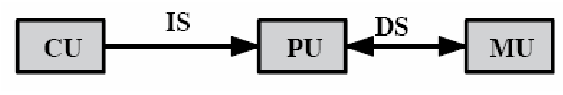
PARALLEL PROCESSING

Classifications of Parallel Processing

Single instruction/single data stream (SISD)

Description: Single processor operates on a single instruction stream from a single memory

Examples: Standard single-processor system



Multiple instruction/single data stream (MISD)

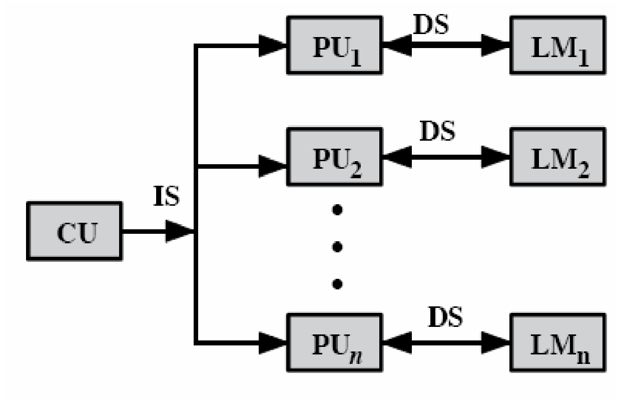
Description: Multiple processors execute different sequences of instructions on a single data set.

Examples: Not commercially implemented

Single instruction/multiple data stream (SIMD)

Description: Lockstep operation of multiple processors on single instruction memory with one data memory per processing element.

Examples: Vector or array processing



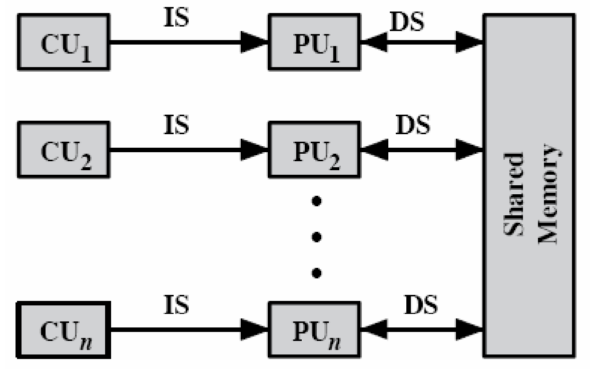
KEY FOR FIGURES

CU = control unit	IS = instruction stream
PU = processing unit	DS = data stream
MU = memory unit	LM = local memory

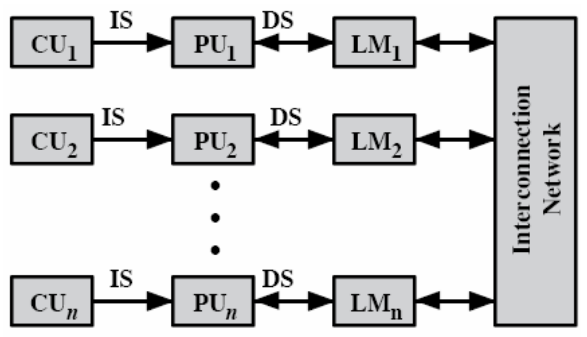
Multiple instruction/mult. data stream (MIMD)

Description: A set of processors simultaneously execute different instructions on different data sets.

