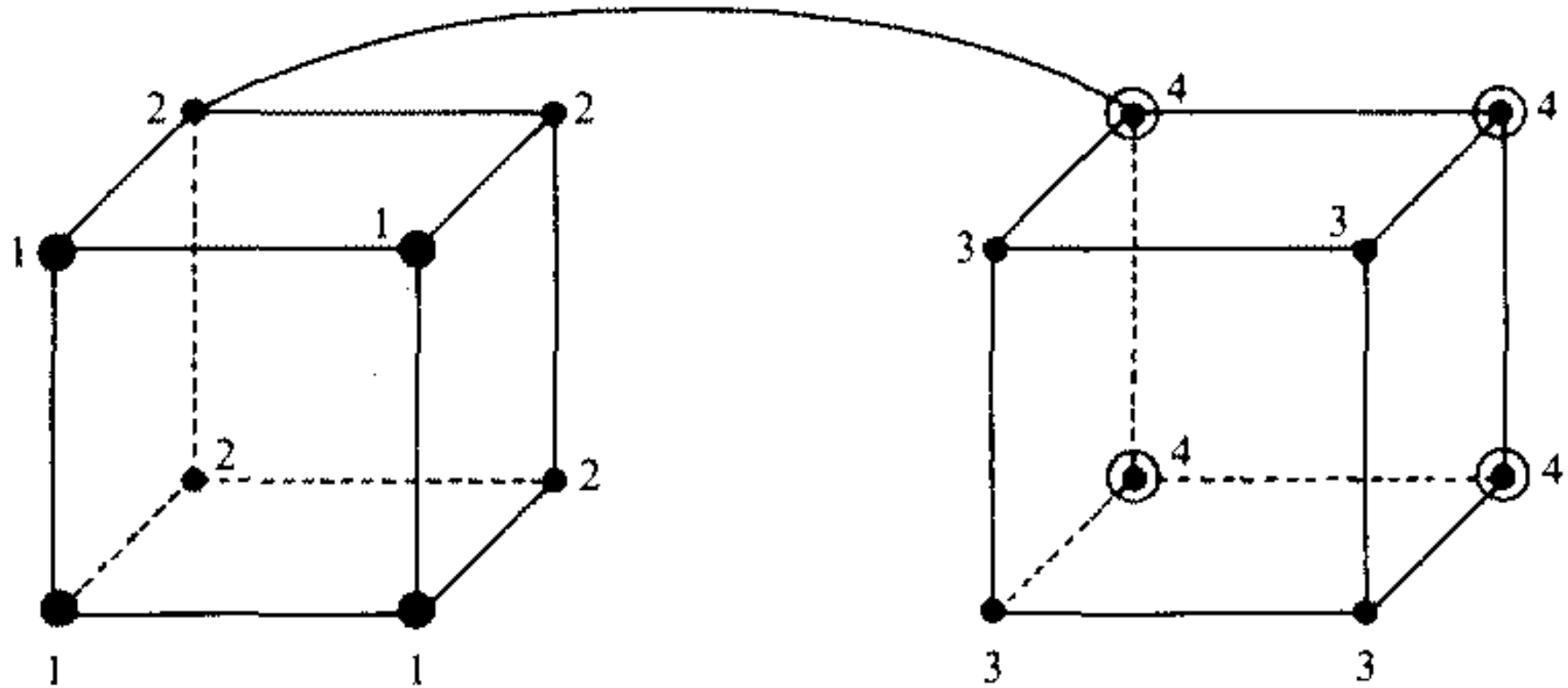
(*b*) 4-way broadcase of rows of *A*

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 5.20*a* 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 5.20*b*.

