HPC

SHARED MEMORY SYMMETRIC MULTIPROCESSOR

By :-

Digvijay	2021UCA1857
Sneha	2021UCA1859
Sreeram	2021UCA1860
Karan	2021UCA1861
Aditya	2021UCA1862

Synchronization of Processes in Shared Memory Computers

In a shared memory parallel computer, a common program and data are stored in the main memory shared by all PEs. Each PE can, however, be assigned a different part of the program stored in the memory to execute with data also stored in specified locations. The main program creates separate processes for each PE and allocates them along with the information on the locations where data are stored for each process. Each PE computes independently.When all PEs finish their assigned tasks, they have to rejoin the main program. The main program will execute after all processes created by it have been finished. Statements are added in a programming language to enable creation of processes and for waiting for them to complete. Two statements used for this purpose are:

1. fork: to create a process and
2. join: when the invoking process needs the results of the invoked process(es) to continue processing.

EXAMPLE

```
Process X;
:
fork Y;
:
join Y;
```

```
Process Y;
:
:
:
end Y;
```

Process X when it encounters fork Y invokes another Process Y. After invoking Process Y, it continues doing its work. The invoked Process Y starts executing concurrently in another processor. When Process X reaches join Y statement, it waits till Process Y terminates. If Y terminates earlier, then X does not have to wait and will continue after executing join Y.

When multiple processes work concurrently and update data stored in a common memory shared by them, special care should be taken to ensure that a shared variable value is not initialized or updated independently and simultaneously by these processes

SEQUENTIAL CONSISTENCY

An important concept in executing multiple processes concurrently is known as sequential consistency. Lamport [1979] defines sequential consistency as: “A multiprocessor is sequentially consistent if the result of any execution is the same as if the operation of all processors were executed in some sequential order and the operations of each individual processor occurs in this sequence in the order specified by its program”.

In order to ensure this in hardware each processor must appear to issue and complete memory operations one at a time atomically in program order. We will see in succeeding section that in a shared memory computer with cache coherence protocol, sequential consistency is ensured.

