7.2 Cache Coherence & Synchronization Mechanisms

(Reference: KAI HWANG)

What is Cache Coherence problem?

In a multiprocessor system, there will be multiple levels in the memory hierarchy system classified as local memory, shared memory and remote memory systems. When multiple levels contain inconsistent copies of same data then it is called as **Cache Coherence Problem.**

To overcome the inconsistency within shared memory system and local processor, two different cache systems are used:

1. **Write through cache:** Whenever a local copy data is modified, the modification is immediately reflected in the shared memory system also.

2. Write back cache: Whenever a local copy data is modified, modification in the shared memory is done only at the time data object is replaced back into memory system.

Causes for cache consistency (Sources for cache inconsistency)

There are three sources for the cache inconsistency:

1. **Sharing of the writable data**

2. Process migration

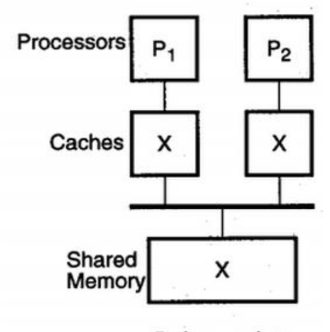
3. I/O activity

We will discuss all the techniques in detail

1. Inconsistency in Data sharing:

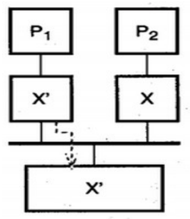
One of the situation were inconsistency in the data occurs when more than one cache shares the same data object and modification is done by the remote processor on the same data object.

→To illustrate this consider the scenario of two processor P1 and P2 each sharing a consistent copy of data item X in its cache memory as shown below:



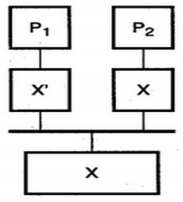
Before update

→ Assume that a **write through cache** system is used and consider that P1 modifies data object **X** to **X**$^{|}$. In this case object in the shared memory is made consistent by the immediate updating. But data object in the remote processor remains inconsistent without any modification. The scenario is shown below:



Write-through

→In the next case assume that **write back cache** is used consider that P1 modifies data object **X** to **X**$^{|}$.In this case object in shared memory also becomes inconsistent until it is replaced back and even in remote processor also it will be inconsistent. The scenario is shown below:



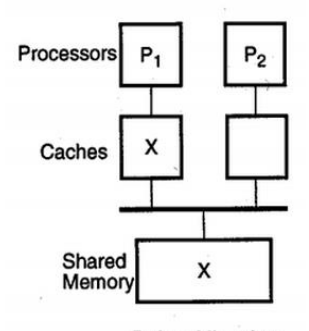
Write-back

Conclusion: when write through is used inconsistency occurs only in one level. But when write back is used inconsistency occurs in two levels.

Inconsistency due to process migration

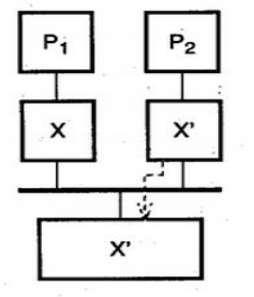
When data from one processor is moved into another processor, then the concept is called as **Process migration.** To understand it in a simple way, it's a inter process communication happening in the system.

→ To illustrate this consider the scenario of two processor P1 and P2, P1 sharing a consistent copy of data item X in its cache memory before the data migration as shown below:



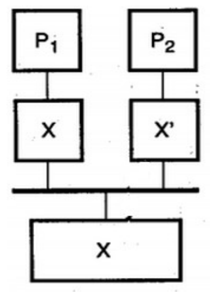
Before Migration

→Next assume that a **write through** cache system is used and processor P1 migrates data object X into processor P2. When processor P2 modifies data from **X** to **X$^|$.** Modified data will be immediately updated in shared memory system and data remains inconsistent only in the processor which has migrated the data. The situation is shown below:



Write-through

→ Next assume that a **write back** cache system is used and processor P1 migrates data object X into processor P2. When processor P2 modifies data from **X** to **X$^|$.** Data remains inconsistent in both the levels of remote processor and shared memory systems. The situation is shown below:

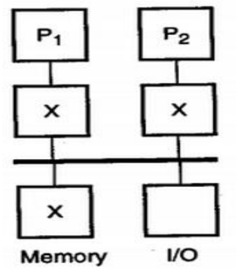


Write-back

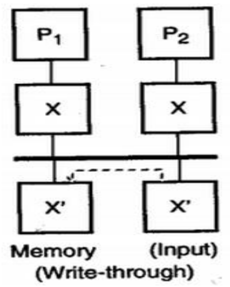
3. Inconsistency caused by I/O activity

This inconsistency occurs when I/O operations bypass the local caches while doing any kind of I/O activity.