Examples: SMP, clusters, and NUMA systems



MIMD (with shared memory)



MIMD (with distributed memory)

Multiple Instruction/Multiple Data Stream

Characteristics:

- Processors are general purpose
- Each processor should be able to complete process by themselves
- Communications methods
  - "Tightly Coupled" – Processors communicate through shared memory
    - Symmetric multiprocessor (SMP) – memory access times are consistent for all processors
    - Nonuniform Memory Access (NUMA) – memory access times may differ
  - "Loosely Coupled" – Either through fixed connections or a network (cluster)

Characteristics of a Symmetric Multiprocessors (SMP)

An SMP system is a stand alone computer with the following traits:

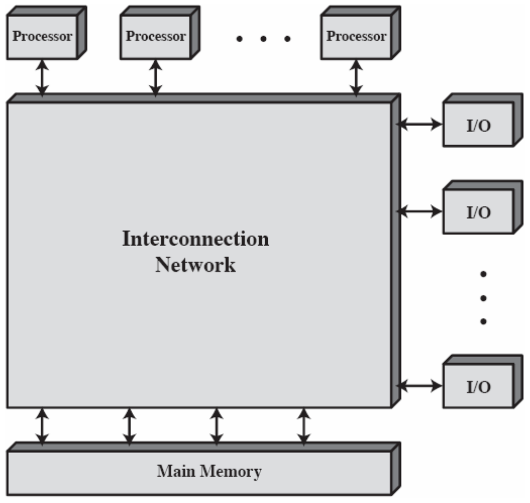
- Two or more similar processors of comparable capacity
- Processors share same memory and I/O
- Processors are connected by a bus or other internal connection
- Memory access time is approximately the same for each processor
- All processors share access to I/O through either same channels or different channels providing paths to same devices
- All processors can perform the same functions (hence symmetric)
- System controlled by integrated operating system providing interaction between processors
- Interaction at job, task, file and data element levels
- Integrated operating system
  - O/S for SMP is NOT like clusters/loosely coupled where communication usually is at file level
  - Can be a high degree of interaction between processes
  - O/S schedules processes or threads across all processors

Advantages of a Symmetric Multiprocessors (SMP)

Advantages only realized if O/S can provide parallelism

- Improved performance, but only if the applications allow some work to be done in parallel
- Availability/reliability – Since all processors can perform the same functions, failure of a single processor does not halt the system
- Incremental growth – User can enhance performance by adding additional processors
- Scaling – Vendors can offer range of products based on number of processors
- Transparent to user – User only sees improvement in performance

Organization of Tightly Coupled Multiprocessor



- Individual processors are self-contained, i.e., they have their own control unit, ALU, registers, one or more levels of cache, and private main memory
- Access to shared memory and I/O devices through some interconnection network
- Processors communicate through memory in common data area
- Memory is often organized to provide simultaneous access to separate blocks of memory

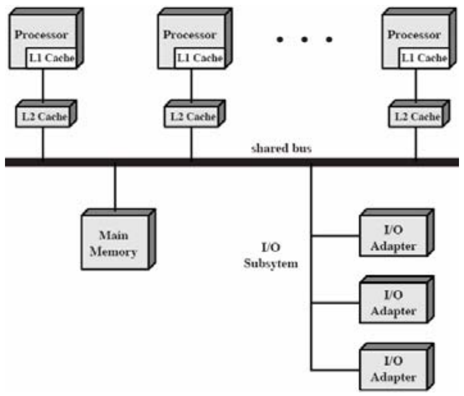
SMP Operating System

- To user, it appears as if there is a single O/S, i.e., single processor multiprogramming system
- User should be able to create multithreaded processes without needing to know whether one processor or more will be used
- Simultaneous concurrent processes
  - O/S routines should be reentrant
  - Expanded O/S tables & other management structures to handle multiple processes & processors
- All processors should be capable of scheduling
- Synchronization – scheduling of resources now more than just for processes but also for processors
- Memory management
  - Shared page replacement strategy
  - Must understand and take advantage of memory hardware
- Reliability and fault tolerance – Must be able to handle the loss of a processor without taking down other processors.

