CHAPTER 7

Parallel Processing

- There are two major motivations for creating and using parallel computer architectures. The first is to achieve vastly speed of computation than what current serial computers offer. The second motivation for the use of parallel architectures is that they should be cheaper than serial computers, for system of moderate speed. Thus, much better price performance could be obtained from parallel machines.

- A battery of satellite in outer are collecting data at the rate of (10^{10}) bits per second. The data represents information on earth's weather, pollution, agriculture, and natural resources. In order for this information to be used in a timely fashion, it needs to be processed at speed of at least (13^{13}) operations per second.

- Back on earth, at term of surgeons wish to view on a special display a reconstructed three-dimensional image of a patient's body in preparation for surgery. They need to be able to rotate the image at will, obtain a cross-sectional view of an organ, observe it in living detail, and then perform a simulated surgery while watching its effort, all without touching the patient a minimum processing speed of (10^{15}) operations per second would make this approach worthwhile.

- Unfortunately, a simple law in physics which is the speed of light puts an upper limit to the speed of electronic components this speed is approximately ($3*10^8$) meters per second now, assume that an electronic device can perform (10^{12}) operations per second. The benefits and gains in speed obtained by building fast electronic components are lost while each of these components is waiting for its input from another one.

- It appears that the only way around this problem is to use parallelism. The idea is that if several operations are performed simultaneously, then the time taken will be reduced to a fraction of the time taken in serial processing, thus parallel computers may satisfy efficiently the read of high speed for most sophisticated applications such as the previous two examples, by parallel computer we mean the computer that contains many processing and designed to receive a problem and breaks it into many smaller subproblems, then all these subproblems will be solved simultaneously each on a different processor, the results are then combined to get final or net result of the original problem.

- Another note to be mentioned is that parallel computers and sequential computers require special algorithms, programming languages, compilers, and operating systems to work an in order to get the needed performance of the system.

Historical view

- The first SIMD computer arised in 1958 to solve numerical problems.
- The first MIMD computer arised in 1962 using up to 4 CPU.
- illiac-iv which is the first processor array become available and operational in the middle of the 1970's, specially in 1975.
- In 1976, Gray-1 which is the first pipelining vector processor was put under operation.
- Later on VLSI enable the development of computing devices, that consists of thousands of transistors, hence, low cost multiprocessor become available.

Problems

The problem is, that as time goes on, the applications used get bigger and bigger, and were application which create a need to faster computers, this problem is solved by one of two ways:

1. One faster processor

   In uniprocessor system the used processor may be replaced by a faster one to increase computation speed, but this is not convenient, since sophistication of applications have no limit, but developing faster processor is limiting by allot of things, such as light speed.

2. Helping processors (parallelism)

   Adding processors to machine in order to cooperate with the existed processor acts as sending some workers to help one worker in transferring some objects from one place to another, instead of carrying one object at a time and transferring it to another place, then coming back to carry another object, and so on.

Parallelism as a device is divided into two parts

1. Hardware parallelism: This part is related to computer itself and to the components and processors it consists of.
2. Software parallelism: This part is related to the software or programs designed to be compatible with the computer (parallel computer) mentioned above.

Degree of parallelism

Since n processors can carry out up to n tasks per unit of time T then the number of tasks carries out is directly proportional to the number of processors. This number of processors is called **degree of parallelism**, (i.e. if there are k processors in a multiprocessors system then the degree of parallelism of this system = k), as a result the degree of parallelism increase the speed of computer increase.

Architectural Classifications

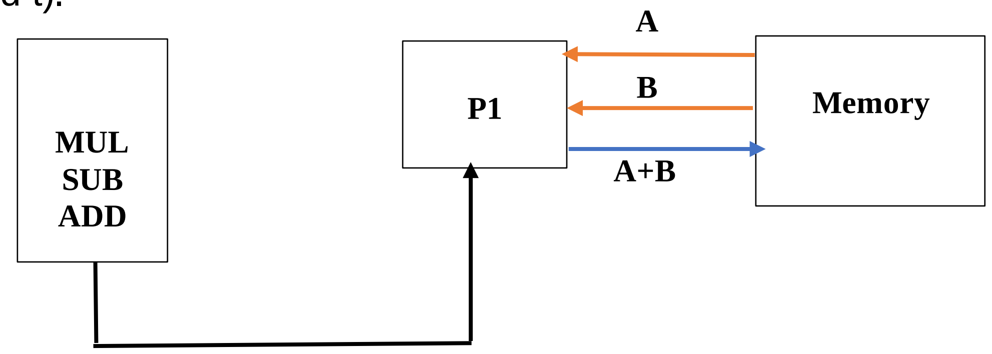
There are several different architectures for parallel computers. But mainly, these architectures can be divided into two parts:

1. Special-purpose parallel architectures are these architectures that are designed to solve a particular problem, but there disadvantage is that although they are the best used to solve this problem but they cannot be used for any other purpose or problem.
2. Multi-purpose parallel architectures, as there name indicates, are those designed for computers with a wide range of applications.

- Any computer weither parallel or sequential, operates by executing instructions on data. A stream of instructions (which is the algorithm) tells the computer what to do at each step. A stream of data (which is the input data) is affected and operated upon by these instructions, using the concepts of instructions and data streams and depending on weather there is one or several of these streams, all computer architectures can be classified into for categories:
    1. Single Instruction stream Single Data stream (SISD).
    2. Multiple Instruction stream Single Data stream (MISD).
    3. Single Instruction stream Multiple Data stream (SIMD).
    4. Multiple Instruction stream Multiple Data stream (MIMD).