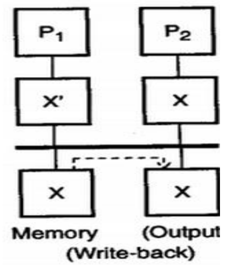
→ To illustrate this consider the scenario of two processor P1 and P2 each sharing a consistent copy of data item X in its cache memory as shown below:



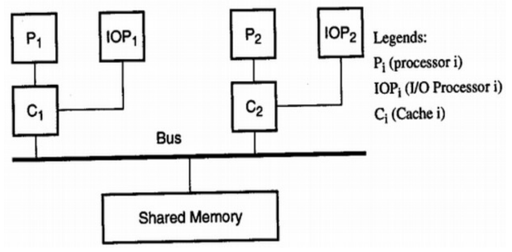
→Next assume that **write through cache** is used and there is an input activity done on the system which directly modifies the data object **X** to **X$^|$** in the shared memory bypassing the processor cache. In that case inconsistency will be caused at the local processor which already shared the data.



→In another case assume that **write back** cache system is used were a data is modified from **X** to **X**$^{|}$ by processor P1 and if data object X is sent to output before the replacement it will display inconsistent data copy bypassing the cache system as shown in the diagram below:



There is a very simple solution to overcome the above problem, were in separate I/O processors is used for each and every cache. Where in cache system is involved for every I/O activity as shown in the figure:



Cache coherence protocols

To overcome the problem of data inconsistency in the remote processors there are methods developed which are called as cache coherence protocols. There are two protocols:

1. **Snoopy bus protocol:** The protocol is used when a processor and memories in a multiprocessor system are connected through a single shared bus.

2. Directory scheme protocol: The protocol is used when a network and switch system is used to connect multiprocessor system were in a special directory is used to maintain the modifications done for the data object.

We will be discussing only on snoopy bus protocols.

Snoopy Bus Protocols:

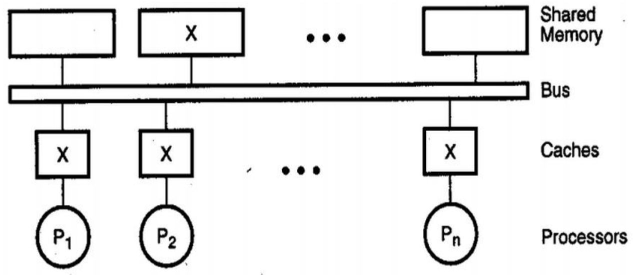
When a bus is used to connect processor and memories in the multiprocessor system, each cache controller present in the cache system can snoop (investigate) on all bus transactions. If any of the bus transactions affects the consistency of locally cached object, then it takes certain actions to maintain the data consistency.

Necessary actions can be taken in two approaches:

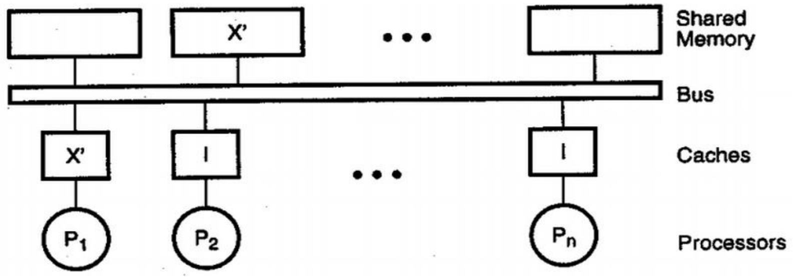
1. **Write invalidate:** In this approach, when a local cached copy is modified, invalidate all remote copies shared.

