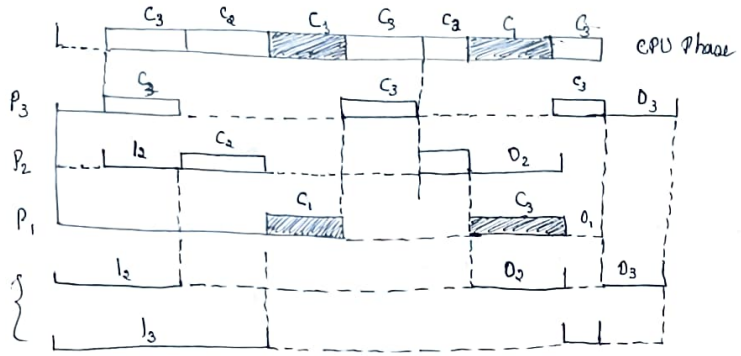


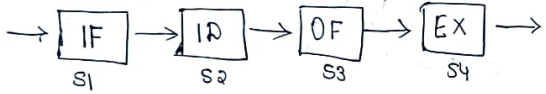
I → Input, O → Output, C → Compute

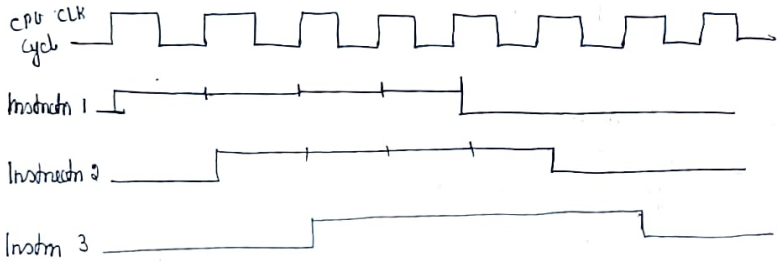
Parallel Computer Architecture :-

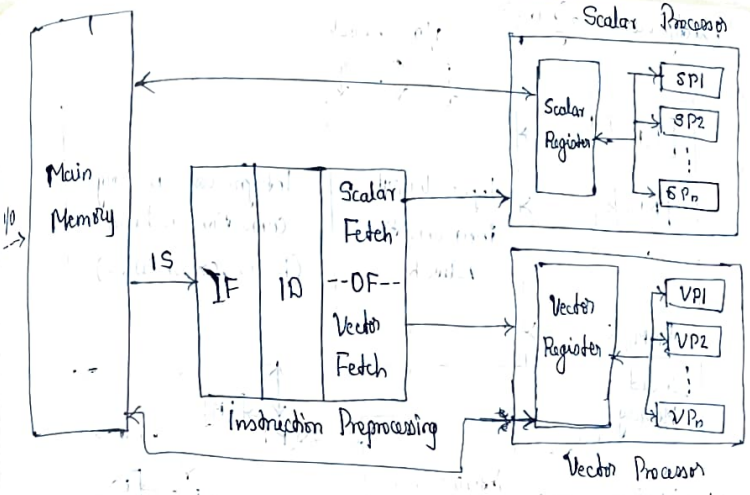
* Parallel computers are those that support parallel processing.
* They devided into 3 architectural configuration
  → Pipelined Computer → Array processor → Multiprocessor system.

Pipelined Computer :-

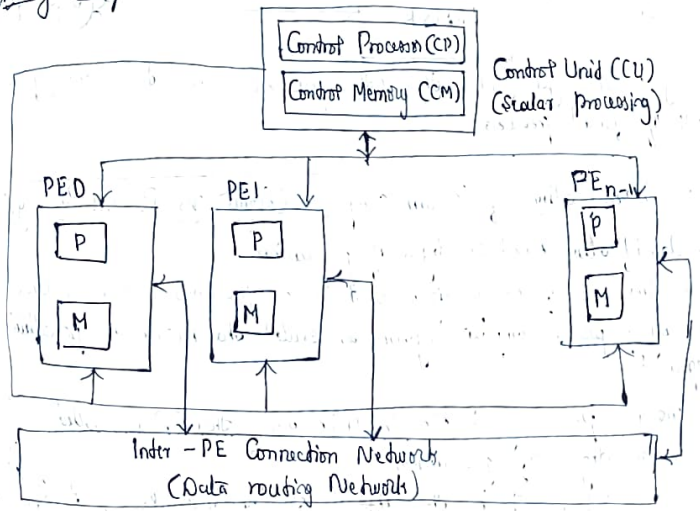
* Pipelined processor is



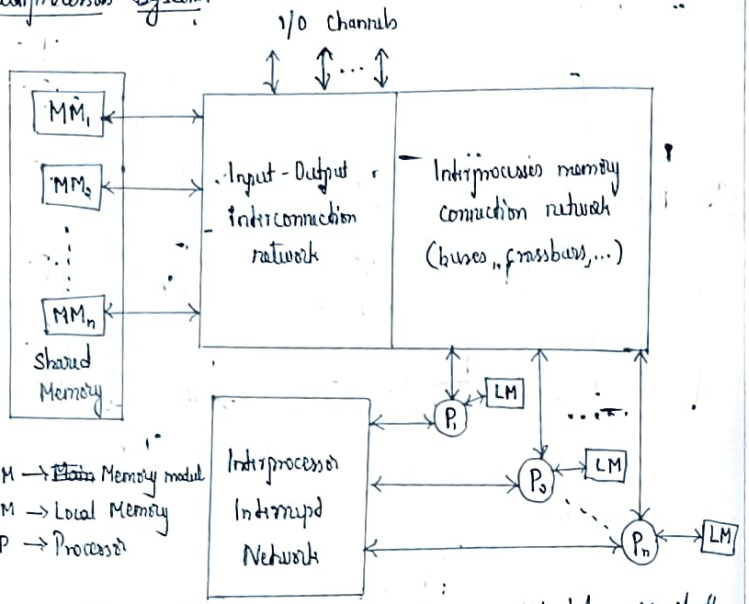




Array Computers :-



Multiprocessor System:-



*If any processor get failed then other processor will take care of the work of failed process.