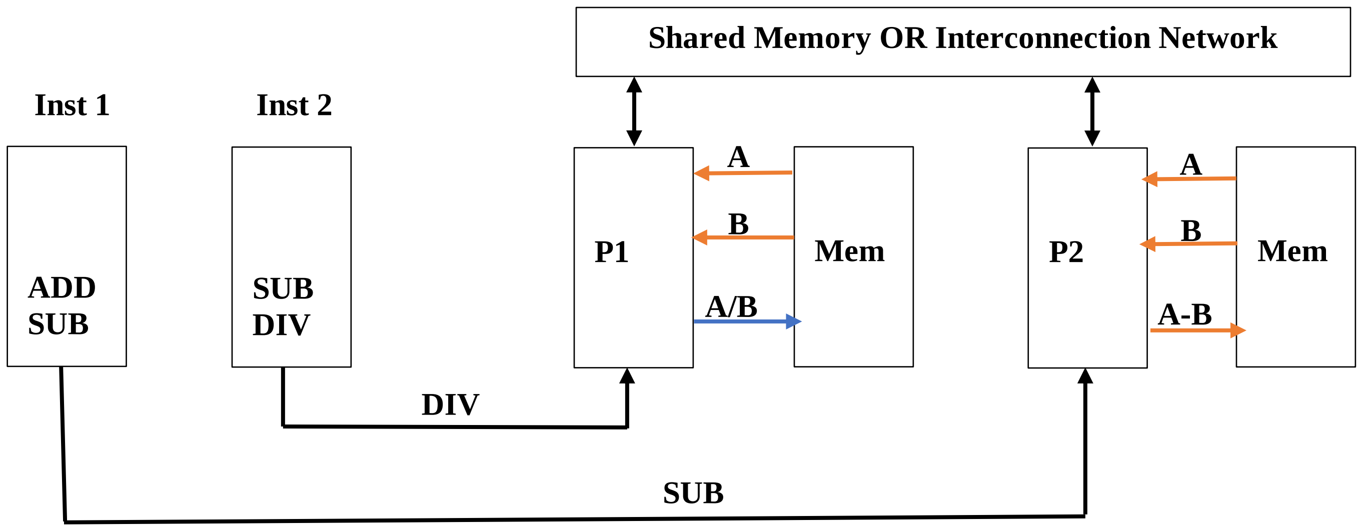
1. SISD

   This is the architecture of sequential computers, because a stream of instructions is applied to a stream of data sequentially (i.e. one instruction applied to one data set at any given time period t).



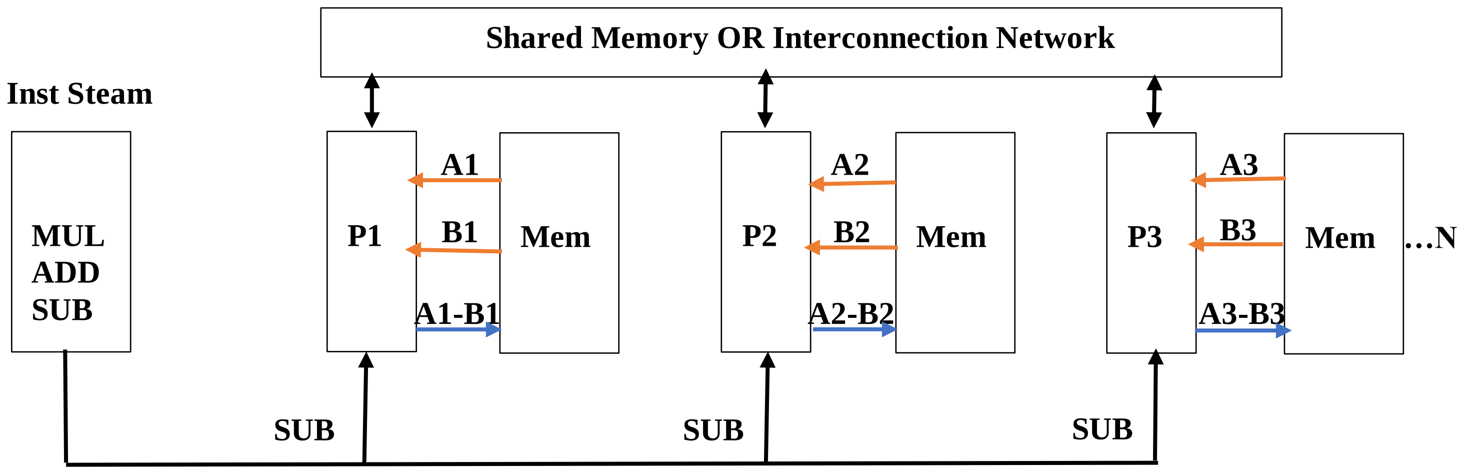
2. MISD

   This architecture is not used in computers since it is not needed for large application it is demonstrated by a simple example which is asking for the result of adding, multiplication, subtraction, and dividing the same two numbers, the four instructions are used upon the same data at the same time.



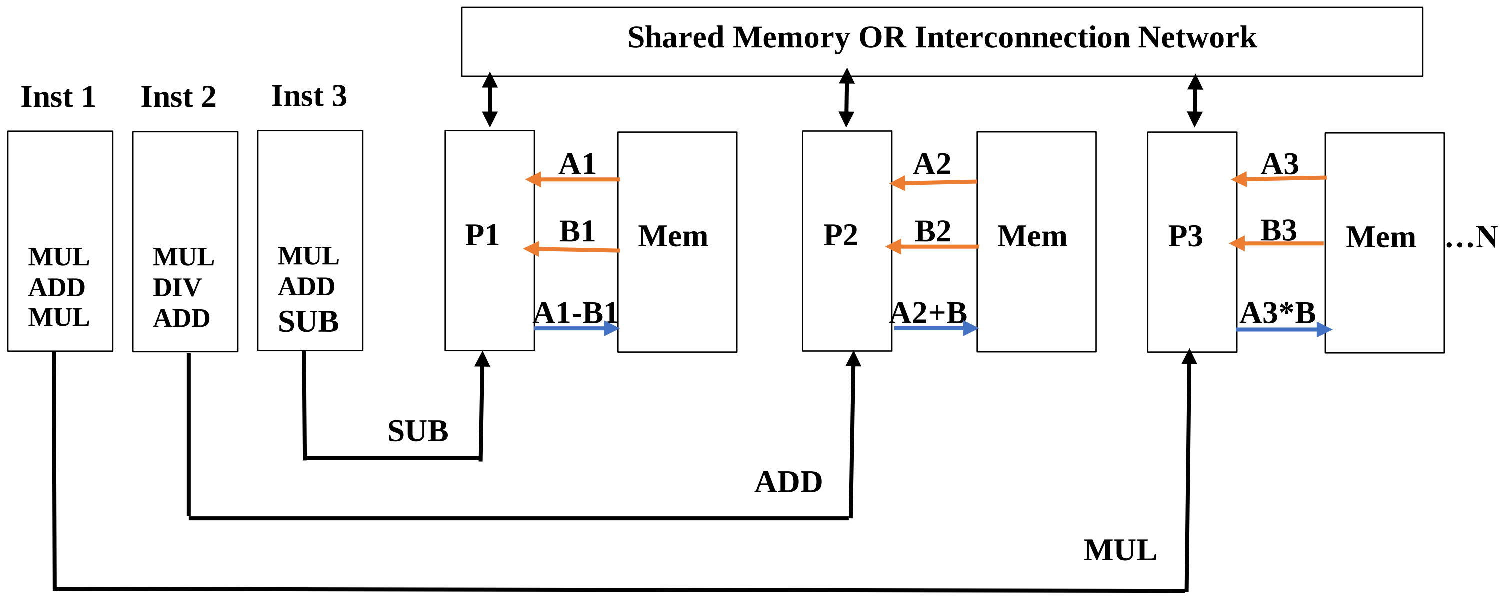
3. SIMD

   A SIMD computer consists of a number of processors operating under the control of a single instruction stream and several data streams, each processor have it's own small and private memory to store data in. During a given time unit T a selected number of processors N are active and execute the same instruction on N different data sets, each set is taken from a different data stream.



