9 Multiprocessor / Multicore / Multicomputer Systems

To enhance system performance and, in some cases, to increase availability multiple processing units are used that can execute in parallel.

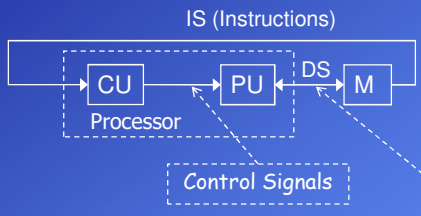
Levels of parallelism in software:

- Instruction level parallelism: Pipelining. Portions of different instructions run in parallel.
- Parallel programming: Portions of a single program (task) run on multiple processors simultaneously (A type of thread-level parallelism).
- Job-level or process-level parallelism : Independent applications run on different processors (Also a type of thread-level parallelism).
- Data-level parallelism: Data pipelining or multiple functional units (ALU). For example different elements of an array can be processed simultaneously.

9.1 Flynn's Taxonomy by Michael J. Flynn (1934-)

There are different types of parallel organizations.
A taxonomy first introduced by Flynn is still the most common way of categorizing systems with parallel processing capability.

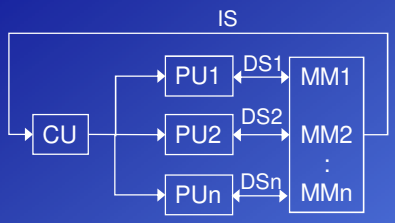
a) SISD: Single Instruction Single Data



CU: Control Unit
PU: Processing Unit
M: Memory
IS: Instruction Stream
DS: Data Stream

A single processor executes a single instruction stream to operate on data stored in a single memory (Uniprocessor).

b) SIMD: Single Instruction Multiple Data



CU: Control Unit
PU: Processing Unit
MM: Memory Module

Each PU can have its own memory module.

A single machine instruction controls all PUs.

Each processing element has an associated data memory, so that the instruction is executed on a different set of data by the different PUs in parallel.

Example: Vector and array processors.

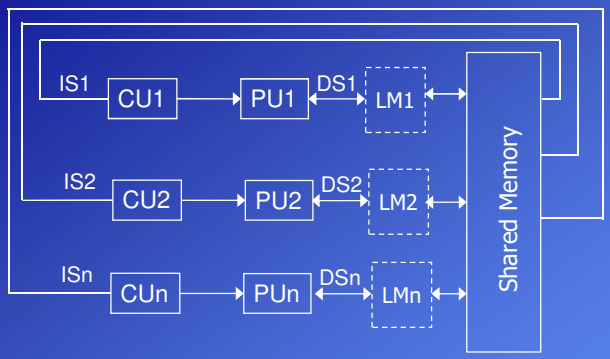
c) MISD: Multiple Instruction Single Data (not commercially implemented)

Different instructions are executed on the same data.

It can be used that different backup systems generate same result for same input (data).

d) MIMD: Multiple Instruction Multiple Data

i. with shared memory

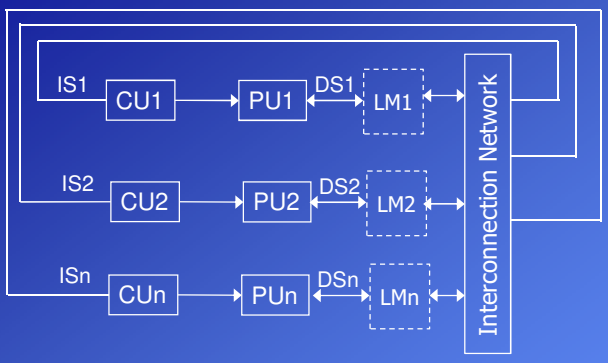


CU: Control Unit
PU: Processing Unit
LM: Local Memory
IS: Instruction Stream
DS: Data Stream

A set of processors simultaneously execute different instruction sequences on different data sets.

- **Shared memory** (*tightly coupled*) systems: The processors share a common memory and they communicate with each other (share data) via that memory.
  The processors may also have their local memories (cache).