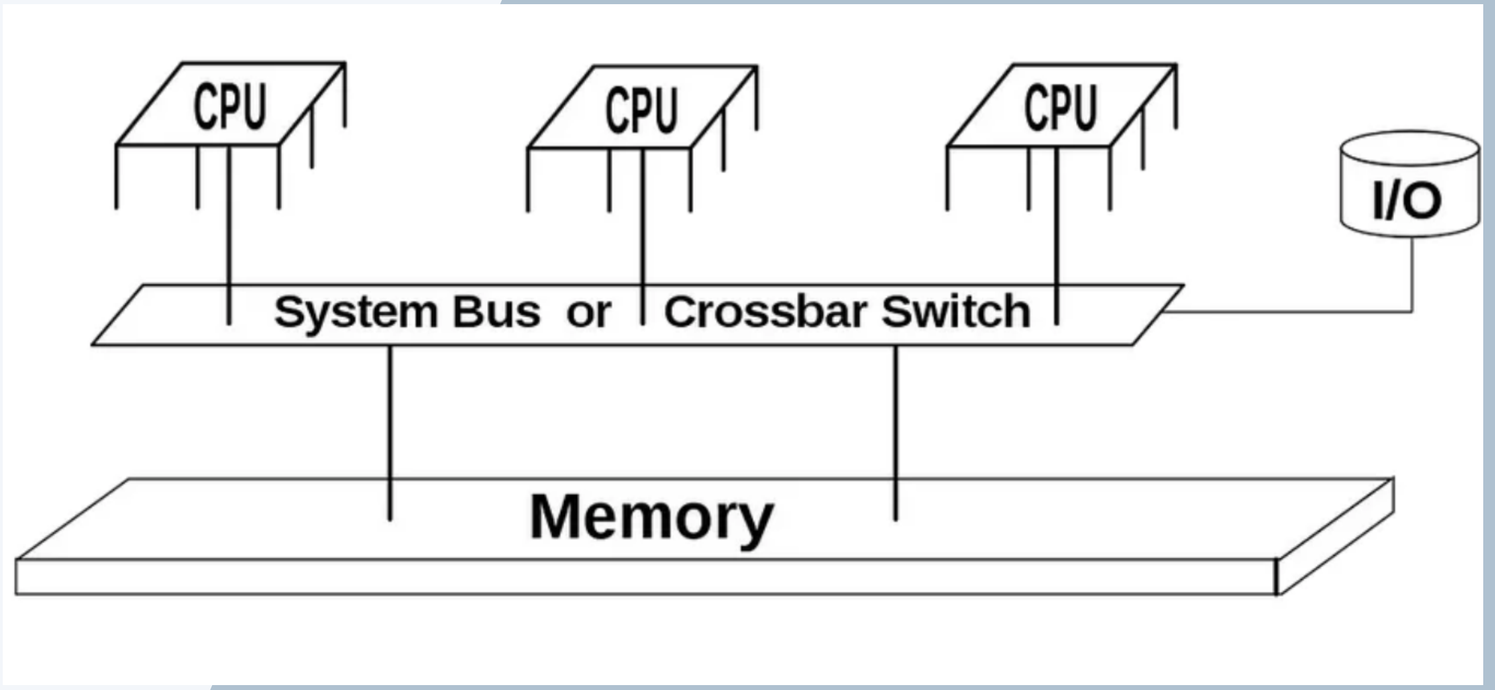
Shared Memory Multiprocessor

1) **UMA** :- *Uniform Memory Access Model*

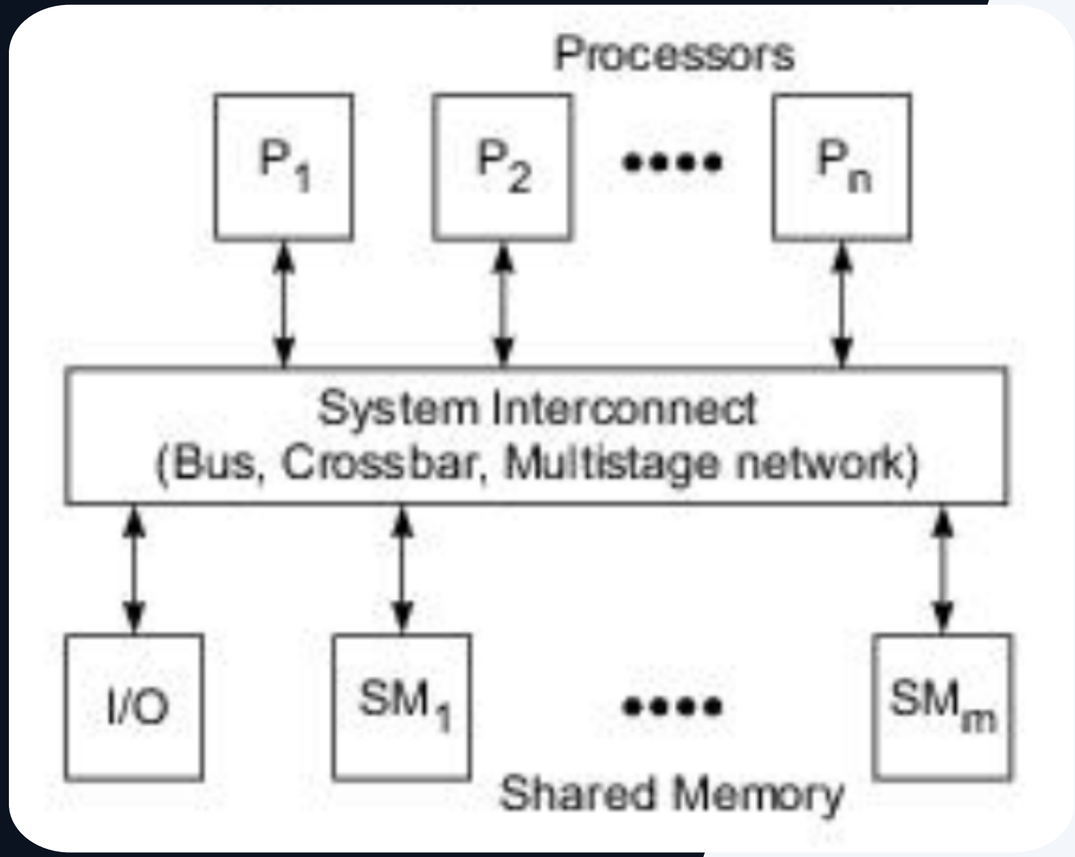
2) **NUMA** :- *Non-uniform Memory Access Model*