4. MIMD

   A MIMD computer work the same way as (SIMD) but (MIMD) uses several instruction streams instead of one, each processor executes an instruction taken from an instruction stream on a set of data taken from one of the data streams, these processors work synchronously and independently on each other.



Facilitating Parallelism

As mentioned before, speed of light puts an upper limit to the speed of electronic components in sequential computers., one processor was not sufficient solution even if it was faster than its precedent. Many architectural advanced more made the improve the system performance, the following are some of these advances:

1. **Bit parallel memory and Bit parallel Arithmetic**

   the early digital computers used bit serial main memory, where each bit which constituted a word was read serially (a bit at a time from memory). A cathode ray tube (CRT), was the first memory to allow parallel access to all bits in a word. The availability of this kind of memory subsequently lead to bit parallel arithmetic becoming possible.

2. **I/O Processors (channels)**

   In early computers CPU was also responsible for carrying out any I/O task, but this causes the CPU to loose a lot of its time since it should wait for the I/O device in order to complete its task before going back to execute the next instruction.

   This problem was solved by using I/O processors or as called (channels), an I/O processor is a processor that takes on its responsibility the performance of I/O tasks.

3. **Cache Memory**

   This a small, extremely fast memory used to speedup the rate of data retrieval by the processor, from the primary memory. This is accomplished by storing the program instructions and data that the processor id likely to require next, in the cache.

4. **Multiple Functional Units**

   A functional unit is a simple processing unit that can perform only one type of activities such as adder and multipliers, a number of these units can be used to perform more than one instruction in parallel. Since some next instruction to be executed can be fetched and placed in the instruction unit, a set of prefetched instructions can be checked to see which operations can be performed in parallel.

5. **Pipelining**

   Pipelining is a simple method to achieve parallelism in any task that could be divided into sub-activities, for example, suppose that a factory produces a product P by assembling it through several steps, and these steps are dependent on each others, which means that at step 1 performs process 1 on product P, then step 2 and after step 1 is finished performs process 2 on the same time P, . . . and so on. Until P is finished and then another item P2 is started with and the same steps are done to it.

   The problem here is loss of time, because if there are N steps in the assembly line, and the item P is at step k, where (k <= N), then the processors of steps (1…k) and (k_1 … N) will be idle until item P is completed.

Pipelining puts the solutions for this problem by performing the steps of the idle processors on a new item. By this way all the steps which are the sub-activities of the original task will be working in parallel and performing a lot of independent processes on the same amount of independent items, and idle processors will not exist anymore.

In computer architectures there are two kinds of pipelining, the first one is the Instruction Pipelining, which is dividing the instruction into different phases, such as instruction fetch, instruction decode, data fetch, execute, store result,…etc, in a second level of instruction dividing, instruction execute may be divided into smaller activities such as add, multiply, subtract,… etc., and all of them work in parallel. The second kind is data pipelining, in this technique, when some sets of data needs to be multiplied they are assigned to the multiplier as a stream of data or a vector of data. The same things happens with data that needs to be divided added, … etc.

6. **Multiprogramming, Timesharing, and Multiprocessing**

   Multiprogramming is the technique where by an operating system may allow multiple programs to be resident in the memory of the computer, and be simultaneously executed by the processors. This is done by the CPU switching to execute another program when the current program blocks for an input/output operation.

   Timesharing is a kind of multiprogramming where the operating system assigns time slices to each process (Program in execution). The CPU is switched to the next process, once the time slice of the current process runs out.

   Multiprocessing is the technique where multiple processors are used to achieve parallelism in processing.

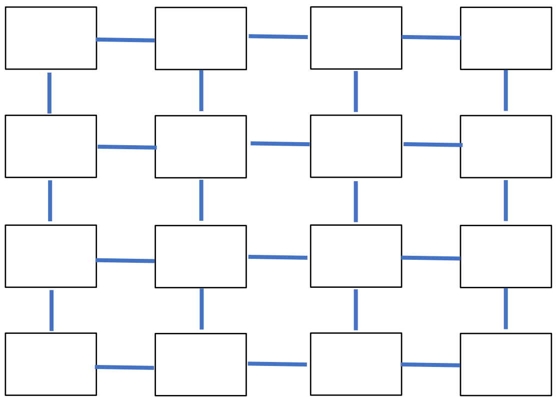
Parallel Processing Topologies

Parallel processing topology is the way by which the processor in a parallel computer are connected, some topologies are frequently used in parallel computers.

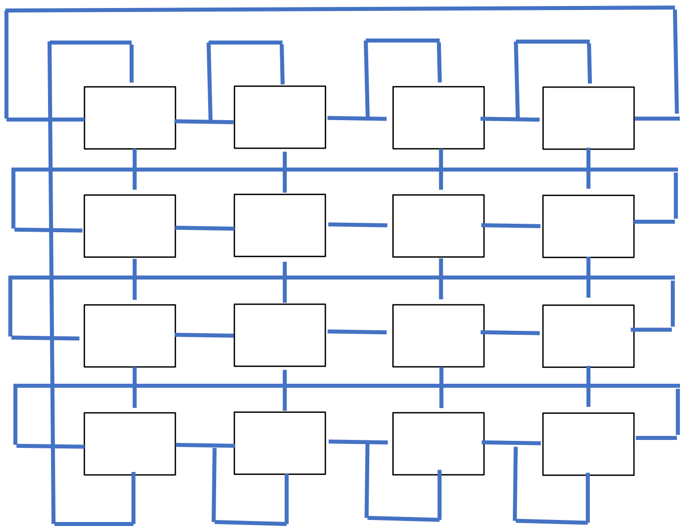
1. **Mesh connected**

   A linguistic of the word mesh is a lattice or a net like connection of nodes, but practically in this topology processors are ordered or arranged in on N dimensional lattice, where each processors is connected to its neighbor processors, which means that any inter process of the mesh is connected to a number of nodes = 2N, a special case of mesh is that case in which there exist wrap around connections between the first and last processor of each row or column.

   This topology is used mainly in sorting and matrix multiplication.