Feng's Classification of Parallel computers (Serial vs. Parallel Processing):-

*This classification based on degree of parallelism.

*The maximum no. of binary digits that can be processed within a unit time by a computer system is called the maximum parallelism degree 'P'.

*If a processor processing P bits in unit time, then P is called maximum degree of parallelism.

*Let i = 1, 2, 3, ..., T be the different timing instants of $P_1, P_2, \ldots$ P_T be the corresponding bits processed, then

Average parallelism degree, $P_a = (P_1 + P_2 + \ldots + P_T)/T$

Processor Utilization, $U = P_a / P$

*If computing power is fully utilized then, $P_i = P$ hence $U = 1$ or 100%

*The maximum degree of parallelism depends on the structure of the Arithmetic and Logic Unit.

*The maximum degree of parallelism is given by the product of number of bits in a word (n) and the number of words processed in parallel (m)

word = 1101010001 (n bits) } m words,
1001010101
1000111101

*The classification is based on the way contents stored in the memory are processed. The contents can be either data / instruction.

*According to Fengs classification computer architecture is classified as fr.

1) Word Serial Bit Serial (WSBS)

*One bit of one selected words is processed at a time. This represents serial processing and needs maximum processing time: i.e. n=1, m=1

2) Word Serial Bit Parallel (WSBP)

*All the bits of selected word are processed at a time. Bit parallel means all bits of a word, i.e. n>1, m=1

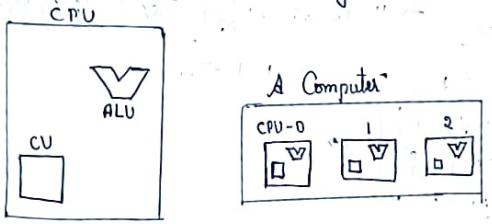
3) Word Parallel Bit Serial (WPBS):

*One bit from all words are processed at a time. Word parallel signifies selection of all word. i.e. n=1, m>1

4) Word Parallel Bit Parallel (WPBP)

*All bits of all words are processes at a time. Maximum parallelism is achieved there. i.e, n>1, m>1

Handler's Classification (Pipeling V/s Parallel)



$t = \langle K \times K', D \times D', W \times W' \rangle$

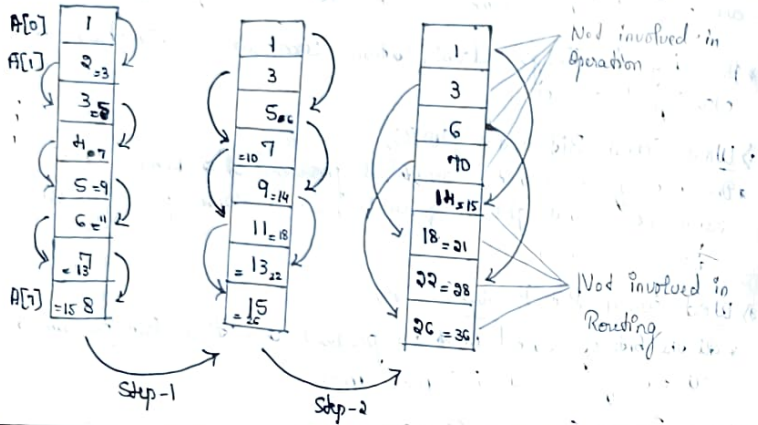
T(c) = ↳ How many Size of ALU ↳ Size of ALU (bits)

↳ how many processing control unit are classifies

Application of Parallel Processing:-

* Weather forecasting → Military Appl^n, * AI, Medical Appl^n
* Oceanography * Astrophysics

Text Book → Hwang & Briggs.



Selection Sorting Algorithm:-

Sequential:-

1) Find the minimum value in the list
2) Swap it with the value in the 1st position
3) Repeat the steps 1 & 2 (now starts from 2nd position)

Ex:- 64 | 25 12 22 11 => 11 | 25 | 12 22 64 => 11 12 | 25 22 64

min = 64 min = 12 min = 22

11 12 22 | 25 64 => 11 12 22 25 | 64

min = 25 min

n-1 times

```
for(int i=0; i<n-1; i++) {
    for(int j=i; j<n; j++) {
        if(a[j] > a[j+1]) {
            v = a[j+1];
        }
```

or

```
for(int i=0; i<n-1; i++) {
    pos = i;
    min = a[i];
    for(int j=i+1; j<n; j++) {
        if (min > a[j]) {
            min = a[j];
            pos = j;
        }
    }
    temp = a[i];
    a[i] = a[pos];
    a[pos] = temp;
}
```

Parallel: 64 25 12 22 11

Core 0	Core 1	Core 2	Core 3	Core 4
min=64 pos=0	min=25 pos=0	min=12 pos=0	min=22 pos=0	min=11 pos=0
input[j] < min				
j=0, 64<64 pos=0	64<25 pos=0	64<12 pos=0	64<22 pos=0	64<11 pos=0
j=1 25<64 pos=1	25<25 pos=0	25<12 pos=0	25<22 pos=0	25<11 pos=0
j=2 12<64 pos=2	12<25 pos=1	12<12 pos=0	12<22 pos=1	12<11 pos=0
j=3 22<64 pos=3	22<25 pos=2	22<12 pos=0	22<22 pos=1	22<11 pos=0
j=4 11<64 pos=4	11<25 pos=3	11<12 pos=1	11<22 pos=2	11<11 pos=0