d) MIMD: Multiple Instruction Multiple Data (cont'd)
ii. with distributed memory

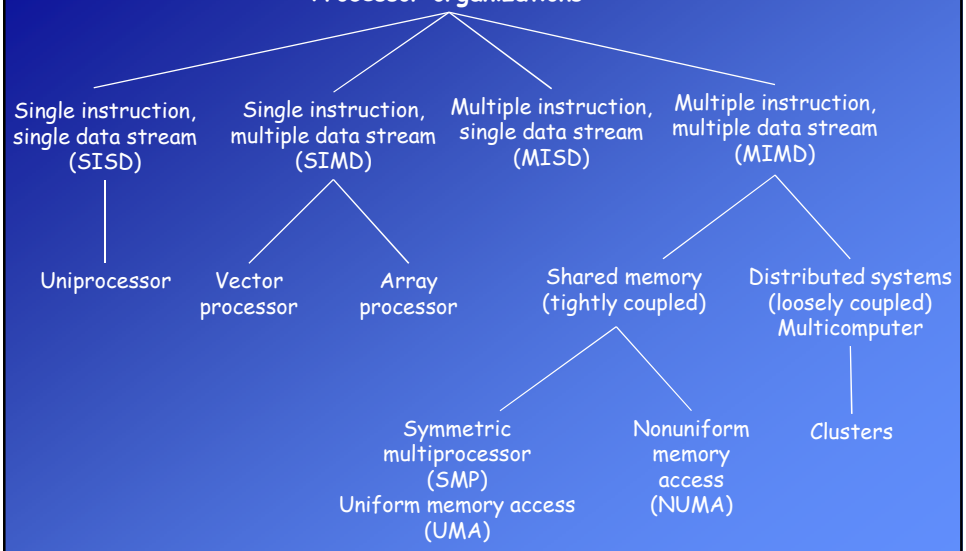


CU: Control Unit
PU: Processing Unit
LM: Local Memory
IS: Instruction Stream
DS: Data Stream

- **Distributed memory** (loosely coupled) systems: A collection of independent uniprocessors are interconnected to form a cluster.

Communication among the computers is either via fixed paths or via some network facility.

Processor organizations



9.2 Shared Memory (Tightly Coupled) Systems

- Single physical address space (shared-memory)
- Processors communicate through shared variables in memory.
- All processors are capable of accessing any memory location via load and store instructions.
- The system is controlled by an integrated common operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.
- Because of the shared variables the operating system must support synchronization.
- There are two different types of the shared memory systems:

a) Symmetric multiprocessor (SMP) or Uniform memory access (**UMA**) systems:
The memory access time is approximately the same for each processor (symmetric), no matter which processor accesses which memory word.

b) Nonuniform memory access (**NUMA**) multiprocessors:
The processors still share the same single address space but memory units are physically distributed in the system.
A processor can access a memory faster when it is close to it.

9.2.1 Symmetric Multiprocessors (SMP) Uniform memory access (UMA) systems

Characteristics:

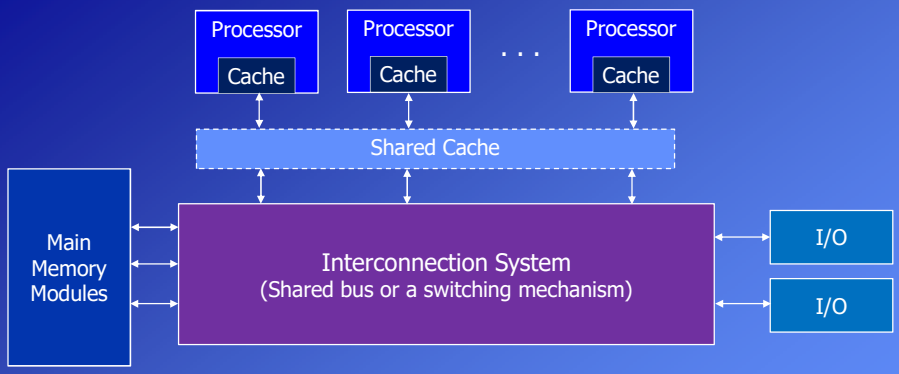
- Single, common address space (shared-memory) , single operating system.
- There are two or more processors with similar (same) capabilities.
- All processors can perform the same functions (symmetric).
- Processors share the same main memory and I/O facilities.
- System components are interconnected by a bus or other internal connection scheme such as crossbar switch.
- The memory access time is approximately the same for each processor (symmetric) (UMA).

Potential benefits:

- Performance is better than a uniprocessor.
- Availability: Because all processors can perform the same functions, the failure of a single processor does not halt the machine.
- Incremental growth: A user can enhance the performance of a system by adding an additional processor.