The approach can be illustrated as shown below for the write through cache system:

In the initial state, all the local processor in the system consists of shared data object **X**

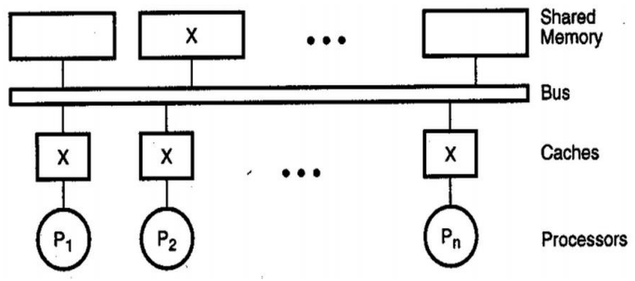


→Next assume that P1 modifies the data to X$^{|}$ and the data in all other caches is made invalid as shown in the next figure:

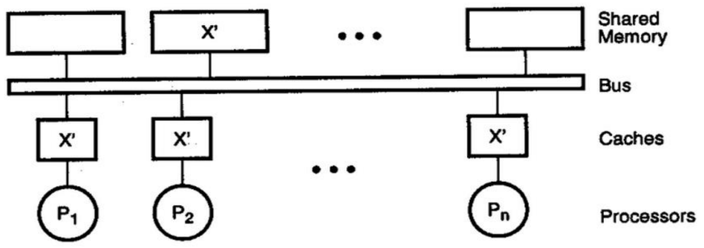


2. Write update: When the local copy is modified, broadcast modified value of data to all the caches over a shared bus system.

→Consider the initial state sharing the consistent copy of data object X in all cache system as shown below:



→Next assume that processor P1 modifies the data and data is broadcasted to all the caches as shown below:



Implementation of write invalidate protocol in write through caches

Here we will see how exactly write invalidate protocol works for a data objects of a particular cache block.

→We will consider **i** as our local processor and any transactions done by that will be represented in terms of i. All other processor connected over a system will be the remote processor and is represented by **j** and any transactions done by any remote processor on a local data copy is represented in terms of j.

→Both local and remote processors can do three different transactions on a local data copy : Read (R), Write(W) and Replace(Z).

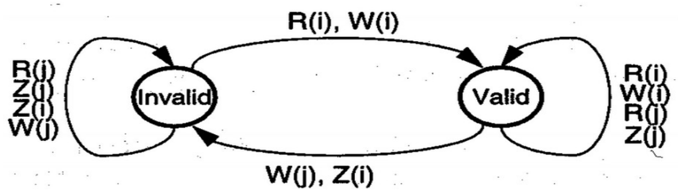
→Local transactions are represented by : R(i), W(i) and Z(i)

→Remote transactions are represented by: R(j), W(j) and Z(j)

→Any local copy of data can have two states **valid** and **invalid.**

→We will see how exactly works according to state transition diagram:

Note: Replacement of data by local processor leads to invalid state.



→Transitions from locally valid to locally invalid occur as a result of a remote processor write or a local processor replacing the cache block.

→Transitions from locally invalid to locally valid occur as a result of the local processor reading or writing the object (necessitating, of course, the fetch of a consistent copy from shared memory).

Implementation of write invalidate protocol in write back caches

When it comes to a write invalidate in write back cache system, a small difference is there. In write through data will be updated in shared memory system immediately and any time if any processor finds its local copy as invalid, it can directly contact shared memory system to fetch the consistent copy of the data.

But in write back, consistent data will be sent back only when it is being replaced and till then it will kept within some local cache. Because of the protocol implements a ownership technique, which tells that a particular processor is owner for the modified data object.

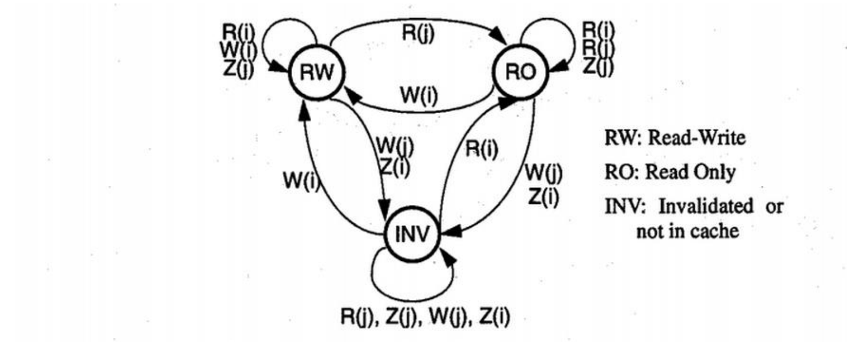
In this protocol, totally there are three states out of which two represent valid state and only one represents the invalid state, the states are:

1. **Read Only (RO):** Represents the valid state and when the local copy is in this state, it means that data copy is being shared by more than one processor. In this state reading by any processor can be done. It also does the job of broadcasting whenever a block takes the ownership.