→ Parallelly, ↓ Sequencially

Output C[] = {11, 12, 22, 25, 64}//

```
__global__ void selectionSort(int *inputA, int arrSz, int *outputA) {
    int tid = threadIdx.x;
    int min = inputA[tid];
    int pos = 0;
    for(int j=0; j<arrSz; j++) {
        if(inputA[j] < min || (inputA[j] == min && j<tid)) {
            pos++;
        }
    }
    outputA[pos] = min;
}
```

Handler's Classification:-

$T(C) = \langle K \times K' ; D \times D' : W \times W' \rangle$

where,

K → no. of processor in computer / control unit

K' → no. of PCU that are pipelined

D = no. of ALU

D' = no. of ALU pipelined

W → word length of ALU

W' → The no. of pipelined stages on all ALU's

Ex:- TI-ASC has one controller controlling four arithmetic unit, Each ALU is an eight stage pipeline with 64-bit word length.

Sol^n:

$T(TI\text{-}ASC) = \langle 1 \times 1 ; 4 \times 1 ; 64 \times 8 \rangle$

Necessity of Data Routing:-

Consider an array

$$A = (A_0, A_1, \ldots A_{n-1})$$

Now for computing:

$$S(n) = \sum_{i=0}^{n-1} A_i$$

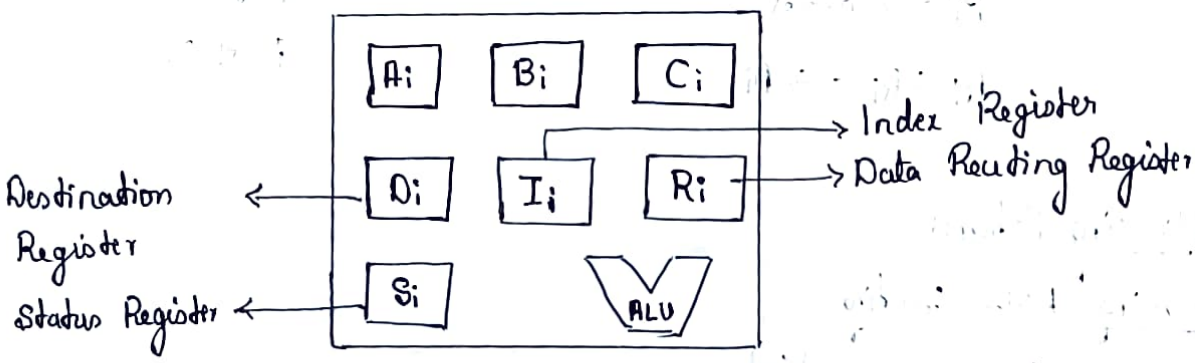
In SISD, the same thing would be take 8 steps or loop by formula:

$$Sum = sum + A[i];$$

∴ It will take $O(n)$

∴ Generalized time complexity $O(\log_2 N)$

Components in a PE:-



A_i, B_i and C_i are the general purpose registers (GPR)

Green-Computing → Saving the power of computation

* Only the content of R_i are transfered to the other PE's during data transfer.
* If $N = 2^m$ (m → no. of bits required to identify a PE) are there then D_i will hold those 'm' bit address of the destination PE
* Each PE_i is either active or inactive during instruction cycle.

$S_i = 1$ then "Active" , $S_i = 0$ then "Inactive"

Algorithm:-

Step-1:- A_i would transfer data in R_i

$A_i \longrightarrow R_i$, $i = 0-6$

$R_i \longrightarrow R_{i+1}$; $i = 0-6$

$A_i + R_i \longrightarrow A_i$; $i = 1-7$

Step-2:-

$A_i \longrightarrow R_i$ $i = 0-5$

$R_i \longrightarrow R_{i+2}$ $i = 2-7$

$A_i + R_i \longrightarrow A_i$

Step-3:- $A_i \longrightarrow R_i$ $i = 0-3$

$R_i \longrightarrow R_{i+4}$ $i = 0-3$

$A_i + R_i \longrightarrow A_i$ $i = 4-7$.

Masking Scheme:

During Data Routing

Step-1: PE_7 is disabled

Step-2: PE_6 & PE_7 are disabled

Step-3: PE_5, PE_6 & PE_7 are disabled

During Addition:

Step-1: PE_0 is not involved

Step-2: PE_0, PE_1 are not involved

Step-3: PE_0 - PE_3 are not involved

SIMD Interconnection Network:-

* Interconnection networks are needed to route data -
  → from processors to memories or
  → from one PE to another.

Measures of Interconnection Performance:

(Metrics for Static Network)

* Node degree (d) → Number of edges (links or channels) incident on a node.

* Node degree tells no. of I/O ports associated with a node. and should ideally be small and constant.

* Diameter (D) :- is a n/w is the max shortest path b/n any two nodes.

* The 'D' of the network indicates the max number of distinct hops b/n any 2^2 nodes

* Measured by the no. of links traversed; this should be as small as possible.

* Channel Bisection Width (b):- the minimum no. of edges cut to split a network into two parts each having the same no. of nodes.