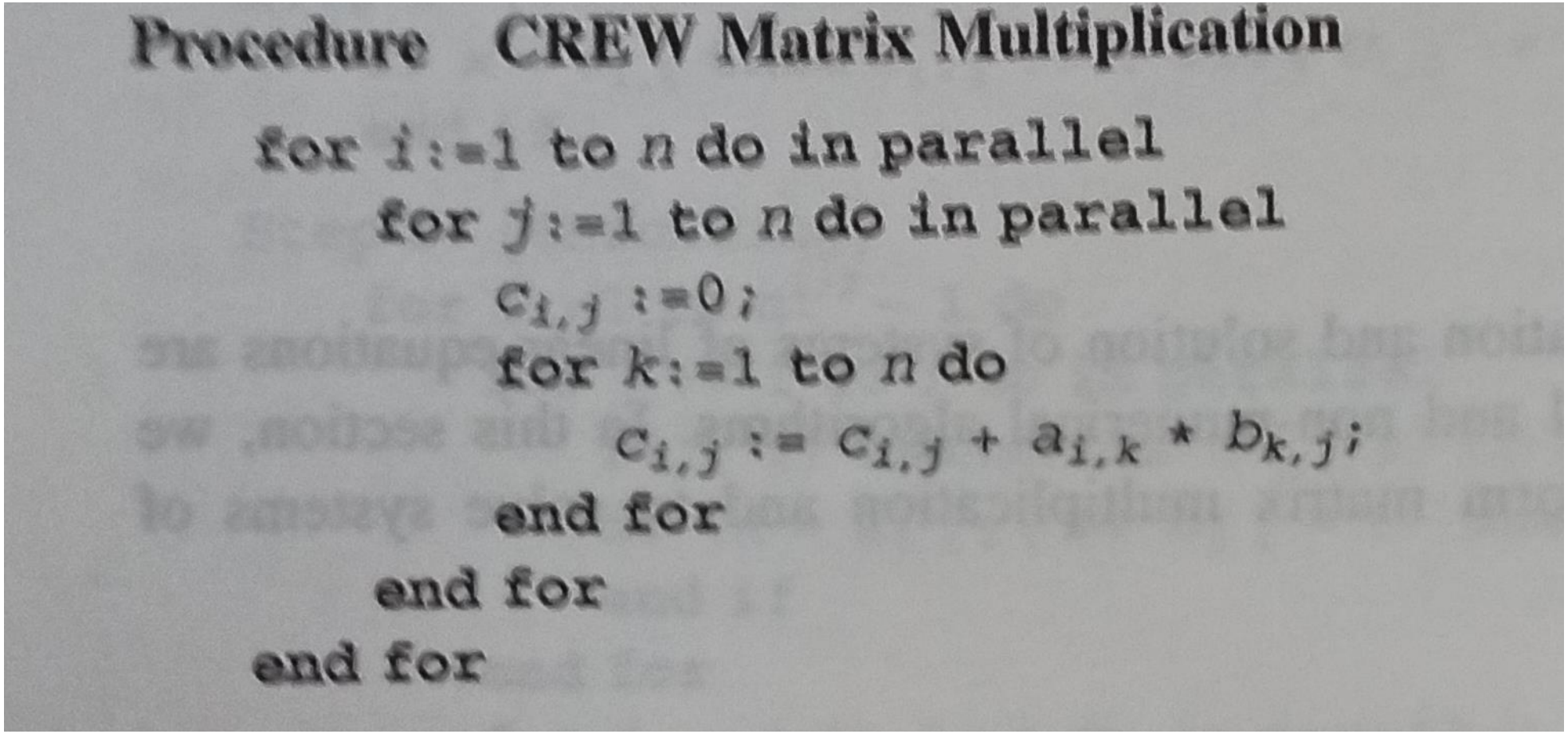
Example 5.5: An $O(n \log_2 n)$ algorithm for matrix multiplication

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 5.20*c*).
2. N-way broadcast each row of B^t to all PEs in the m cube