Without Wrap Around



With Wrap Around

2. **Pyramid connected**

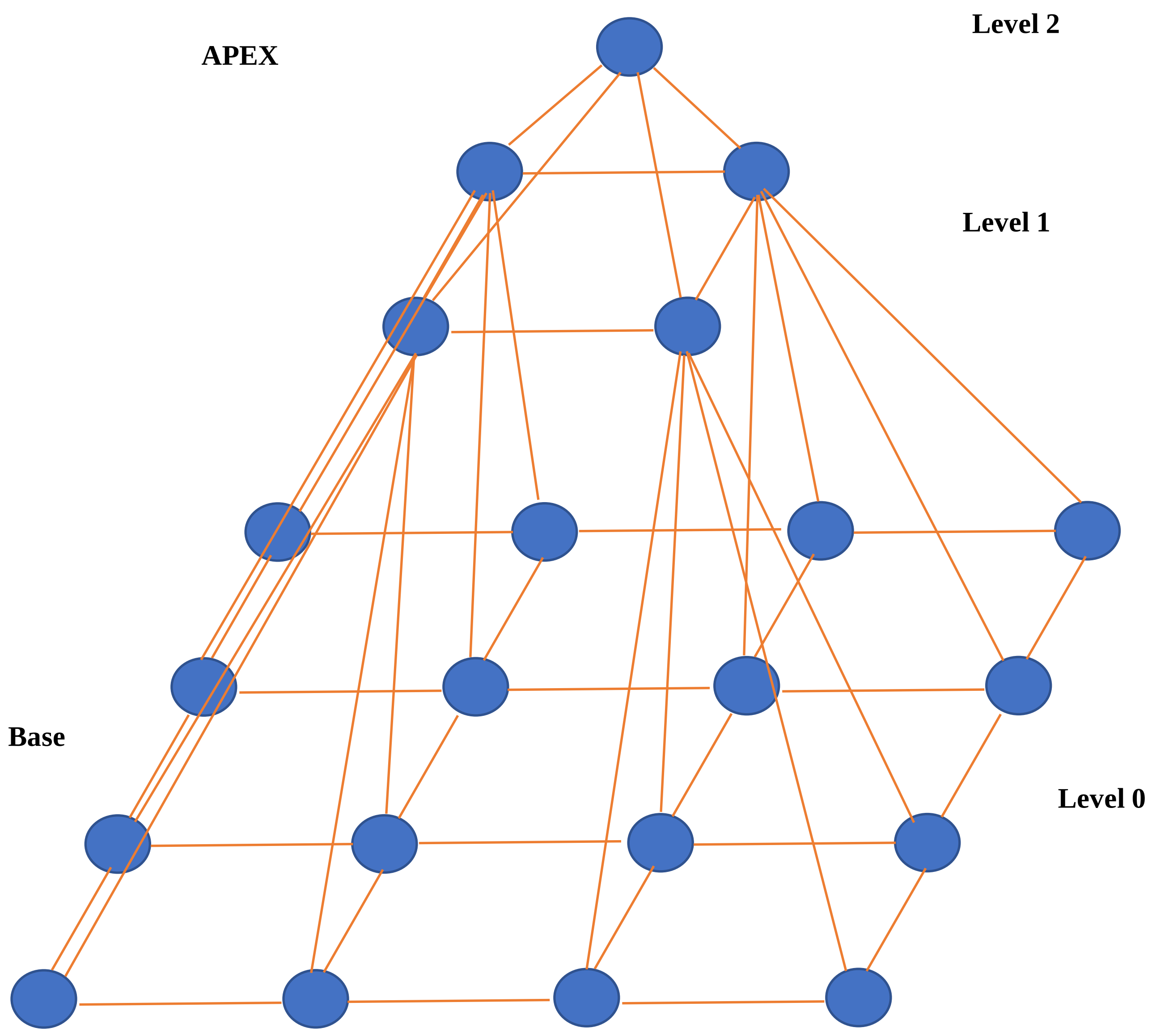
   In this topology and as it's name indicates processors are arranged in a way similar to a pyramid. In each level of this pyramid, each processor is connected to 4 children which are elements in the low lower level, there are 3 terms that are connected of in this topology

   - **APEX** which is the first parent node (processor), which is in the upper most level and has no parent as the root.
   - **Base** which is the lowest level in the pyramid.
   - **The size** of the pyramid network, which is the number of processors in the base level.

Also, it is important that if size = P, then number of processors in this pyramid equals (4/3 P – 1/3), these processors are distributed on the level of the pyramid that are number (1, 1, 2, …, $\log_4 P$), each inner processor is connected to (9) processors (1 parent, 4 neighbors, and 4 children's), processors in each level are connected according to mesh topology.

Example:

If the network **size = P = 16**, then the **base** level (level 0) contains 16 processors, and the **APEX** is in level 2 ($\text{lop}_4 P = \log_4 16 = 2$), and the number of processor = 21 (4/3*P – 1/3 = 4/3 * 16 – 1/3 = 21), number of processors in level 2 = 4/4 = 1 processor which is the APEX.


Level 2
APEX
Level 1
Base
Level 0

3. **Perfect Shuffle and Shuffle Exchange**

This topology is used only if the number of processors is a power of 2, as an example, let number of processors = $\mathbf{2^n}$, n = 1, 2, 3,…, in the perfect shuffle each processors (I) is connected to the processor (j) where

$$j = \begin{cases} 2*i & 0 \le i < \frac{n}{2} \\ 2*i + 1 - n & \frac{n}{2} \le i \le n \end{cases}$$

For example, suppose that we have $\mathbf{2^3}$ processors = 8, numbered 0, 1, 2, …, 7, then connections will be as follows

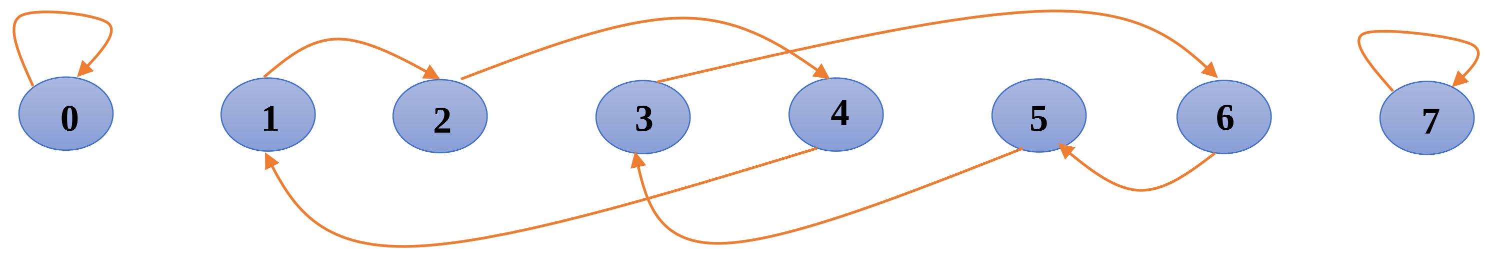
0 → 0	1 → 2	2 → 4	3 → 6
4 → 1	5 → 3	6 → 5	7 → 7

Another way for determining connections is by links node i with node **2*i % (n-1),** except node **n-1** which connects to itself.