Interconnection Network Taxonomy:-

1) Static → 1D, 2D, HC (Hyper Cube) & 3D

2) Dynamic → Bus-based, Switch-Based
   - Bus-based → Single, Multiple
   - Switch-Based → Single switch, Multi switch, Crossbar

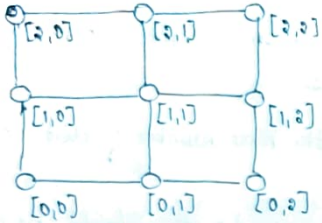
Linear Array (1D)

* N nodes connected by N-1 links
* Interior nodes will have the degree 2 & end nodes have degree 1
* Diameter = n-1
* Bisection width = 1

SDL:- Static Networks: Ring, Chordal ring, Star, Completely connected network, K-ary n-cube network, 3-CCC network

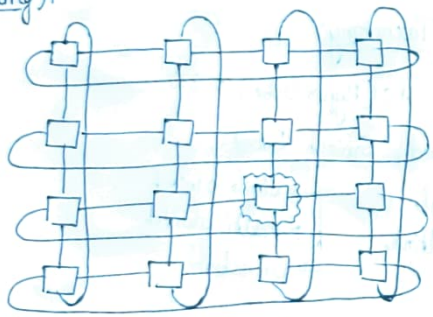
---

2D (Mesh):



* A n-dimentional mesh is an extension of the linear array
* Node degree = 2 to 4 [min is 2 & max = 4
  (2,0) => 2 link (1,1) = 4
* A k-dimensional mesh with $N = n^k$ (n is the same mesh) nodes has an interior node degree of 2k and the network diameter is, k(n-1)

2D (Ring):



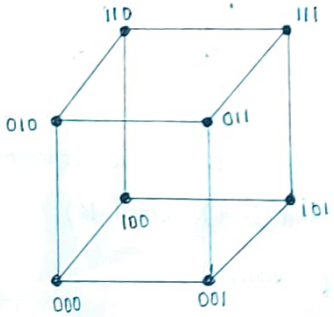
This is called Torus Network

* A n-dimensional torus or wraparound mesh is an extension of the linear array
* Degree = 4
* The marked node is furthest node so its diameter = 4.

1D Torus:

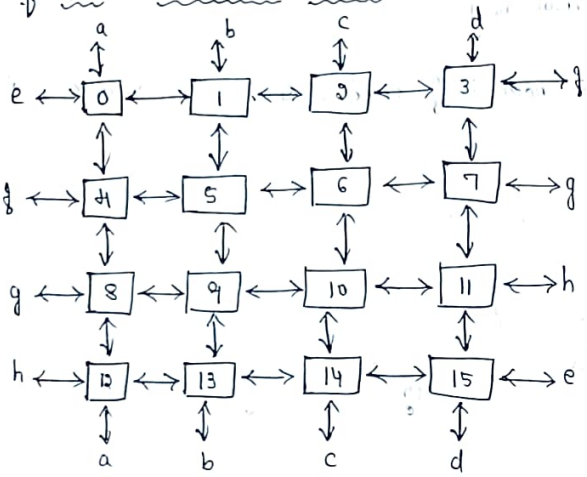
3 Static Interconnection Network 3D:-

Hypercubes



* An hypercube is a multi-dimensional mesh with exactly two processors in each dimension.
* A n-cube consists of $N = 2^n$ nodes.
* The node degree of n-cube equals to n & so does the network diameter.

Design of Mesh / Interconnection Network (ILLIAC IV Nw)



* In the Illiac IV, each processor 'i' was connected to processors, $\{i+1,\ i-1,\ i+4,$ and $i-4\}$ (mod 16)

* Here are the routing functions:

$R_{+1}(i) \equiv (i+1) \bmod N$

$R_{-1}(i) \equiv (i-1) \bmod N$

$R_{+r}(i) \equiv (i+r) \bmod N$

$R_{-r}(i) \equiv (i-r) \bmod N$ where, $r = \sqrt{N}$

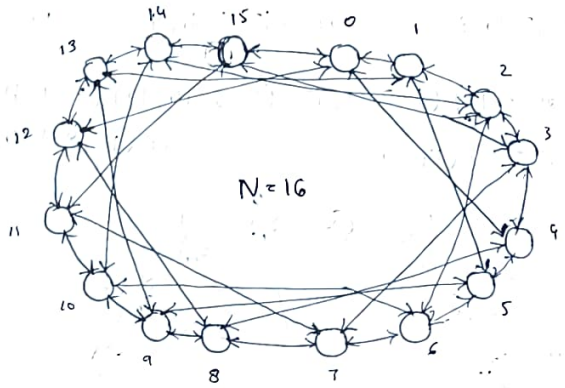
where $0 \le i \le N-1$

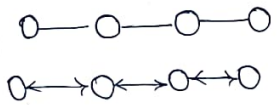
$\therefore R_{+1} = (0, 1, 2, 3, \ldots, 15)$

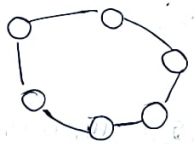
$R_{-1} = (15, 14, 13, \ldots, 0)$

$R_{+4} = (0,4,8,12)(1,5,9,13)(2,6,10,14)(3,7,11,15)$

$R_{-4} = (15,11,7,3)(14,10,6,2)(13,9,5,1)(12,8,4,0)$



2) Linear Array $\Rightarrow$  $\Rightarrow R(i) = i+1,\ 0 \le i \le N-1$ and $R(i) = (i-1)$

3)  $\Rightarrow R(i) = (i+1) \bmod N$

Barrel Shifting Network:

* It is also called plus-minus-2^i network.