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

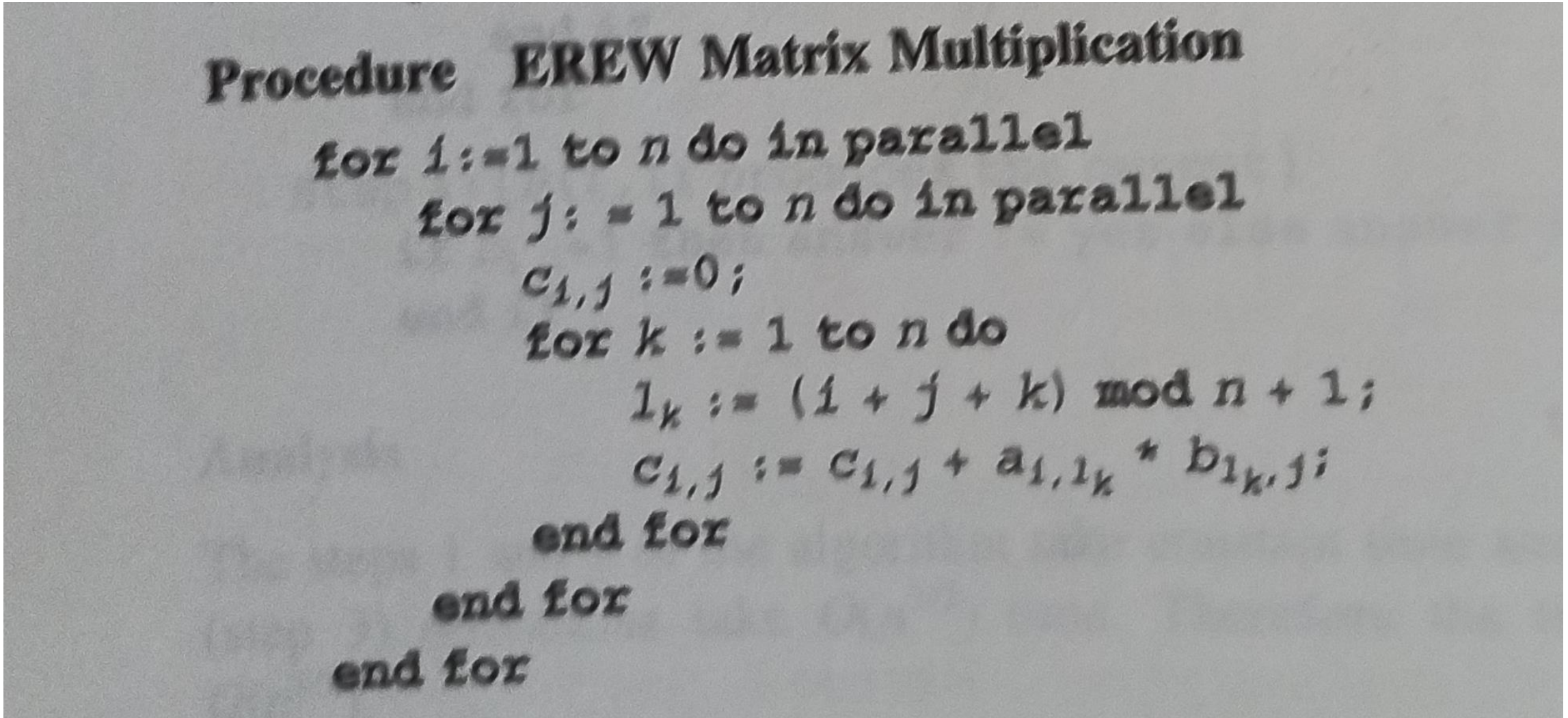
   in $n \log_2 n$ steps (Figure 5.20*d*).
3. N-way broadcast each row of A residing in PE $p_{2m-1} \cdots p_m p_{m-1} \cdots p_0$ to all PEs in the m cube $x_{2m-1} \cdots x_m p_{n-1} \cdots p_0$ in $n \log_2 n$ steps (Figure 5.20*b*). All the n rows can be broadcast in parallel.
4. Each PE now contains 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_{1,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
```