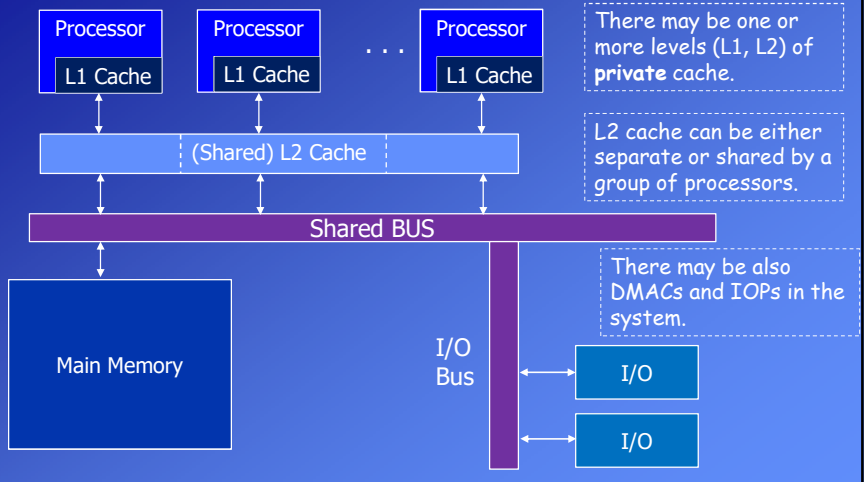
But! As more processors are added, competition for access to the bus leads to a decline in performance (max. 64 processors).

Organization (SMP, UMA):



- Each processor includes a control unit, ALU, registers, and, typically, one or more levels of cache.
- The memory can be interleaved or a multiport memory, where multiple simultaneous accesses to separate blocks of memory are possible.
- The interconnection system can be designed in different ways: shared bus, crossbar switch.

Symmetric Multiprocessor Organization with a Shared Bus:



Time-sharing: When one module is controlling the bus, other modules are locked out and must, if necessary, suspend operation until bus access is achieved.
Bus arbitration is necessary.

Advantages:

- **Simplicity:** The physical interface and the addressing, arbitration, and time-sharing logic of each processor remain the same as in a single-processor system.
- **Flexibility:** It is generally easy to expand the system by attaching more processors to the bus (but it has limits).
- **Reliability:** The bus is essentially a passive medium, and the failure of any attached device should not cause failure of the whole system.

Drawbacks:

- **Performance:** All memory references pass through the common bus.
  The bus cycle time limits the speed of the system.
  The common bus is used in time-sharing manner. When a processor or DMAC is accessing the bus other processors cannot access the main memory.
  The number of CPUs is limited (16 - 64).

Solution:

- Local cache memories. Most frequently used data are kept in cache memories.
  Hence, the need to access the main memory is reduced.
- **Cache coherence** problem: If a word is modified in a local cache, the copies of the same word in other caches will be invalid. Other processors must be alerted. (Explained in chapter 9.4 Cache Coherence)

9.2.2 Nonunifrom memory access (NUMA) multiprocessors

In SMP systems the common bus is a performance bottleneck.

The number of processors is limited.

Loosely coupled systems (clusters) can be a solution, but in these systems applications cannot see a global memory.

NUMA systems are designed to achieving large-scale multiprocessing while retaining the advantages of shared memory.

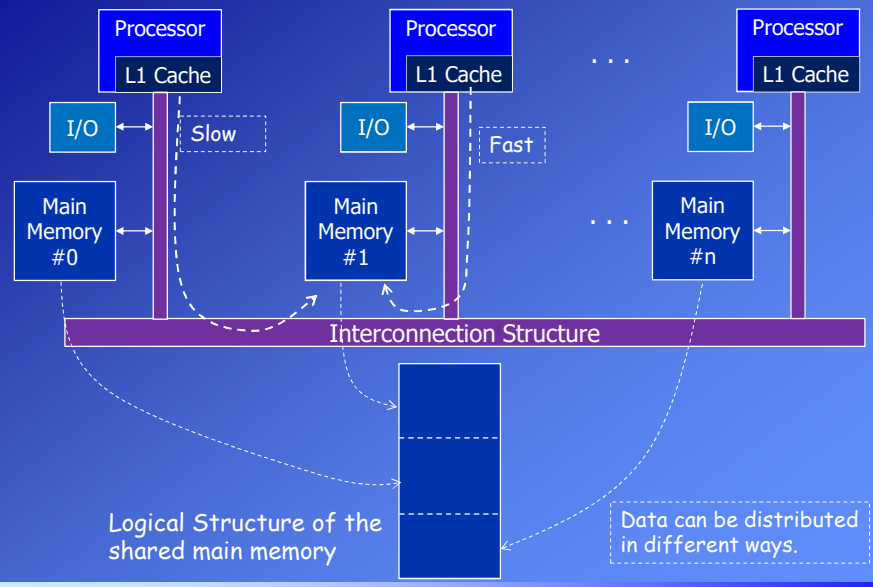
Characteristics:

- Single Address Space (shared-memory) , single operating system.
- The shared memory is physically distributed to the CPUs. These systems are also called **distributed shared memory** systems.
- A CPU can access its own memory module faster than other modules.