* Routing function:

$B_{+i}(j) = (j + 2^i) \bmod N$

$B_{-i}(j) = (j - 2^i) \bmod N$

where, $0 \le j \le N-1$ and $0 \le i \le \log_2 N$

$\therefore$ If $N=16$, then $0 \le j \le 15$ and $0 \le i < 4$

$\therefore B_{+0}(j) = (0, 1, 2, 3, \ldots, 15)$

$B_{-0}(j) = (15, 14, 13, 12, \ldots, 0)$

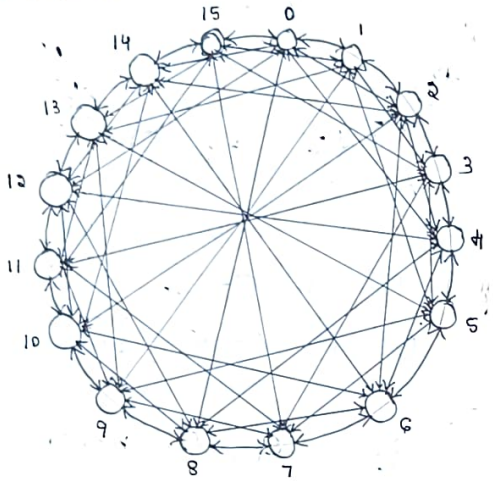
$B_{+1}(j) =$ ~~$(2, 3, 4, 5, \ldots, 15, 0, 1)$~~ $(0, 2, 4, 6, 8 \ldots 14)$ $(1, 3, 5, 7, 9, \ldots, 15)$

$\Rightarrow i=1,\ j=0$ $j=2$ $j=4$ $j=6$ ...

B_{-1} = (14, 12, 10, 8, ... 0), (15, 13, 11, 9, ... 1)

B_{+2} = (0, 4, 8, 12) (1, 5, 9, 13) (2, 6, 10, 14) (3, 7, 11, 15)

B_{-2} = (15, 11, 7, 3) (14, 10, 6, 2) (13, 9, 5, 1) (12, 8, 4, 0)



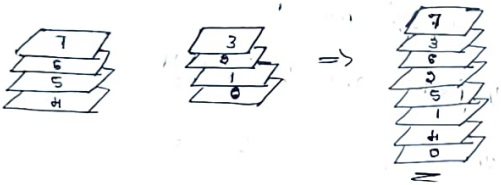
B_{+3} = (0, 8), (1, 9), (2, 10), (3, 11), (4, 12), (5, 13), (6, 14), (7, 15)

B_{-3} = (15, 7), (14, 6), (13, 5), (12, 4), (11, 3), (10, 2), (9, 1), (8, 0)

Perfect - Shuffle Interconnection:-

* This interconnection n/w is defined by the routing function

$S(a_{n-1}, \ldots, a_1, a_0)_2 = (a_{n-2} \ldots a_1 a_0 a_{n-1})_2$

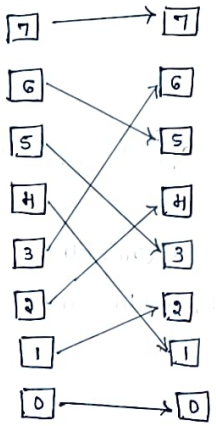


Perfect shuffle shift the 1st one bit to left.

then,

S(000) = 000, S(001) = 010, S(010) = 100, S(011) = 110,

S(100) = 001, S(101) = 011, S(110) = 101, S(111) = 111

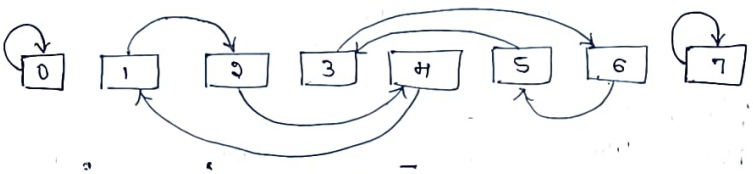


Perfect Shuffle.

Here Permutation can be expressed

S = (0) (1, 2, 4) (3, 6, 5) (7)

* a shuffle n/w is not a complete interconnection n/w.
* This can be seen by looking at what happens as data is recirculated through the network.
* An exchange permutation can be added to a shuffle n/w to make it into a complete interconnection structure.



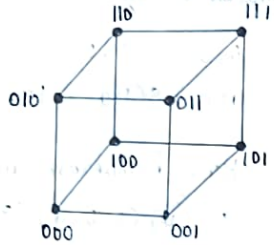
$E(a_{n-1} \ldots a_1 a_0)_2 = a_{n-1} \ldots a_1 \bar{a}_0$

Here permutation can be expressed as for Exchange permutation. i.e. complement the least significant bit.

S(000) = 001, S(001) = 000, S(010) = 011, S(011) = 010.

$\therefore$ S = (0, 1) (2, 3), (4, 5), (6, 7)

Hypercube:-

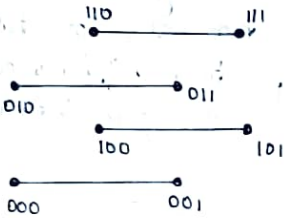


$C_i(a_{n-1} \dots a_{i+1}\, a_i\, a_{i-1} \dots a_0)_2 = (a_{n-1} \dots a_{i+1}\, \bar{a}_i\, a_{i-1} \dots a_0)_2$