2. **Read Write (RW):** Represents the valid state and when the local copy is in this state it means that it is the owner of the copy and only this block has the consistent copy of the data. In this scenario all other processors has to contact local cache for the consistent copy of the data. In this state local processor can do any transactions. Before writing the data, local processor has to enter this state to take the ownership.

3. **Invalid (INV):** Only state representing the invalidation of data and is same as the one discussed in previous section.

State transition for this system is shown below:



W(i) = Write to block by processor *i*.	W(j) = Write to block copy in cache *j* by processor *j*
R(i) = Read block by processor *i*.	R(j) = Read block copy in cache *j* by processor *j*
Z(i) = Replace block in cache *i*.	Z(j) = Replace block copy in cache *j*

Three Generations of Multicomputers (Short notes question)

First Generation:

→Caltech's Cosmic Cube was the first of first generation multicomputers.

→Typical system comprised of 64 nodes each with 512K bytes of memory per node.

→Each node is implemented on printed circuit board with 8 I/O ports with eight port is meant for Ethernet connection.

→Communication was done by the method of **store and forward** technique which had a capacity of transferring 100 byte message in a block.

→Latency for the local node communication was far lesser compared to latency with non local nodes.

Second Generation:

→Multicomputer system still in the present usage.

→Major improvement includes the usage of much better processors for the node and memory per node was also increased by 10 times.

→Typical system comprised of 256 nodes.

→Communication was implemented through the introduction of hardware supported routing such as **wormhole routing.** Mesh Routing Chips (MRCs) are used to establish four neighbor connections.

→Communication latency for both remote and local memory systems is same.

Third Generation

→Nodes were implemented through VLSI technology.

→Typical system comprised of 1024 nodes with improved memory capacity.

→Communication latency was reduced to 0.5 ns which is very less compared to other two generations.

→Research is still underway on this multicomputer system.

Data Flow Architecture (Reference: 2.3 & 9.8 Kai Hwang)

Program Flow Mechanisms

For the execution of the instructions, basically there are two program flow mechanisms:

1. Control flow mechanism
2. Data flow mechanism.

1. Control Flow Computers

→In this method, order of program execution is explicitly stated in the user programs. The type of computers belong to this category are called as **Control flow computers.**

→These type of computers use Program Counter (PC) to sequence execution of the instructions in the program.

→Use shared memory systems to hold data objects and program instructions. Variables in shared memory system are updated by many instructions.

2. Data Flow Computers

→In this computer system, execution of instruction is driven by data availability. I.e. instruction should be ready for the execution whenever the operands are available.

→Instead of storing data in a shared memory, it is directly held within the instructions.

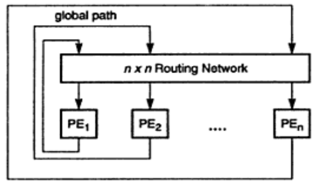
→Computation results which are called as **data tokens** are directly passed between the instructions. Data generated by the instructions will be duplicated into many copies and sent into all needy instructions. Data tokens once consumed will be no longer available for the usage.

→There is no shared memory, no program counter and no sequencer in the architecture. There has to be a mechanism to enable the data token.

Dataflow Architecture

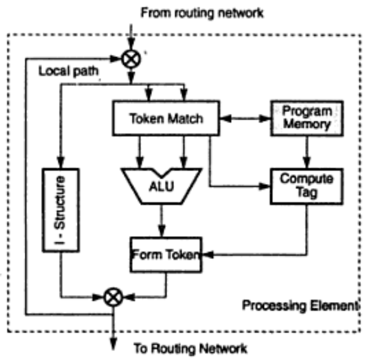
The architecture is a tagged token architecture for building data flow computers were every data that is manipulated in the architecture is tagged with the name of the instruction that has processed the result and the name of the destination register.

Below diagram shows the overview of the architecture



(a) The global architecture

Architecture has n processing elements interconnected by nxn routing network. The system supports pipelined dataflow operations in all the processing elements. The internal architecture of each of the processing element is shown below:



(b) Interior design of a processing element

The architecture has following units:

1. **Token match**: Unit provides a low level token matching mechanism which dispatches those instructions whose input data tokens are available.
2. **Program memory:** Memory used to store the instructions.
3. **I Structure:** It is used for the overlapped usage of the data structure which stores 2 bit information if the word is empty, full or has pending requests.