3) **COMA** :- *Cache Only Memory Access Model*

CONTINUE



UMA



Physical Memory is uniformly shared

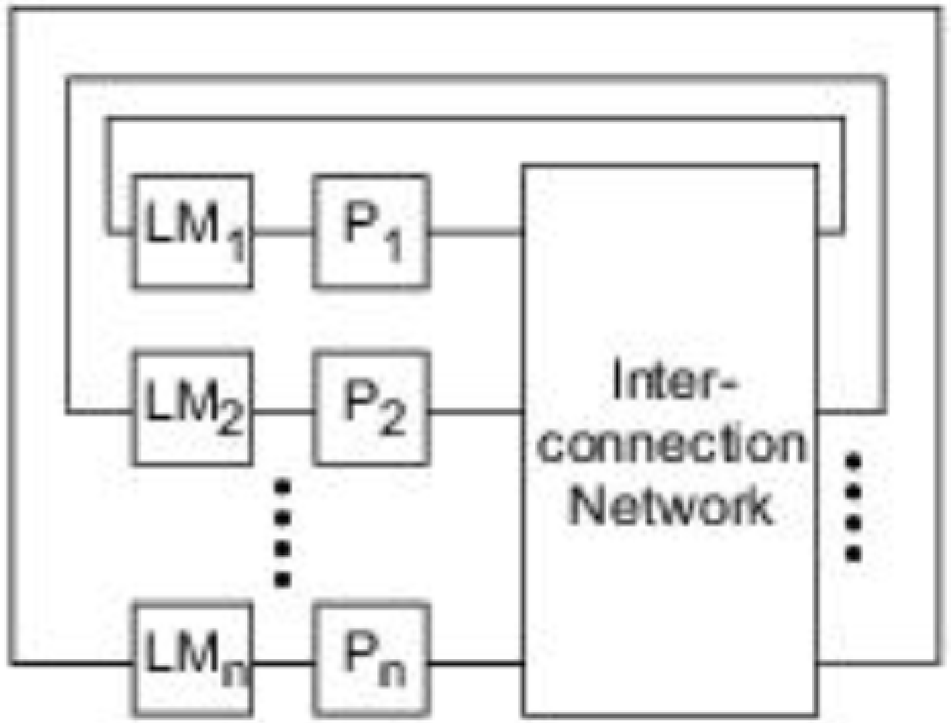
Equal access time for each processor

Suitable for general-purpose and time sharing applications by multiple users

Can be used to speed up the execution of a single large program

Synchronization and communication among processors are done through using shared variables in the common memory.

NUMA



Physical Memory is non-uniformly shared

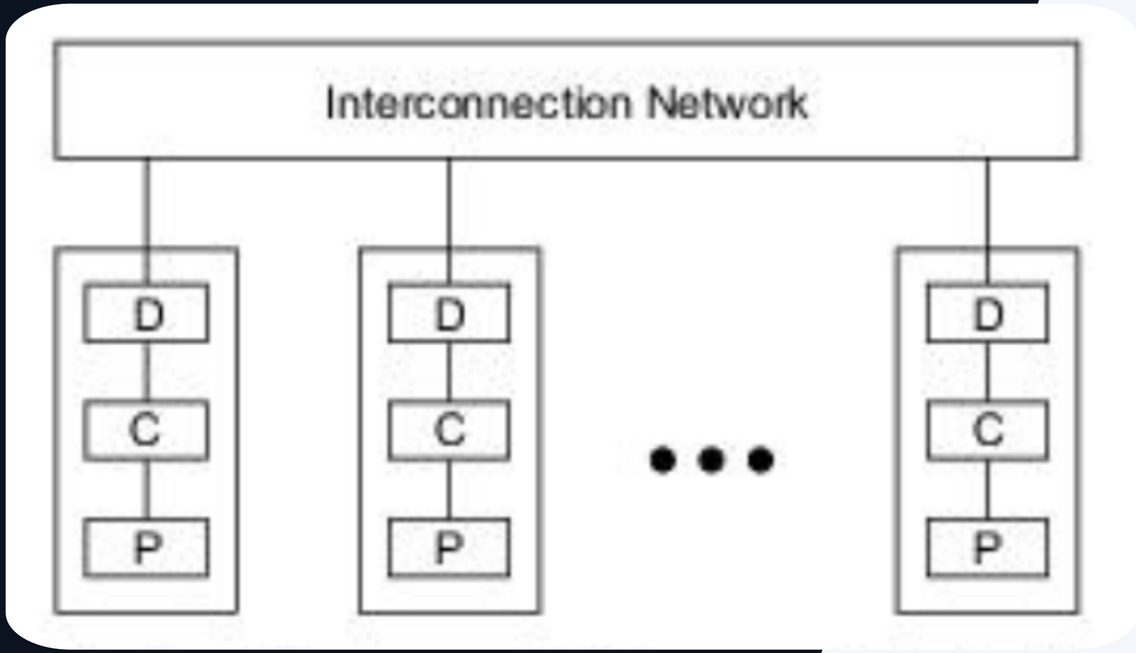
Access time depends on the location of word

Collection of all local memories forms the global address space which is accessible to all

Access time reduces as we move from remote memory to global and then to local

Can be seen as collection of clusters where each cluster works independently and each cluster has equal access to global memory