Another way for determining connections is by using the bit representation of the processor's indices, let's take processor #3 as an example:

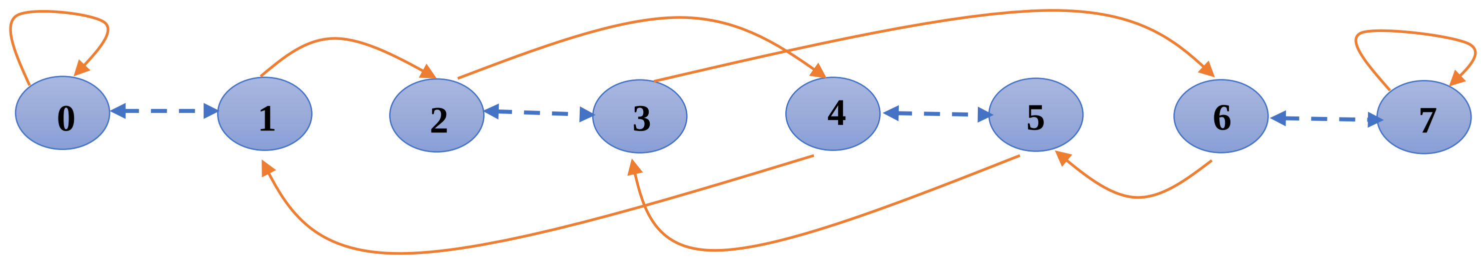
bit representation of number 3 = 011, the processor that processor #3 should be connected to is determined by cyclically shifting the binary representation (011) one position to the left so (011) will become (110) which equal 6 in decimal, so processor 3 should be connected to processor 6. All other connection are as follows:

000 $\rightarrow$ 000	001 $\rightarrow$ 010	010 $\rightarrow$ 100	011 $\rightarrow$ 110
100 $\rightarrow$ 001	101 $\rightarrow$ 011	110 $\rightarrow$ 101	111 $\rightarrow$ 111



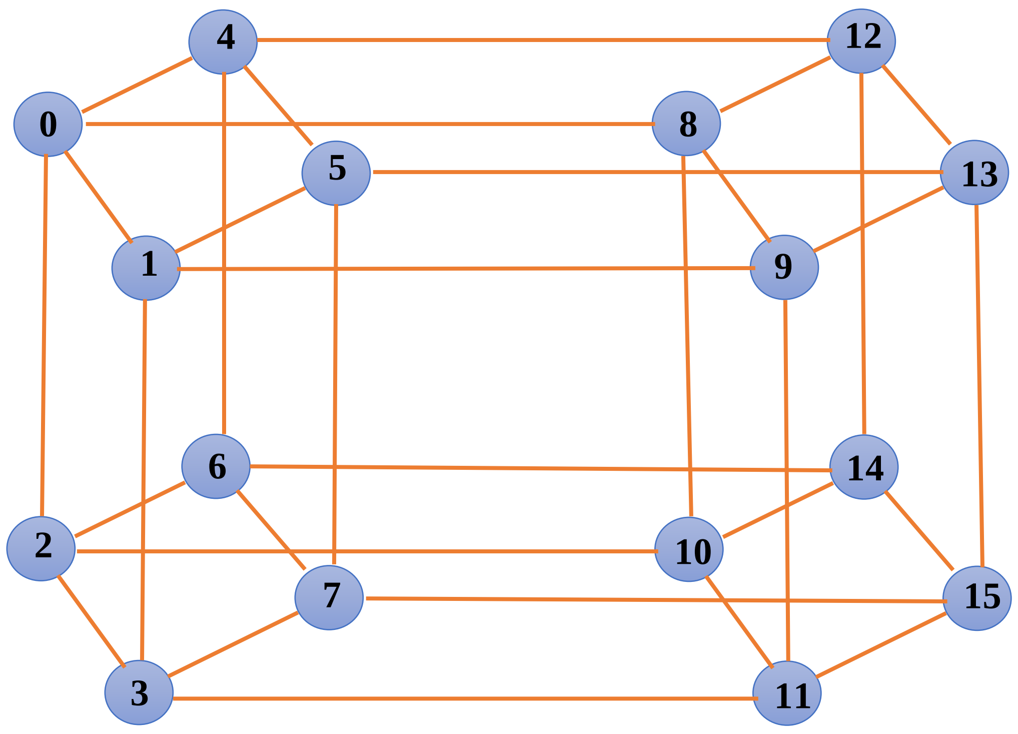
Shuffle-Exchange is the same as perfect-shuffle, but differs in one point which is an addition of a **two directional** connection in the shuffle-Exchange between each even numbered processor and it’s successor, these connections are called exchange connections, and this is where Shuffle-Exchange topology go it’s name from. Using **binary representation** makes it easier to determine exchange connections, this is done by connecting each processor to it’s successor only if this processor binary representation has a zero value in it’s least significant bit.
Another way for determining connections is by link pairs of processors whose numbers differ in their least significant bit.

One most important property of the perfect-shuffle topology is that the data returns back to its original processor after **m** mapping, where **m** is the number of bits used to represent the number of processors.



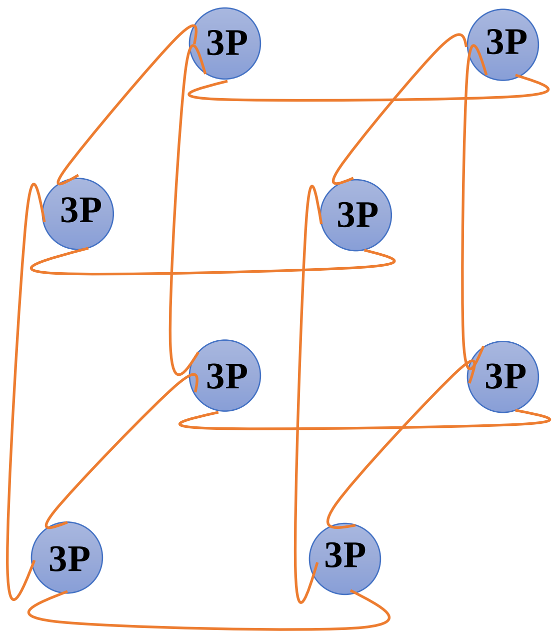
4. **Hyper Cube (Cube Connection)**

This topology involves arranging **n** processors where n = $\mathbf{2^k}$ in a k-dimensional lattice, these processors are also numbered 0, 1, 2, …, n-1, and every processor differ in it's number from it's adjacent processor by one, as in figure below that shown a 4-dimensional Hyper Cube with $\mathbf{2^4}$ = 16 processors.



5. Cube Connected Cycles (CCC)

This is same as Hyper Cube, but the difference is that in Cube Connected cycles each node of the k-dimensional Hyper Cube is a cycle of k processors, this property makes it possible to connect k * 2^k processors in a k-dimensional lattice instead of 2^k only as in the Hyper Cube, these cycles are shown in figure below where 24 processors are connected in a three dimensional lattice with 3 processor in each cycle: 3 * 2^3 = 24 processors, each processor which is connected cyclically within it's cycle is also connected to the processor of the same dimension in the adjacent cycle.