Functioning:

Data that is processed will enter all the processing elements through the routing network. Once after entering the routing network, token match unit will check if there is a match between the required data operand and processed data if there us a match, then dispatches the data, executes data in the ALU unit and forms the token for the result and again broadcasts the result through routing network.

Data flow mechanism using a simple Data Flow Graph

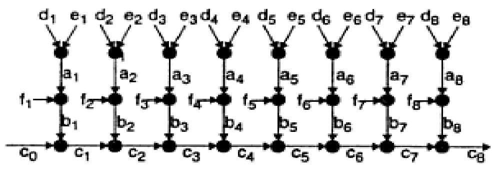
Data flow graphs can be used as machine level languages for the data flow computers. These graphs will give the clear picture flow of execution taking place in a data flow computers.

Data flow graphs are made of nodes and labeled arrows. Where each node represents functional unit performing the operation and labeled arrows represents the data operands operated by the functional unit.

To get the clear picture of the Data flow graph representation, consider the following instructions:

```
input d, e, f
  $c_0 = 0$
for i from 1 to 8 do
  begin
    $a_i := d_i + e_i$
    $b_i := a_i * f_i$
    $c_i := b_i + c_{i-1}$
  end
output a, b, c
```

Data flow graph for the code mentioned above

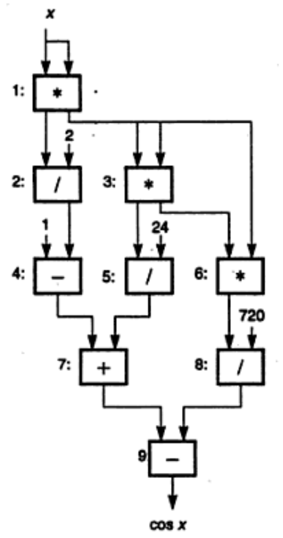

d1 e1 d2 e2 d3 e3 d4 e4 d5 e5 d6 e6 d7 e7 d8 e8
a1 a2 a3 a4 a5 a6 a7 a8
f1 f2 f3 f4 f5 f6 f7 f8
b1 b2 b3 b4 b5 b6 b7 b8
c0 c1 c2 c3 c4 c5 c6 c7 c8

Above shown Data flow graph is one type of representation, were in the representation is basically done for loop kind of instructions.

Consider another example of instructions which shows the representation of data flow graph in another type

This dataflow graph shows how to obtain an approximation of cos x by the following power series computation:

$$\cos x \simeq 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}$$



System Interconnect Architectures (Reference: 2.4 Kai Hwang)

System Interconnect architectures are the one that is used to carry data between processor and memory systems. There are two ways of interconnecting multiprocessor and multicomputer systems. They are:

1. Static Networks
2. Dynamic Networks

Static Connection Network

→These are also called as Direct Network. The network connections are formed of point-to-point direct connections which are fixed and will not change during the execution. There will be a fixed link between every connection.

→Are best used for the system were communication pattern are almost fixed.

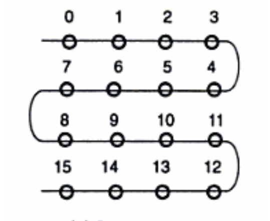
Types of static Interconnection Network

(Some basic types are mentioned and explained. Please go through Kai Hwang for more information)

1. Linear array

→A one dimensional network were N nodes are connected by N-1 links. All intermediate nodes have the degree 2 and terminal nodes have the degree 1

→Below figure shows the representation of linear array with 16 nodes

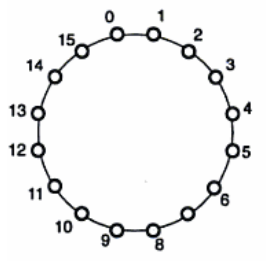


(a) Linear array

2. Ring and Chordal ring network

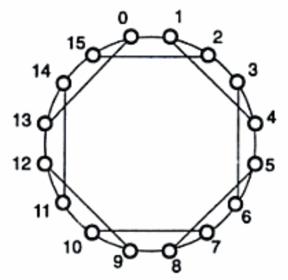
→Ring network is obtained by connecting terminal nodes of the linear array so as to form the ring structure.