Performance:

- If processes and data can be distributed in the system so that CPUs are mostly accessing their main memory modules (or local cache memories) and rarely remote memory modules ,then the performance of the system increases.
- Spatial and temporal locality of programs and data play an important role again.

A NUMA Multiprocessor Organization:



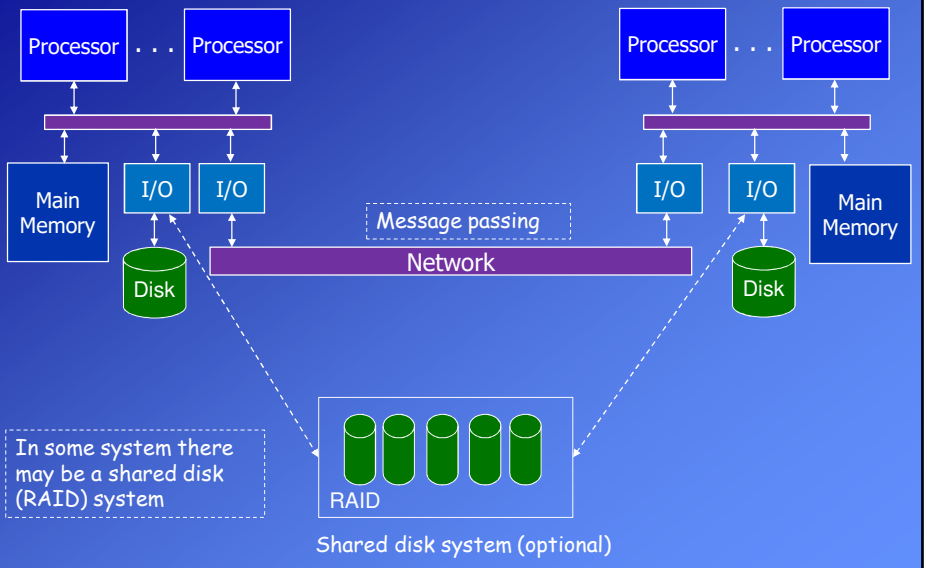
9.3 Distributed (loosely coupled) systems, Multicomputers

- Each processor has its own physical address space.
- These processors communicate via message passing.
- The most widespread example of message passing system are **clusters**.
- Clusters are collections of computers that are connected to each other over standard network equipment.
- When these clusters grow to ten of thousands of servers and beyond, they are called warehouse-scale computers (cloud computing).

Benefits:

- **Scalability:**
  - A cluster can have tens, hundreds, or even thousands of machines, each of which is a multiprocessor.
  - It is possible to add new systems to the cluster in small increments.
- **High availability:**
  - Each node in a cluster is a standalone computer, therefore failure of one node does not mean loss of service.
- **Superior price/performance:**
  - By using cheap commodity building blocks, it is possible to build a clusters with a great computing power.

A Loosely Coupled (distributed) System Organization:

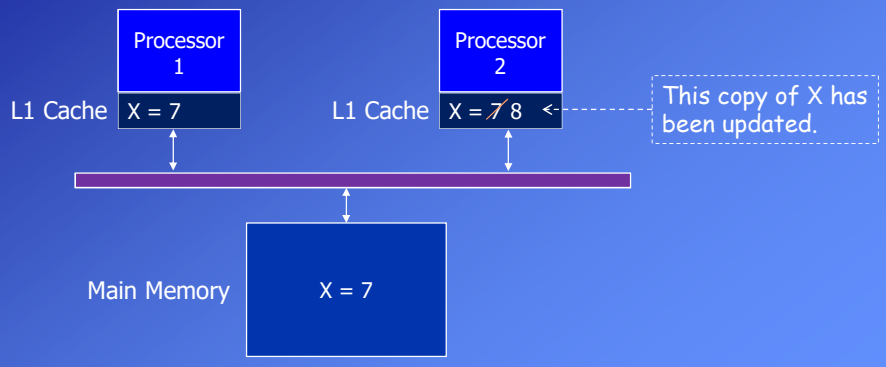


9.4 Cache Coherence

To reduce the average access time and the required memory bandwidth cache memories are used.

Caching the shared data introduces the **cache coherency problem**.

Multiple copies of the same data can exist in different caches simultaneously, and if processors are allowed to update their own copies freely, an inconsistent view of memory can result.



9.4.1 Software solutions:

- No need for additional hardware circuitry.
- Compiler and operating system deal with the problem at compile time.
- But they make conservative decisions, leading to inefficient cache utilization.
- Compiler-based mechanisms perform an analysis on the code to determine which data items may become (when) unsafe for caching, and they mark those items.