Evaluation Algorithm

- Running time and Speed-up

  These two measures may be more important than any other measure, because the main aim of parallelism and constructing parallel computers is to increase the speed of computations and reduce the whole time taken by the system to carry up the execution of the program. Two factors affect time consumed during execution. The first factor is data transfer for communication between the processors and memory, the second one is the time taken by a processor to execute an operation. Seep-up ratio (SP) is a measure that indicates how much parallel algorithms are better than sequential. SP is given below:

$$\text{Speed-up (SP)} = \frac{Runtime\ of\ fastest\ sequential\ algorithm}{Runtime\ of\ parallel\ algorithm}$$

- Speed-up is not applied to all sequential algorithms since not all of them is able to be parallelized, while any parallel algorithm can be implemented sequentially.

- Number of processors

  Number of processors is another measure for the evaluation of parallel algorithm, it is obvious that as the number of processors increase the cost of the solution needed to obtain increases.

  Speed-up can be also computed for a single processor amongst a set of processors, this speed-up is also known as processor efficiency, and this efficiency is different from the overall efficiency that will be given below, processor efficiency is given by the following equation:

Processor Efficiency (EP) = $\frac{SP}{P}$ = Speed-up Preprocessor

- Cost

  Since running time is defined by the time taken from the start of execution until the processor with the largest time taken finishes it's work, then the cost of the whole system may be given as the following product:

  Cost = parallel running time * number of processors used

  Hence,

  System overall efficiency = $\frac{\boldsymbol{Fastest\ Running\ time\ for\ sequential\ algorithm}}{\boldsymbol{Cost\ of\ parallel\ algorithm}}$

Bitonic Sort Algorithm using the SIMD (Hyper Cube) model

- Overview of Bitonic sort

  Bitonic sorting networks are based on the concept of merging pairs of sub lists or subsequences having the Bitonic property.

  The Bitonic property as follows:

  A sequence (or list) $\{x_1, x_2, \ldots, x_{2n}\}$ is said to be Bitonic if either

  1. There exists an index j, $1 \le j \le 2n$, such that

     $x_1 \le x_2 \le x_3 \le \ldots \le x_j \ge x_{j+1} \ge x_{j+2} \ge \ldots \ge x_{2n}$ OR

  2. There exists a cyclic shift of indices for which condition (1) is satisfied.

- Consider the Bitonic sequence $\{ x_1, x_2, \ldots, x_{2n} \}$ to be Bitonic sequence of size **2n,** where n=2 power m, for some integer $m \ge 0$. Suppose we assume that the length of the ascending portion of the sequence is equal to be length of the descending portion, then we can write $x_1 \le x_2 \le \ldots \le x_n$ and $x_{n+1} \ge x_{n+2} \ge \ldots \ge x_{2n}$.

  Now we form two new sequences by pairing off corresponding elements in the two-given sequence, as follows:

  a. $\min(x_1, x_{n+1}), \min(x_2, x_{n+2}), \ldots , \min(x_n, x_{2n})$ and
  b. $\max(x_1, x_{n+1}), \max(x_2, x_{n+2}), \ldots , \max(x_n, x_{2n})$

  It can be seen that both these are Bitonic of size n. Further, each element in the first sequence is smaller than every element in the second sequence

$$\text{Max}\{ \min(x_i, x_{i+m}) / i = 1, 2,\ldots,n \} \le \text{Min}\{ \max(x_i, x_{i+m}) / i = 1, 2, \ldots,n\}$$

These properties indicate that a Bitonic sequence {x_{1}, x_{2}, . . ., x_{2n}} can be sorted using the following steps:

1. A Bitonic sequence of size 2n is transformed into two Bitonic sequences of size n, in a single compare exchange steps, using n processing units.
2. Since both these subsequences are Bitonic, they can be recursively sorted using a network which sorts Bitonic sequences of order **n**. The **n** smallest elements of a sorted sequence of length **2n** will be generated by one of these networks, and the **n** largest elements by another.

- Let us consider how this sorting algorithm could be implemented. From the sequences (a) and (b), it can be determined that Bitonic sort always compares elements whose indices differ in exactly one bit. Since the processors in the SIMD-CC model (Hyper Cube) are connected such that processors where indices differ in one bit are adjacent, this model is suitable for implementing this algorithm.

  The processors compare and exchange data amongst themselves.
- Suppose **n** elements are to be sorted where n = 2^{m}. Then we use n = 2^{m} processors, where each element is assigned to a different processor. Hence, we require an m-dimension Hype Cube. Pairs of adjacent processors perform compare-exchange operations between themselves. Thus, there are m(m+1) such processing steps, where each step constitutes m/2 (2^{k-1}) simultaneous

compare-exchange operations, between pairs of n/2 adjacent processors. This process distributes elements in the processors, in ascending order of their indices.

Minimum value in processor 0, Maximum value in processor n-1

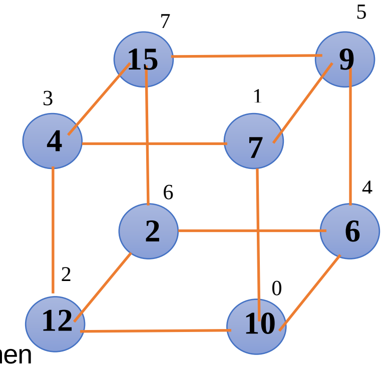
Algorithm for Bitonic sort using SIMD-CC model

```
BitonicMergeSort ( SIMD-CC )
Begin
    for i = 0 to m-1 Do
        for j = i down to 0 Do
            d = $2^j$
            for all $p_k$ ,where  $0 \le k \le 2^m - 1$ Do
                if ( k Mod 2d < d ) then
                    // get value from adjacent processor
                    $t_k = a_{k+d}$
                    if ( k Mod $2^{i+2} < 2^{i+1}$ ) then
                        $b_k = \max(t_k, a_k)$
                        $a_k = \min(t_k, a_k)$
                    else
                        $b_k = \min(t_k, a_k)$
                        $a_k = \max(t_k, a_k)$
                    end if
                end if
            end for
        end for
    end for
end
```

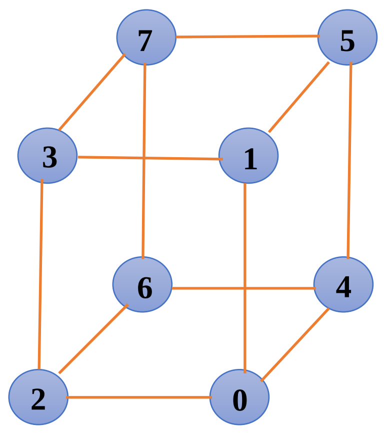
Consider the list {10, 7, 12, 4, 6, 9, 2, 15}. Suppose these are distributed amongst 8 processors. We would require 3-dimension (2^3 = 8 processors) hyper cube. According to the algorithm, the following 6 (= 3(3+1)/2) steps would be performed, with each step consisting of 8/2 (= 2^{3-1}) simultaneous compare-exchange operations between pairs of adjacent processors.