$i = 0$

$C_0(000) = 001$, $C_0(001) = 000$, $C_0(010) = 011$, $C_0(011) = 010$,

$C_0(100) = 101$, $C_0(101) = 100$, $C_0(110) = 111$, $C_0(111) = 110$



$C_1(000) = 010$

$C_1(010) = 000$

$C_1(001) = 011$

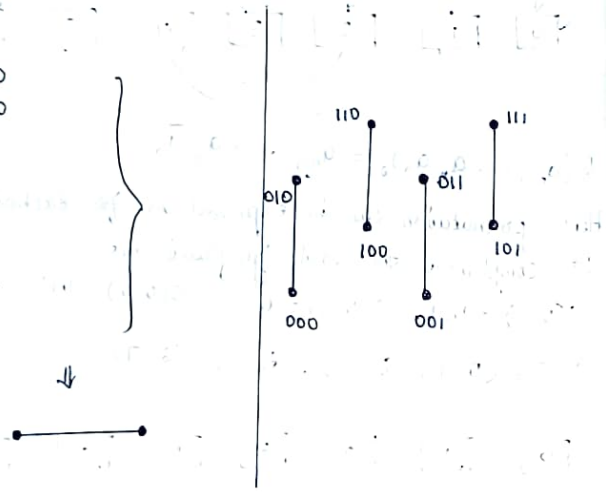
$C_1(011) = 001$

$C_1(100) = 110$

$C_1(110) = 100$

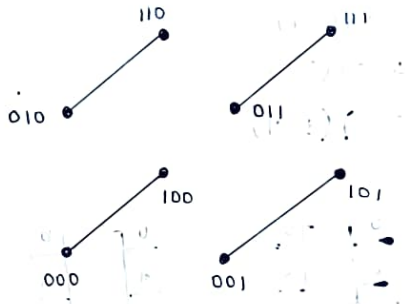
$C_1(101) = 111$

$C_1(111) = 101$



$C_2(000) = 100$, $C_2(100) = 000$, $C_2(001) = 101$, $C_2(101) = 001$,

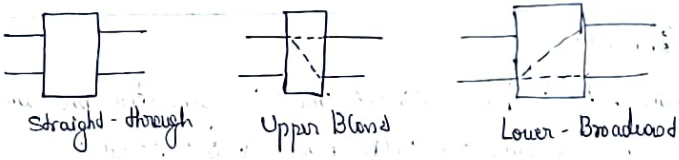
$C_2(010) = 110$, $C_2(110) = 010$, $C_2(011) = 111$, $C_2(111) = 011$

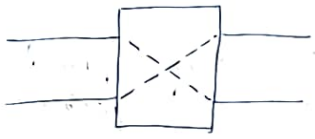


* A hypercube is a generalized cube. In a hypercube, there are 2^n nodes, for some n. Each node is connected to all other nodes whose numbers differs from id on only one bid position.

Multistage Interconnection Network:- (Switch Modules)

* A 2x2 switch can be configured for
  - → Straight - through
  - → Crossover
  - → Upper broadcast (upper input to both outputs)
  - → Lower broadcast (lower input to both outputs)





Crossover

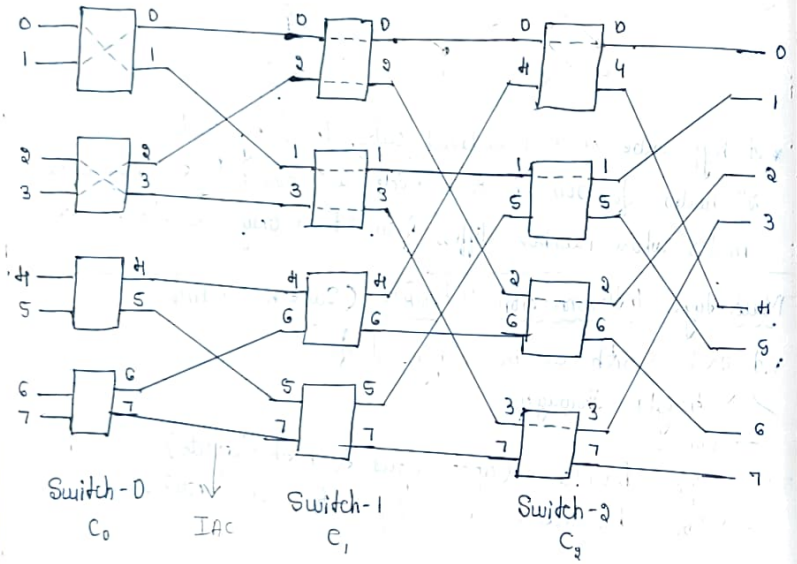
The routing function have this form

$$C_i(a_{n-1} \cdots a_{i+1}\, a_i\, a_{i-1} \cdots a_0)_2 = a_{n-1} \cdots a_{i+1}\, a_i\, a_{i-1} \cdots a_0$$

$C_0 = (0\,1)(2\,3)(4\,5)(6\,7)$ → Crossover → Straight through → Straight through

$C_1 = (0\,2)(1\,3)(4\,6)(5\,7)$ → Straight → Crossover → Straight

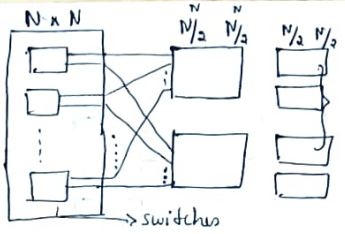
$C_2 = (0\,4)(1\,5)(2\,6)(3\,7)$ → Straight → Straight → Crossover



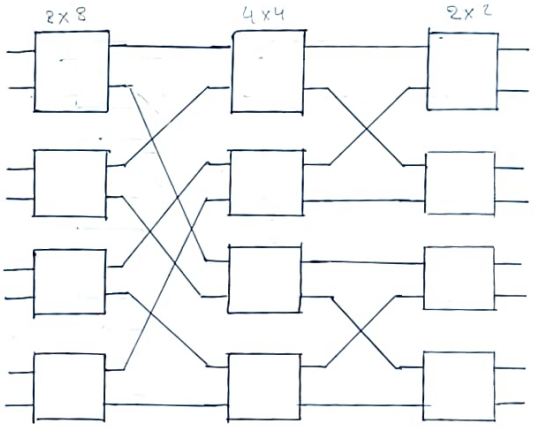
Baseline Network:-

* A baseline network has a simple recursive generation procedure.
* The first stage contains one N×N block & 2nd stage two $(N/2) \times (N/2)$ subblocks.
* The construction process is recursively applied to the subblocks until the N/2 subblocks is of size 2×2 are reached.
* The subblocks are straight and crossover.