  The operating system or hardware then prevents these items from being cached.
- The simplest approach is to prevent any shared data variables from being cached (too conservative and inefficient).
- More efficient approach is to analyze the code to determine safe and critical periods for shared variables and inserting instructions into code to enforce cache coherence.

9.4.2 Hardware solutions:

a) Directory protocols:

There is a centralized controller that maintains a directory that is saved in the main memory.

The directory contains information about which processors have a copy of which lines (frames) in its private cache.

- When a processor wants to write to a local copy of a line, it must request exclusive access to the line from the controller.
- The controller sends a message to all processors with a cached copy of this line, forcing each processor to invalidate its copy.
- After receiving acknowledgments back from each such processor, the controller grants exclusive access to the requesting processor.
- When another processor tries to read a line that is exclusively granted to another processor, a miss occurs.
- If write-back mechanism is used the controller issues a command to the processor holding that line that requires the processor to do a write-back to main memory.
- The line may now be shared for reading by the original processor and the requesting processor.

a) Directory protocols (cont'd):

Drawbacks:

- Centralized controller is a bottleneck. All requests to the same controller
- Overhead of the communication between local cache controllers and the central controller.

Advantage:

- Effective in large-scale systems that involve multiple buses or some other complex interconnection scheme.

b) Snoopy protocols:

- The responsibility for maintaining cache coherence is **distributed** among all of the cache controllers in the multiprocessor system.
- When a shared cache frame (line) is updated, the local controller announces this operation to all other caches by a broadcast mechanism.
- Each cache controller is able to "snoop" on the network to observe these broadcasted notifications, and react accordingly (for example invalidating the copy).
- Snoopy protocols are suitable for bus-based multiprocessors, because the shared bus provides a simple mechanism for broadcasting and snooping.
- Remember local caches are used to decrease the traffic on the shared bus.

  Therefore care must be taken not to increase the traffic on the shared bus by broadcasting and snooping.

b) Snoopy protocols (cont'd):

There are two types of snoopy protocols: write-invalidate and write-update

Write-invalidate protocol:

- When one of the processors wants to perform a write to the line in the private cache, it sends a "invalidate" message.
- All snooping cache controllers invalidate their copy of the appropriate cache line.
- Once the line is exclusive (not shared), the owning processor can write to its copy.
- If write-through method is used the data is also written to the main memory.
- If another CPU attempts to read this data a miss occurs and data is fetched from the main memory.

Write-update protocol:

- When one of the processors wants to update a shared line, it broadcasts the new data to all other processors so that they can also update their private caches.
- At the same time the CPU updates its own copy in the cache.

Experience has shown that **invalidate protocols use significantly less bandwidth**.

The MESI (Modified Exclusive Shared Invalid) **Protocol**

- A snoopy, write-invalidate cache coherency protocol
- It allows the usage of the write-back method. Main memory is not updated until it is necessary to replace the frame.
- Each cache frame (line) can be in one of four states (2 status bits):

M (Modified): The frame in this cache is modified. It is different from the main memory.
This frame is valid only in this cache.

E (Exclusive): The frame in the cache is the same as that in main memory and is not present in any other cache.

S (Shared): The frame in the cache is the same as that in main memory and may be present in another cache.

I (Invalid): The line in the cache does not contain valid data.

	Modified	Exclusive	Shared	Invalid
Is the cache frame valid?	Yes	Yes	Yes	No
Is the data in the main memory valid?	No	Yes	Yes	-
Copies exist in other caches?	No	No	Maybe	Maybe

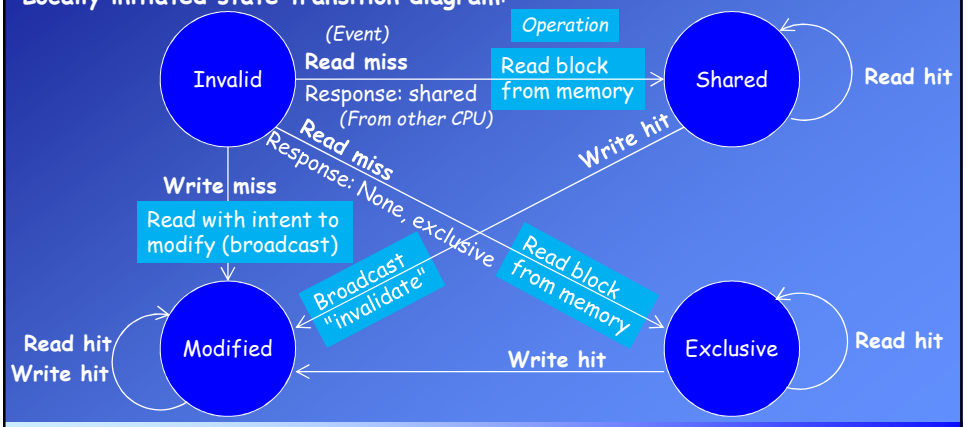
State transition of frames:

2 bits are assigned to each cache frame in tag memory to keep the state value.