b_k	a_k		i	j	d		t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
P_{k+1}	P_k		0	0	2^0=1		7	X	4	X	9	X	15	X
P_{k+2}	P_k		1	1	2^1=2		12	4	X	X	2	15	X	X
P_{k+1}	P_k			0	2^0=1		4	X	10	X	9	X	2	X
P_{k+4}	P_k		2	2	2^2=4		9	5	15	2	X	X	X	X
P_{k+2}	P_k			1	2^1=2		6	2	X	X	10	12	X	X
P_{k+1}	P_k			0	2^0=1		2	X	7	X	9	X	12	X

```
BitonicMergeSort ( SIMD-CC )
Begin
        for i = 0 to m-1 Do
                for j = i down to 0 Do
                        $d = 2^j$
                        for all $p_k$ ,where  $0 \le k \le 2^m - 1$ Do
                                if ( k Mod 2d < d ) then
                                        $t_k = a_{k+d}$
                                        if ( k Mod $2^{i+2} < 2^{i+1}$ ) then
                                                $b_k = \max(t_k, a_k)$
                                                $a_k = \min(t_k, a_k)$
                                        else
                                                $b_k = \min(t_k, a_k)$
                                                $a_k = \max(t_k, a_k)$
                                        end if
                                end if
                        end for
                end for
        end for
end
```

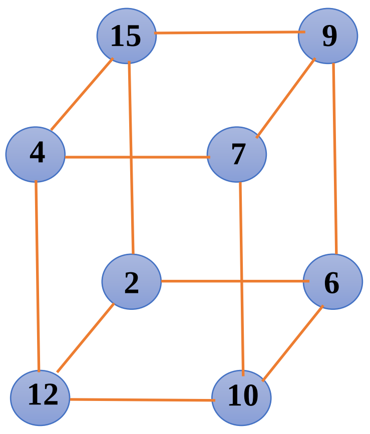


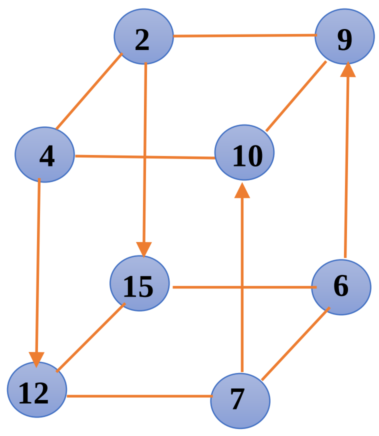
b_k	a_k		i	j	d		t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
P_{k+1}	P_k		0	0	$2^0=1$		7	X	4	X	9	X	15	X
P_{k+2}	P_k		1	1	$2^1=2$		12	4	X	X	2	15	X	X
P_{k+1}	P_k			0	$2^0=1$		4	X	10	X	9	X	2	X
P_{k+4}	P_k		2	2	$2^2=4$		9	5	15	2	X	X	X	X
P_{k+2}	P_k			1	$2^1=2$		6	2	X	X	10	12	X	X
P_{k+1}	P_k			0	$2^0=1$		2	X	7	X	9	X	12	X