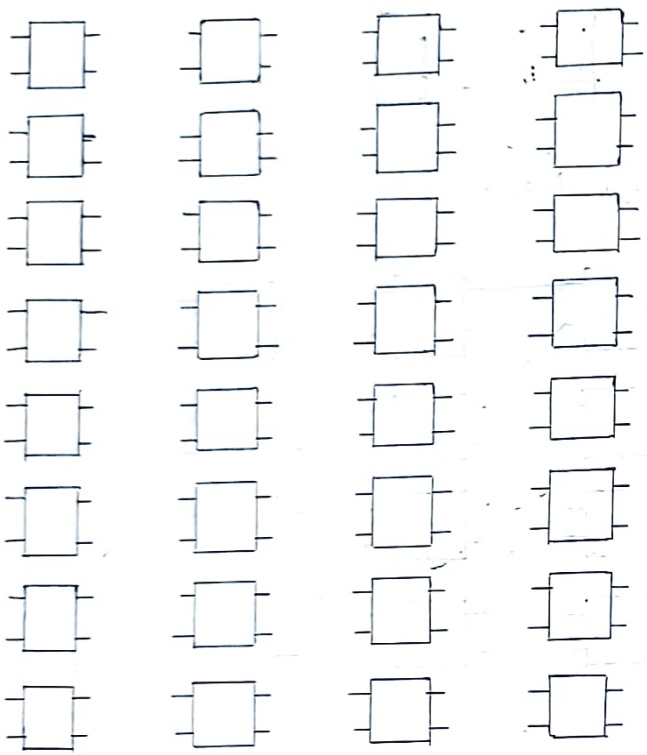
Ex:-



8×8 Baseline Network:-



16x16 Baseline Network:



Elementary Parallel Algorithms:

1) Summation (Hypercube SIMD)

Parameter: $n \rightarrow$ number of elements to add
$p \rightarrow$ number of processing elements

$n = 100$, $p = 16$

$\lceil \frac{100}{16} \rceil = 7$

$\lfloor \frac{100}{16} \rfloor = 6$

```
Global j
Local  local.size, local.value[1...⌈n/p⌉], sum, tmp
{
   for all P_i, where 0 ≤ i ≤ p-1 do
      if i < (n modulo p) then
         local.size ← ⌈n/p⌉
      else
         local.size ← ⌊n/p⌋
      endif
      sum ← 0;
   endfor
   for (j=1; j<=⌈n/p⌉; j++) do
      for all P_i, where 0 ≤ i ≤ p-1 do
         if local.size ≥ j then
            sum ← sum + local.value[j]
         endif
      endfor
   endfor
```

O(1)

O(n/p)

```
    for (j = log p - 1; j >= 0; j--) do
        for all p_i, where 0 ≤ i ≤ p-1 do
            if i < 2^j then
                temp ⇐ [i + 2^j] sum
                sum ← sum + temp
            endif
        endfor
    endfor
}
```

O(log p)

∴ Time Complexity = O(1) + O(n/p) + O(log p)

= O(n/p + log p)

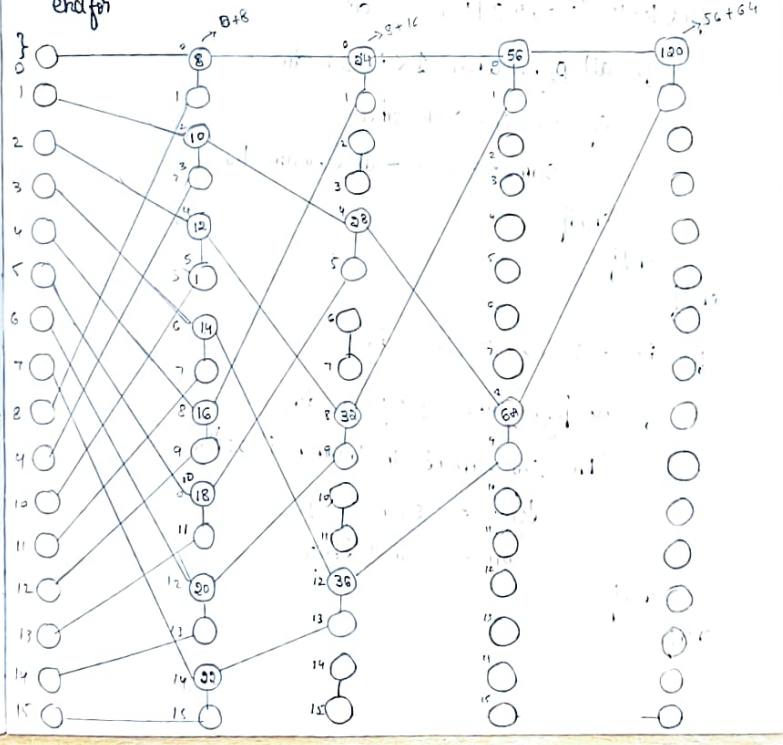
---

2) Summation (Shuffle - Exchange SIMD) :-

```
Parameter n, p
Global j
Local local.size, local.value [1 ... ⌈n/p⌉], sum, temp
{
    for all p_i, where 0 ≤ i ≤ p-1 do
        if i < (n modulo p) then
            local.size ← ⌈n/p⌉
        else
            local.size ← ⌊n/p⌋
        endif
        sum ← 0;
    endfor
```

```
    for (j = 1; j <= ⌈n/p⌉; j++) do
        for all p_i, where 0 ≤ i ≤ p-1 do
            if local.size ≥ j then
                sum ← sum + local.value [j]
            endif
        endfor
    endfor
    for (j = 0; j <= log p - 1; j++) do
        for all p_i, where 0 ≤ i ≤ p-1 do
            shuffle (sum) ⇐ sum
            exchange (temp) ⇐ sum
            sum ← sum + temp
        endfor
    endfor
}
```