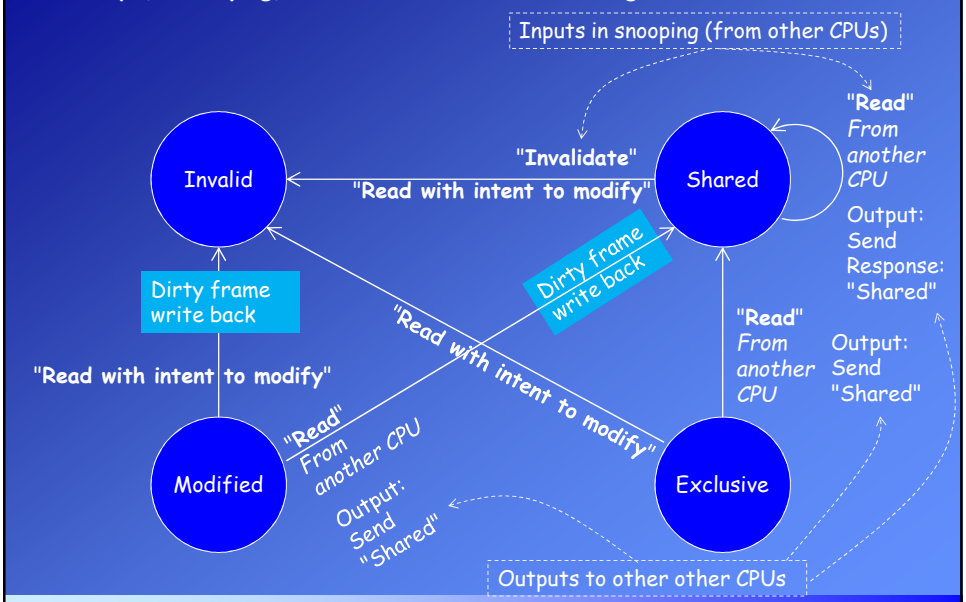
The state of a cache frame can change because of two reasons:

- Due to local processor activity (i.e. cache access)
- Due to bus activity in snooping (activity caused by another processor)

Locally initiated state transition diagram:



Remotely (in snooping) initiated state transition diagram:



The MESI Protocol (cont'd)

Operation of the protocol:

Read Miss (Invalid state):

- The processor starts to fetch the frame from the main memory.
- The CPU signals other cache controllers to snoop the operation.
- Possible responses:

A. If another cache has a unmodified (clean) exclusive copy, it indicates that it shares this data.

The responding cache frame goes from exclusive to shared state.

The initiating CPU reads the frame from the memory and the cache frame goes from invalid to shared state.

B. If other caches have unmodified (clean) shared copies, they indicate that they share this data.

The responding cache frames stay in the shared state.

The initiating CPU reads the frame and the frame goes from invalid to shared state.

Read Miss (Invalid) cont'd:

- Possible responses (cont'd):

C. If another cache has a modified (dirty) copy, it blocks memory read operation and provides the requested frame.

This data is also written to the main memory.

There are different implementations. The requesting CPU can read the data from the responding CPU or from the main memory after the memory has been updated.

The cache frame of the responding CPU goes from modified to shared state.

The cache frame of the initiating CPU goes from invalid to shared state.

D. If no other cache has a copy of the requested frame, then no signals are returned.

The initiating CPU reads the frame from the memory and the cache frame goes from invalid to exclusive state.

Read Hit:

The CPU simply reads the required data from the cache.
The cache frame remains in the same (current) state: modified, shared or exclusive.

Write Miss (Invalid state):

- The processor starts to fetch the frame from the main memory.
- The CPU issues the signal *read-with-intent-to-modify* on the bus.
- Possible scenarios:

A. If another cache has a modified copy of the frame, it signals the requesting CPU (some words in this frame have been modified).

The requesting CPU terminates the bus operation and waits.

The other CPU writes the modified cache frame back to main memory, and transitions the state of the cache from modified to invalid.

The initiating CPU issues the signal *read-with-intent-to-modify* on the bus again and reads the frame from main memory.