→In this network, all the nodes have the degree 2. Below figure shows the representation of Ring network with 16 nodes



(b) Ring

By increasing the node degree of ring network by more than 2, we obtain chordal ring. Below mentioned figure shows the representation of chordal ring of degree 3(every node has 3 connections for other node)

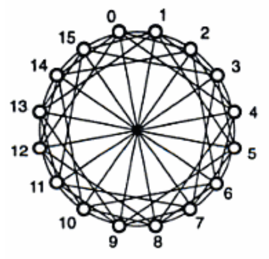


(c) Chordal ring of degree 3

3. Barrel Shifter

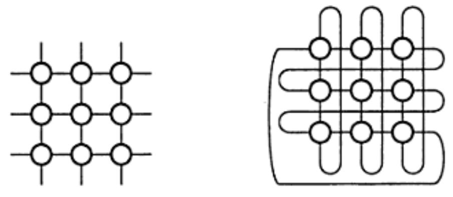
→Obtained from ring network by adding an extra link from each node to those nodes having the distance equal to an integer power of 2

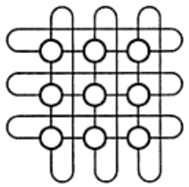
→Here node I is connected to node J if J-I = 2^r for some r = 0, 1, 2….., n-1



(e) Barrel shifter

4. Mesh, Iliac Mesh and Torus


(a) Mesh (b) Illiac mesh


(c) Torus

Dynamic Interconnection Networks

→Dynamic networks are formed of switched channels, which are dynamically configured to match the communication demand in user programs.

→In this type of network, each node is connected to a subset of switches specified. Connections are later applied by configuring the switches dynamically.

→Such connection includes:

1. Bus based connection
2. Single stage cross bar switch- One large switch for whole network.
3. Multistage Interconnection network- More than one switch for each node.

In this section we will be sticking our discussion on to Multi stage interconnection networks and its types.

Multi stage interconnection Networks

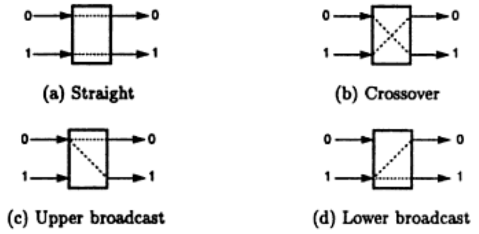
→A type of dynamic network were in more than one switch is used to represent single node. Communication in this type takes place through fixed number of stages based on the population of the nodes.

→Any communication from source to destination completes crossing all the different fixed stages for the network combination.

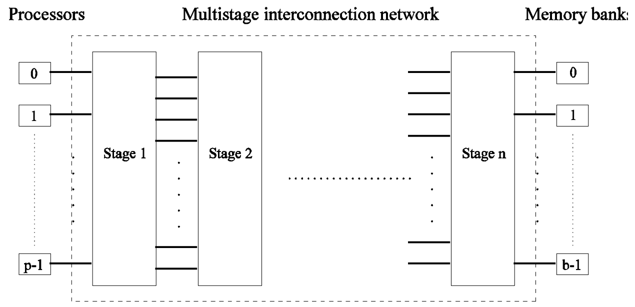
→Switch types used for the communication in the network are called as **cross bar switches.**

→Cross bar switches are the one that is represented in the form of mxn were 'm' represents the number of input nodes and 'n' represents the number of output nodes. m need not be equal to n but generally we represent the switches such that m=n and are represented in the form powers of 2 as 2X2, 4X4, 16X16 and so on.

→ A typical 2X2 switch is being shown below and the its combination of communication:



→Typical representation of multi stage dynamic network is shown below:



→Using this cross bar switch combination, there are two types of dynamic interconnection networks:

1. Omega network
2. Baseline network

1. Omega Network

→These are the type of multistage networks were in there will be many number of switch to represent a single node in different stages.

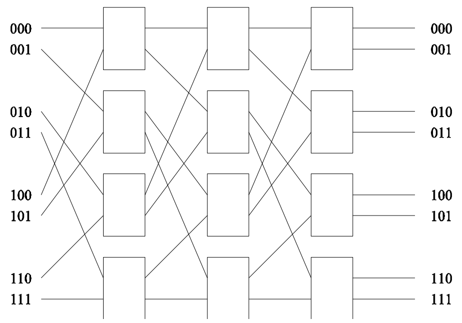
→Number of switches for each stage is same and is fixed based on the module number of the switch.