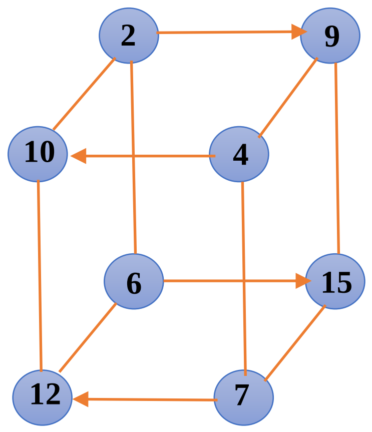


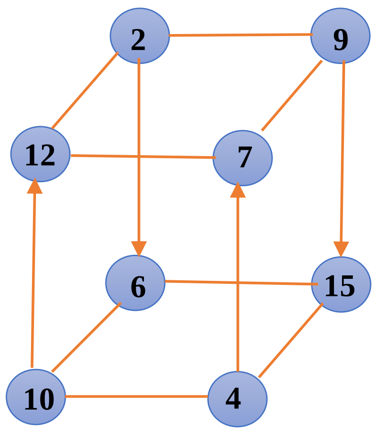
Let c =

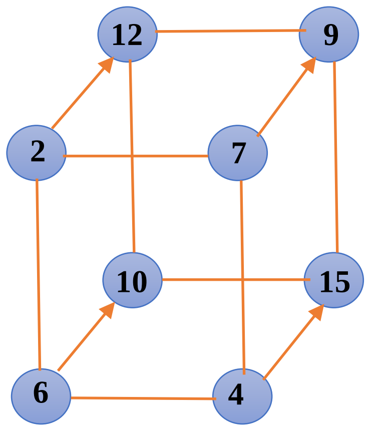
{ 10, 7, 12, 4, 6, 9, 2, 15 }

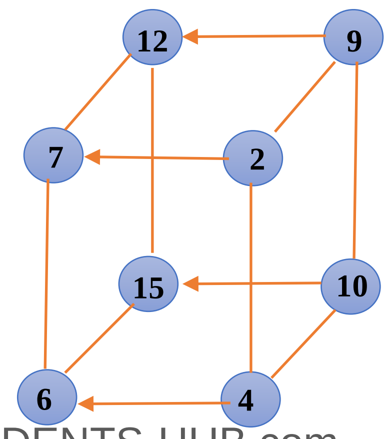


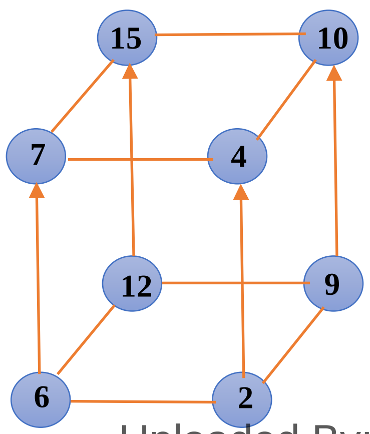










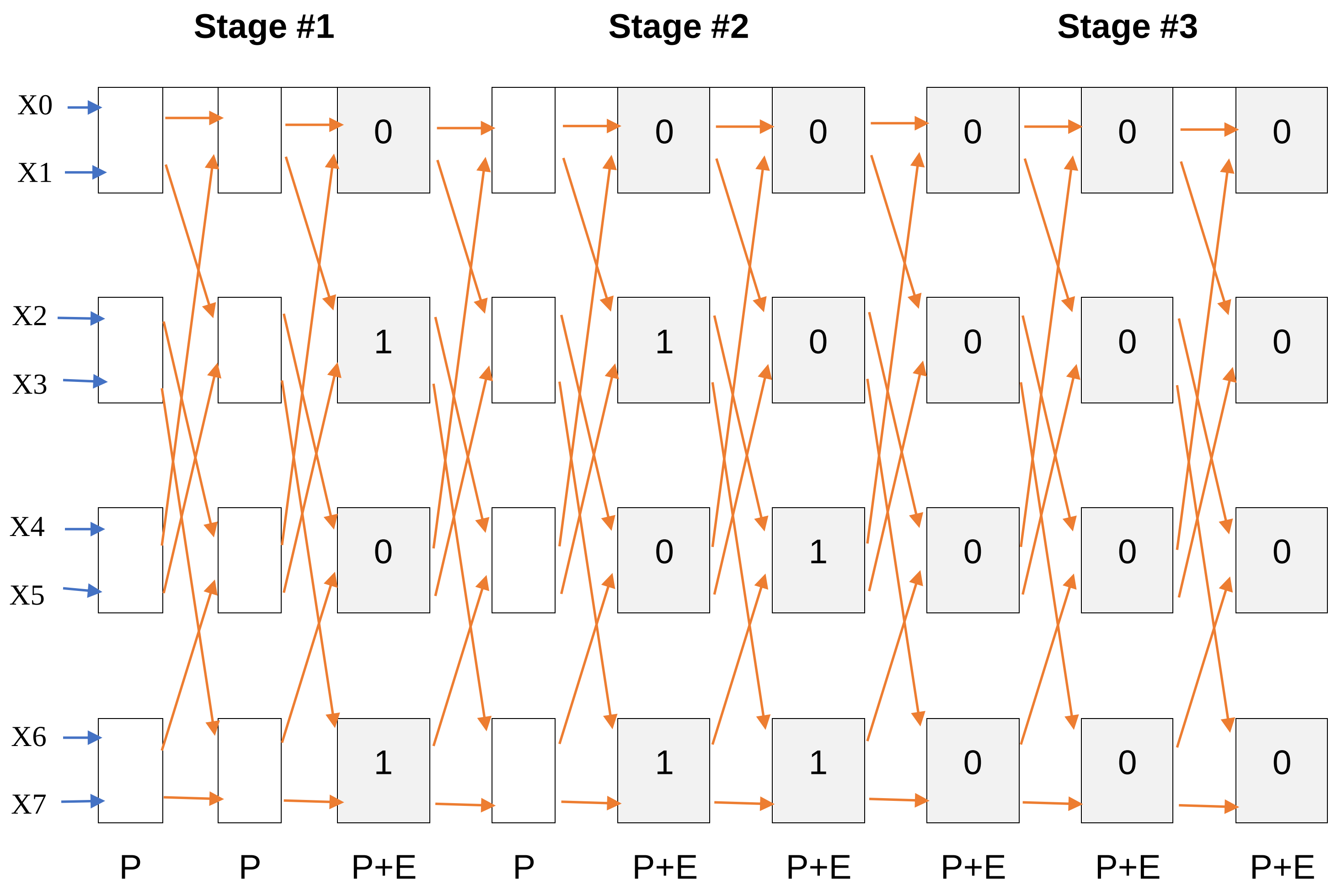


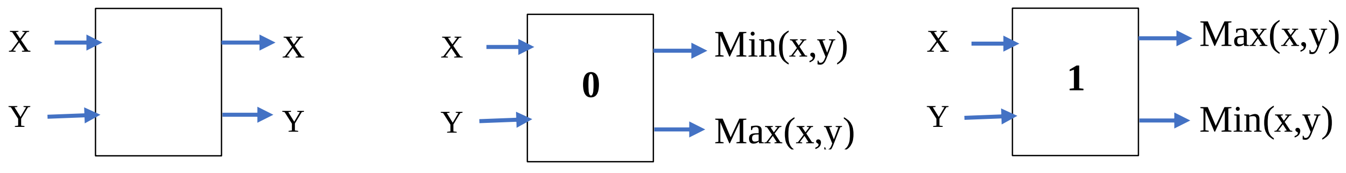
Bitonic Sort Algorithm using the Perfect-Shuffle model

- This technique sorts n elements using n/2 processors, and exploits the exchange and shuffle connections inherent in the PS network. Each processor stores two values and has to sort them in either ascending or descending order. It makes this decision based upon a value stored in the MASK vector. If the value stored in the MASK vector corresponding to the processor is 0, the number are stored in ascending order, and if 1 in descending order. This iteration is performed a maximum of $\log^2 n$ times, during which time the data is subjected to shuffle and exchange operations. Eventually, the data will be found to be stored in the required order.

Algorithm for Bitonic sort using perfect shuffle modal

```
BitonicMergeSort ( SIMD-Perfect_Shuffle)
    V = vector ( 0, 1, 0, 1, 0, 1, …, 0, 1)
    MASK = V
    for  i = 1 to m Do
        MASK = MASK (XOR) V
        shuffle (MASK)
    end for
    compare-exchange( L )
    for i = 0 to m-1 Do
        shuffle (V)
        MASK = MASK (XOR) V
        for  j = 1 to m-1-i Do
            shuffle (L)
        end for
        for j = m-I to m Do
            shuffle (L)
            compare-exchange (L)
        end for
    end for
end.
```


Stage #1
Stage #2
Stage #3
X0
X1
X2
X3
X4
X5
X6
X7
0
1
0
1
0
1
0
1
0
0
1
1
0
0
0
0
0
0
0
0
0
0
0
0
P
P
P+E
P
P+E
P+E
P+E
P+E
P+E


X
Y
X
Y
X
Y
0
Min(x,y)
Max(x,y)
X
Y
1
Max(x,y)
Min(x,y)

Sorting the list { 3, 9, 2, 6, 1, 0, 4, 8 }

P : Perfect Shuffle

P+E : Perfect Shuffle followed by comparison exchange

index	Input	P	P	P	E		P	P	E	P	E		P	E	P	E	P	E
0	3	3	3	3	3		3	3	3	3	2		2	2	2	1	1	0
1	9	1	2	9	9		0	6	6	2	3		8	8	1	2	0	1
2	2	9	1	2	6		9	0	8	6	6		3	3	8	6	2	2
3	6	0	4	6	2		1	8	0	9	9		4	4	6	8	3	3
4	1	2	9	1	0		6	9	2	8	8		6	1	3	0	6	4
5	0	4	6	0	1		8	2	9	4	4		1	6	0	3	4	6
6	4	6	0	4	8		2	1	4	0	1		9	0	4	4	8	8
7	8	8	8	8	4		4	4	1	1	0		0	9	9	9	9	9
		Stage #1					Stage #2						Stage #3					