The CPU modifies the word in the frame and transitions the state of the frame to modified.

B. If no other cache has a modified copy of the requested frame, then no signals are returned.

The initiating CPU reads the frame from main memory and modifies it.

If one or more caches have a clean copy of the frame in the shared or exclusive state, each cache invalidates its copy of the frame.

Write Hit:

The CPU attempts to write (modify a variable) and the variable (frame) is in the local cache.

Operations depend on the state of the frame being modified.

Shared:

- The CPU broadcasts "invalidate" signal on the shared bus.
- Other CPUs that have a shared copy of the frame in their cache transition the state of that frame from "shared" to "invalid".
- The initiating CPU updates the variable and transitions its copy of the frame from "shared" to "modified".

Exclusive:

- The CPU has already the single (exclusive) copy of the data.
- The CPU updates the variable and transitions its copy of the frame from "exclusive" to "modified".

Modified:

- The CPU has already the single modified copy of the data.
- The CPU updates the variable. The state remains as "modified".

Example:

In a symmetric multiprocessor (SMP) system with a shared bus, there are two CPUs (CPU1 and CPU2) that have local cache memories.

The system does not include a shared L2 cache.

The cache control units use the set associative mapping technique, where each set contains two frames (2-way set associative).

In write operations Flagged Write Back (FWB) with Write Allocate (WA) methods are used.

Assume that there is a shared variable X in the system. To provide cache coherence the snoopy MESI protocol is used.

The following questions can be answered independently.

a) Assume that caches of the both CPUs include valid copies of the variable X. If the copy of X is in set:1, frame:0 in the cache of CPU1, can we know its location in the cache of CPU2? Why?

Solution:

In a symmetric multiprocessor (SMP) system CPUs use the same memory space, therefore the variable X has the same address in both spaces of CPU1 and CPU2.

If it is in set:1, frame:0 in the cache of CPU1, then it must be also in set:1 of the cache of CPU2. But we cannot know which frame in set 1.

Example (cont'd):

b) Assume that the frame in the cache of CPU1 including the variable X is in state "exclusive". What is the state of the corresponding frame in the cache of CPU2?

Solution:

In this case, valid copies of the variable X are in the main memory and in the cache of CPU1. Therefore, the state of the corresponding frame in the cache of CPU2 must be in state "**invalid**".

c) Assume that the frame in the cache of CPU1 including the variable X is in state "modified" and the CPU2 wants to write to variable X. List the operations performed by the MESI protocol.

Solution:

If it is in modified in CPU 1, then it does not exist (invalid) in CPU 2.

- CPU2 (Write miss) issues the signal *read-with-intent-to-modify*.
- CPU1 signals the requesting CPU2 "*Main memory is not valid*".
- CPU1 writes the modified cache frame back to main memory, and transitions the state of the cache from "modified" to "invalid".
- CPU2 issues the signal *read-with-intent-to-modify* again and reads the frame from main memory.
- CPU2 modifies the word in the frame and transitions the state of the frame to "modified".

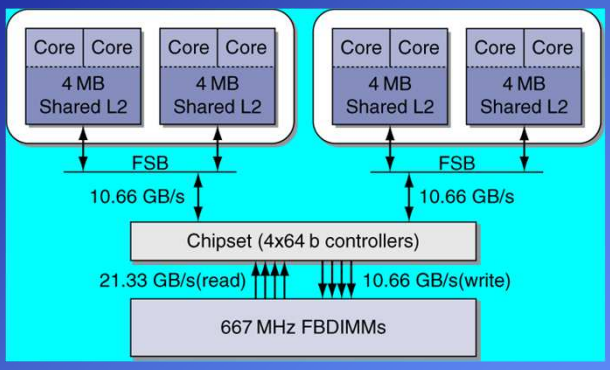
9.5 Challenges of parallel processing

- Limited parallelism in programs. Some processors cannot be loaded.
- Overhead for communication. High cost of communications between processors.
- Writing parallel programs is difficult.
- Partitioning into independent parts with similar loads: Scheduling and load balancing problem.
- Synchronization: Dependencies, critical sections

9.6 Exemplary Multiprocessor Systems [1]

When more than one processor is implemented on a single chip, the system is called **multicore chip processor**.