→Along with the number of switches, number of stages is also fixed and is decided based on the number of nodes used.

→Communication path is also fixed and combination is done by the implementation of perfect shuffle method for each stage.

→We will get a better understanding of the omega network through the example that is considered below

Example: Implementation of 8X8 omega network using 2X2 switches



→Example shows the implementation of 8X8 omega network using 2X2 switches.

→**Selecting number of switches:** An 8X8 omega network has 8 input nodes and 8 output nodes. Each 2X2 switch has connection 2 input nodes and 2 output nodes. So for each stage it requires 4 switches to have 8 input and 8 output.

→ **Naming the nodes:** Nodes are named with binary value from 0 to n-1. Here in this example it starts from 0 and ends with 7. Use uniform number of bits for naming all the nodes and select the number of bits equal to the minimum number of bits required to represent the last node 7. Here the node number starts with the binary value 000 and ends with the binary value 111.

→**Deciding number of stages:** To decide number of stages, consider the highest numbered node in the network and represent the node in the binary value using minimum number of bits. In the above example, highest numbered node is 7 and the binary value of 7 with minimum number of bits is 111, were number of bits required is 3. Hence the number of stages is equal to minimum number of bits to represent the highest node in binary. I.e. the system has three stages and each stage has 4 2X2 switches.

→**Communication mapping:** A perfect shuffle (left circular shift) method is used for the communication path combination. Input for the very first stage is implemented by perfect shuffle of the node number and perfect shuffle is applied over the input for each and every stage.

for example: consider the node number 001. When a perfect shuffle is applied to this binary value, it becomes 010. So the value 001 is fed as input to the node 010. Then output of stage 1 is again perfect shuffled and is fed into the input node 100 in stage 2 and so on.

Base line network

→These are the type of multistage networks were in there will be many number of switch to represent a single node in different stages.

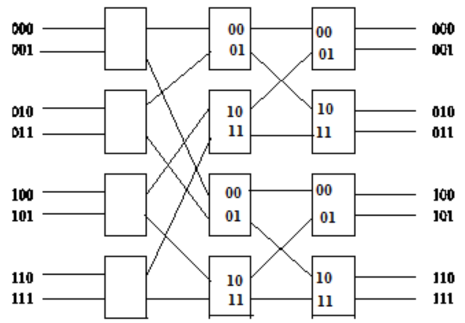
→Number of switches for each stage is same and is fixed based on the module number of the switch.

→Along with the number of switches, number of stages is also fixed and is decided based on the number of nodes used.

→Communication path is also fixed and combination is done by the implementation of inverse perfect shuffle (right circular shift) method for each stage.

→We will get a better understanding of the omega network through the example that is considered below

Example: Implementation 8X8 baseline network using 2X2 switches



→Example shows the implementation of 8X8 baseline network using 2X2 switches.

→**Selecting number of switches:** An 8X8 baseline network has 8 input nodes and 8 output nodes. Each 2X2 switch has connection 2 input nodes and 2 output nodes. So for each stage it requires 4 switches to have 8 inputs and 8 outputs.

→ **Naming the nodes:** Nodes are named with binary value from 0 to n-1. Here in this example it starts from 0 and ends with 7. Use uniform number of bits for naming all the nodes and select the number of bits equal to the minimum number of bits required to represent the last node 7. Here the node number starts with the binary value 000 and ends with the binary value 111.

→**Deciding number of stages:** To decide number of stages, consider the highest numbered node in the network and represent the node in the binary value using minimum number of bits. In the above example, highest numbered node is 7 and the binary value of 7 with minimum number of bits is 111, were number of bits required is 3. Hence the number of stages is equal to minimum number of bits to represent the highest node in binary. I.e. the system has three stages and each stage has 4 2X2 switches.

→**Communication mapping:** An inverse shuffle (right circular shift) method is used for the communication path combination. Input for the very first stage is fed up directly to the nodes. Input to the second stage will be the inverse shuffle of the node number and there will be n number of mapping from output of first stage to input of the second stage. Output from the second stage will be having n/2 mappings into the input of third stage were in each bit from MSB is reduced from the input of the second stage and the system continues so on.

for example: consider the node number 001.001 is mapped to the node 001 output of 001 is inverse shuffled and is mapped to the value 100.