Shared-Memory Multiprocessors

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization and Embedded Systems, (6e), McGraw Hill Publication, 2017.
Ch 12: 12.3

Shared-Memory Multiprocessors

- All processors have access to the same memory.
- Tasks running in different processors can access shared variables in the memory using the same addresses.
- The size of the shared memory is likely to be large.
- bottleneck when many processors make requests to access the memory simultaneously.
- Distributed across multiple modules so that simultaneous requests from different processors are more likely to access different memory modules, depending on the addresses of those requests.

Shared-Memory Multiprocessors-UMA

- Interconnection networks-enables any processor to access any module that is a part of the shared memory.
- All requests to access memory must pass through the network, which introduces latency.
- A system which has the same network latency for all accesses from the processors to the memory modules is called a **Uniform Memory Access (UMA)** multiprocessor.

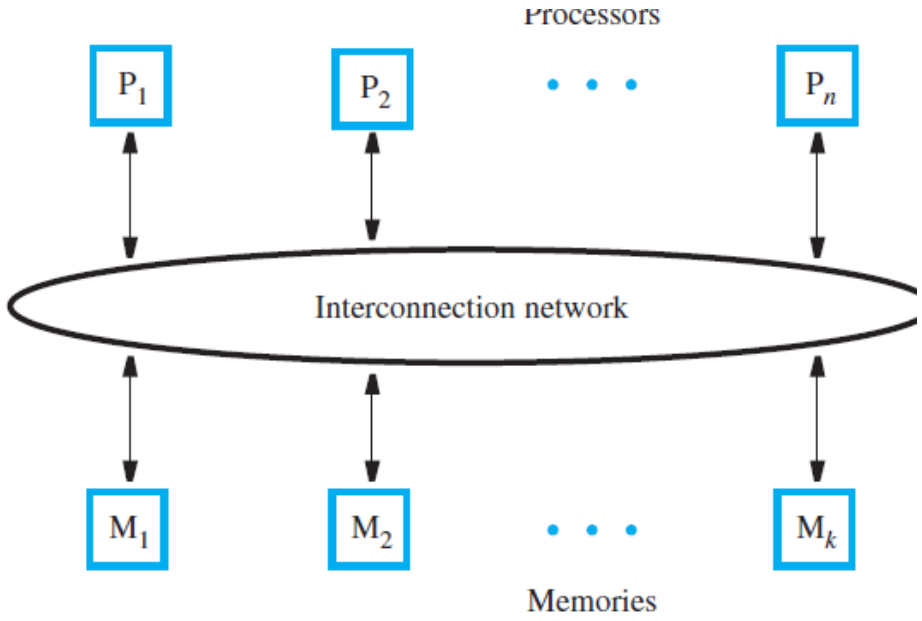

Figure 12.2 A UMA multiprocessor.

NUMA

- It is desirable to place a memory module close to each processor.
- The result is a collection of nodes, each consisting of a processor and a memory module.
- Nodes are then connected to the network, as shown in Figure 12.3.
- The n/w latency is avoided when a processor makes a request to access its local memory.
- a request to access a remote memory module must pass through the n/w
- Because of the difference in latencies for accessing local and remote portions of the shared memory, systems of this type are called **Non-Uniform Memory Access (NUMA)** multiprocessors.

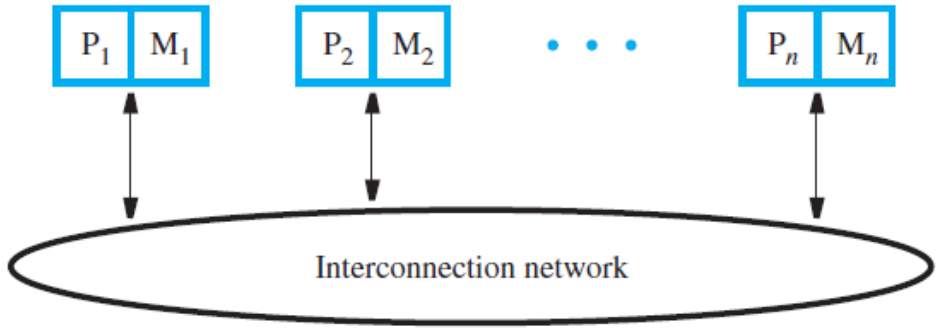

Figure 12.3 A NUMA multiprocessor.

Cache Coherence

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization and Embedded Systems, (6e), McGraw Hill Publication, 2017.
Ch 12: 12.4, 12.4.1, 12.4.2.

Cache Coherence

- Possibility that copies of shared data may reside in several caches.
- When any processor writes to a shared variable in its own cache, all other caches that contain a copy of that variable will then have the old, incorrect value.
- They must be informed of the change so that they can either update their copy to the new value or invalidate it.
- This is the issue of maintaining cache coherence, which requires having a consistent view of shared data in multiple caches.

Write-Through Protocol

A write-through protocol can be implemented in one of two ways

1. updating the values in other caches
   - When a processor writes a new value to block of data in its cache, the new value is also written into the memory module containing the block being modified.
   - Since copies of this block may exist in other caches, these copies must be updated to reflect the change caused by the Write operation. The simplest way
     - broadcast the written data to the caches of all processors in the system.
   - As each processor receives the broadcast data, it updates the contents of the affected cache block if this block is present in its cache

2. invalidating the copies in other caches
   - When a processor writes a new value into its cache, this value is also sent to the appropriate location in memory, and all copies in other caches are invalidated.
   - broadcasting can be used to send the invalidation requests throughout the system

Write-Back protocol

- Based on the concept of ownership of a block of data in the memory.
- If some processor wants to write to a block in its cache, it must first become the exclusive owner of this block. To do so, all copies in other caches must first be invalidated with a broadcast request.
- The new owner of the block may then modify the contents at will without having to take any other action
- When another processor wishes to read a block that has been modified, the request for the block must be forwarded to the current owner. The data are then sent to the requesting processor by the current owner.
- The data are also sent to the appropriate memory module, which reacquires ownership and updates the contents of the block in the memory.
- The cache of the processor that was the previous owner retains a copy of the block. Hence, the block is now shared with copies in two caches and the memory.
- Subsequent requests from other processors to read the same block are serviced by the memory module containing the block.