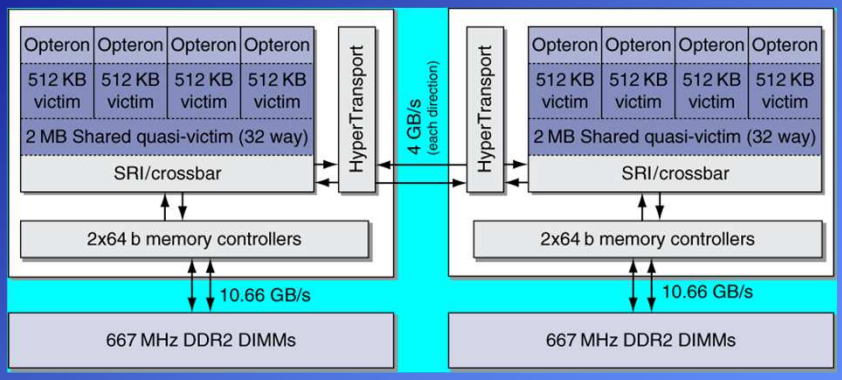
A SMP system with 2 × quad-core Intel Xeon e5345 (Clovertown):



L1 Cache:
128 KB (code)
128 KB (data)

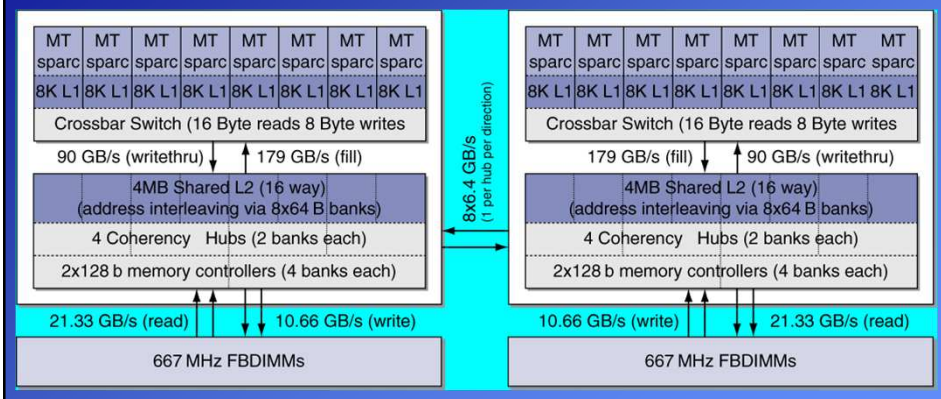
1 Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × quad-core AMD Opteron X4 2356 (Barcelona)



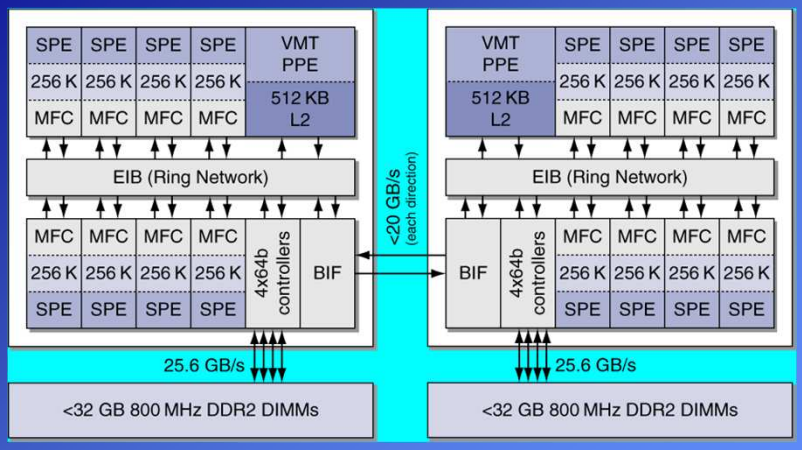
Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × oct-core Sun UltraSPARC T2 5140 (Niagara 2)



Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × oct-core IBM Cell QS20:



Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

The performance wall and search for new solutions *

Computing has evolved because of improvements in semiconductor devices (transistors) and computer architecture (cache memories, pipeline etc.).

But this improvements (especially the Moore Law) are ending.

The designers increase clock frequency and/or the number of transistors in a integrated circuit (IC) to increase the processing speed of the computers.

But this causes heat/cooling problems (power wall).

Architectural solutions such as pipeline, multicore systems have also its own problems.

However, demands for performance in excess of 1 million trillion floating-point operations per second (1 exaflops) are arising from novel software paradigms to address problems in big data, machine learning, security.

Many industry experts believe that, by 2020, computing will reach the long-predicted **performance wall**.

Visit the web site of the IEEE Rebooting Computing Initiative to explore the future of computing systems in the architecture, device, and circuit domains.

http://rebootingcomputing.ieee.org/

*Source: T. M. Conte, E. P. DeBenedictis, P. A. Gargini, and E. Track, "Rebooting Computing: The Road Ahead," Computer, vol. 50, no. 1, pp. 20–29, Jan. 2017.