COMA



Uses cache only memory and thus assumes COMA model

Special case of NUMA where directories are converted to caches

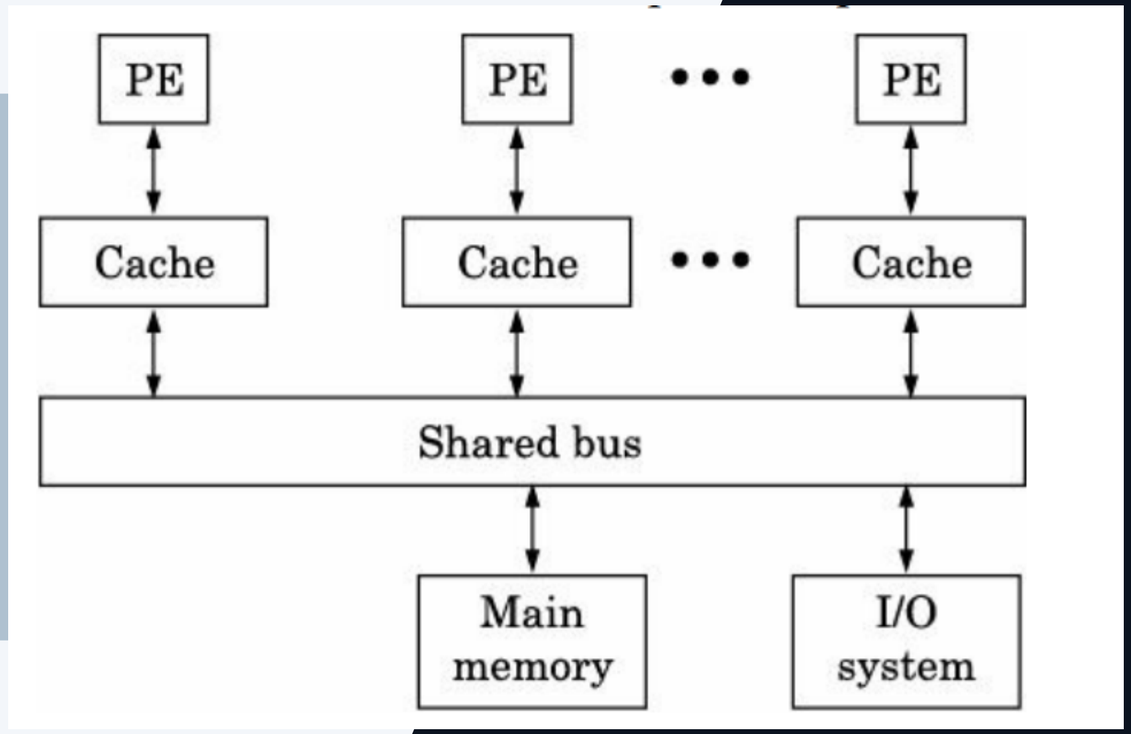
All caches together form global address space which can be assisted by directories

Can be modified with other architectiures and together form different architectures (*eg . CC-NUMA*)

Shared Bus Architecture

A shared bus parallel computer refers to a type of parallel computing architecture where multiple processing elements (PEs) communicate and share data through a common bus

1. **Reducing Access Time:** The private cache memory allows each processing element to access data and programs more quickly compared to accessing them directly from the main memory.
2. **Reducing Bus Traffic:** By storing frequently accessed data and programs in the private cache memory of each processing element, the need for all processors to access the main memory using the bus is reduced.



Let's understand with the help of an example

Consider a shared bus parallel computer built using 32-bit RISC processors running at 2 GHz which carry out one instruction per clock cycle. Assume that 15% of the instructions are loads and 10% are stores. Assume 0.95 hit rate to cache for read and write through caches. The bandwidth of the bus is given as 20 GB/s. 1. How many processors can the bus support without getting saturated? 2. If caches are not there how many processors can the bus support assuming the main memory is as fast as the cache?

SOLUTION

Assuming each processor has a cache.

Number of transactions to main memory/s = Number of read transactions + Number of write transactions
= 0.15 × (0.05) × 2 × 10 9 + 0.10 × 2 × 10^9
= (0.015 + 0.2)10^9 = 0.215 × 10^9 Bus bandwidth
= 20 × 10^9 bytes/s. Average number of bytes traffic to memory
= 4 × 0.215 × 10^9 = **0.86 × 10^9**

∴ Number of processors which can be supported = (20 × 10 9)/(0.86 × 10 9)
= **23**

If no caches are present, then every load and store instruction requires main memory access.
Average number of bytes traffic to memory = 2 × 4 × 0.25 × 10 9 = 150 MB/s
∴ Number of processors which can be supported = (20 × 10 9)/(2 × 10 9) = **10**