S = {0) (1, 2, 4, 8) (3, 6, 12, 9) (5, 10,



8 3) Summation (2D Mesh):-

```
Parameter d (Mesh has size d*d)
Global i
Local tmp, sum, local.size, local.value[n/p]
{
    for all p_{i,j} , where 1 ≤ i, j ≤ d do
        if (i <= (n modulo p) && j == 1) then
            local.size ← ⌈n/p⌉
        else
            local.size ← ⌊n/p⌋
        endif
        sum ← 0;
    endfor

    for (k = 1; k <= ⌈n/p⌉; k++) do
        for all p_{i,j} , where 1 ≤ i, j ≤ d do
            if local.size ≥ k then
                sum ← sum + local.value[k]
            endif
        endfor
    endfor

    for (i = d-1; i >= 1; i--) do
        for all p_{j,i} , where 1 ≤ j ≤ d do
            (Processing elements in column i' active)
                temp ⇐ east(sum)
                sum ← sum + temp
        endfor
    endfor

    for (i = d-1; i >= 1; i--) do
        for all p_{i,1} do
            tmp ⇐ south(sum)
            sum ← sum + tmp
        endfor
    endfor
}
```

Heterogeneous Computing with OpenCL

* The Open Computing Language (OpenCL) is a heterogeneous programming framework
* OpenCL is a framework for developing applications.
* It supports a wide range of levels of parallelism.
* It currently supports CPUs and it has been adopted into graphics card tools by both AMD & NVIDIA
* The architecture supported including CPU and also GPU.

* The Khronos group has developed by OpenCL API
* Using the C99 language & currently following the specifications.
* The model set by OpenCL consists portable, vendor and device-independent programs.
* The OpenCL API is a C with C++ wrapper API that is defined in terms of the C API

OpenCL Specifications:

1) Platform Model:- Specifies that there is one process coordinating execution and one or more processes capable of executing OpenCL C-Code.
* It defines the abstract hardware model. It defines the relationship b/n the host & device.

2) Execution Model:- Defines how the OpenCL environment is configured
* This includes setting up an OpenCL context on the host.
* The host,

3) Memory Model:- Defines abstract memory hierarchy that kernels use, regardless of the actual memory architecture.
* The data within the kernel is allocated by the programmer to specific parts of an abstract memory hierarchy

4) Programming Model:- Defines how the concurrency model is mapped to physical hardware.
* Finally the hardware thread context that executes the kernel.

Kernels & OpenCL Execution Model:-

* The OpenCL API enables an appl^n to create a context + describing the movement of data + the execution
* A serial C implementation, a threaded C implementation & a OpenCL implementation.

Eg:- ↓

C-Implementation

```
void vecadd(int *C, int *A, int *B, int len) {
    for(int i=0; i<len; i++) {
        C[i] = A[i] + B[i];
    }
}
```

Threaded C-Implementation

```
void vecadd(int *C, int *A, int *B, int N, int Np, int tid) {
    int vpt = N/Np      // Elements per thread
    for(int i= tid * vpd; i<(tid+1)*vpd; i++).
    {
        C[i] = A[i] + B[i];
    }
}
```

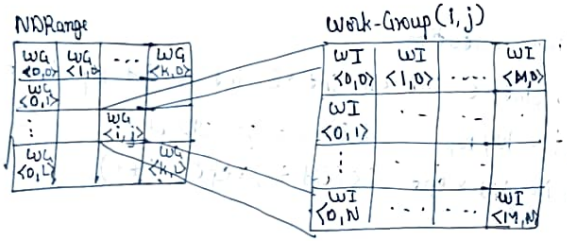
* The unit of concurrent execution in OpenCL is a work-item or WI
* Each work-item executes the kernel function body.
* We map single iteration to work-item

* The call to "get_global_id(0)" allows the programmer to make use of the position of the current work-item:

```
__kernel void vecadd (__global int *C, __global int *A,
                      __global int *B) {
    int tid = get_global_id(0);
    C[tid] = A[tid] + B[tid]
}
```

* Here parameter '0' represent x-direction, & similar to threadIdx.x.
* Work-items behave independently OpenCL allows the local workgroup size to be ignored by the programmer and generated automatically by implementation, i.e.

Work-item	Work-Group	NDRange
↓ III by	↓ III by	↓
thread	block	grid



* If the total no. of WI per array is 1024, this results in creating 16 work groups

[1024 work-items / (64 work-item per workgroup)] = 16 workgr

OpenCL Steps - Implementation :-

1) Discover & initialize the platform
2) Discover & initialize the devices
3) Create context
4) Create command queue
5) Create device buffers
6) Write host data device & buffers.
7) Create and compile the program
8) Create the kernel
9) Set the kernel arguments
10) Configure the work-items structure.
11) Enqueue the kernel for execution
12) Read the output buffer back to the host
13) Release OpenCL resources