Three Types of Tightly Coupled Multiprocessor Interconnection Schemes

Time-shared or common bus

- Structure and interface similar to single processor system (control, address, and data)
- Sharing bus is similar to DMA sharing with single processor
- Following features provided
  - Addressing - distinguishes modules on bus
  - Must have an arbitration scheme where any module can be temporary master
  - Time sharing - if one module has the bus, others must wait and may have to suspend
- Now have multiple processors as well as multiple I/O modules



- Advantages
  - Simplicity – not only is it easy to understand, form already used with DMA
  - Flexibility – adding processor involves simple addition of processor to bus
  - Reliability – As long as arbitration does not involve single controller, then there is no single point of failure
- Disadvantages
  - Waiting for bus creates bottleneck, but this can be helped with a bigger investment in individual caches
  - Cache coherence policy must be used (usually hardware)

Central controller (arbitrator)

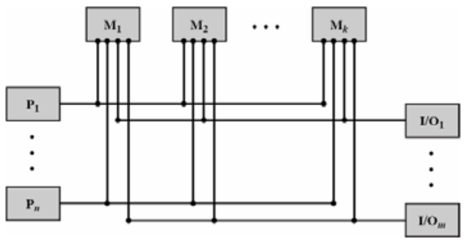
As opposed to time-shared where arbitration is distributed, a central controller monitors and manages access between all devices. The functions of the controller include:

- Funneling separate data streams between independent modules
- Buffering requests
- Performing arbitration and timing
- Passing status and control
- Performing cache update alerting
- Advantage – Uses same control, addressing, and data interfaces as typical processor, therefore, interfaces to modules remain the same
- Disadvantages
  - Very complex control unit
  - Control unit could end up being a bottleneck

Multi-port memory

Processors and I/O modules can share a multi-port memory through direct, independent connections that are much like a single processor's connection to a memory

- Logic internal to memory required to resolve conflicts
- Little or no modification to processors used to single processor applications or I/O modules required



- Advantages
  - Removing bus access bottleneck
  - Dedicate portions of memory to only one processor
  - Better security
  - Better recovery from faults
- Disadvantages
  - Complex memory logic
  - More PCB wiring
  - Write through policy should be used for caches

PROBLEM!

If we use caches to minimize the number of bus accesses required by multiple processors, how do we keep all of the caches up to date?

- Typically, there are one or two levels of cache associated with each processor – this is essential for performance
- Problem
  - Multiple copies of same data in different caches
  - Can result in an inconsistent view of memory

WRITE POLICY REVIEW

During our cache discussion, we presented the different methods for keeping caches up to date.

- Write back policy
  - Write goes only to cache
  - Main memory updated only when cache block is replaced
  - Can lead to inconsistency
- Write through policy
  - All writes made to cache and main memory
  - Inconsistencies can occur unless all caches monitor memory traffic

Cache Coherence Solutions

Software solutions attempt to avoid cache coherence problems by relying on the compiler and operating system. Compiler analyzes code to determine which items are cache-able and which are unsafe to cache. This information is then used to manage variables with methods such as:

- Marking shared variables as non-cacheable – this is too conservative
- Adding special instructions to enable/disable caching for variables. Then compiler can analyze code to determine safe periods for caching shared variables
- Benefits:
  - Overhead of detecting problems is transferred from run time to compile time
  - Design complexity is transferred from hardware to software
- Drawback – Software tends to make conservative decisions leading to inefficient cache utilization

Hardware solutions (cache coherence protocols) handle cache coherence problems real time.

- More efficient use of cache because it only deals w/problems when they occur
- Transparent to programmer and compiler
- Methods
  - Directory protocols
  - Snoopy protocols

Directory Protocols – Central Control for Management of Multiple Caches

When a central memory controller is used for cache coherence, the controller maintains directory of where all the copies of blocks are held and the state of each of the blocks, i.e., has it been updated, is it obsolete, etc. When a request is made for a block, the controller performs the necessary transfers.

Write Process for Directory Protocols

- When a processor needs to write to a block, it makes a request to the central controller
- Using its directory, the controller tells all other processors with copy of same data to invalidate
- Write is granted to requesting processor and that processor has exclusive rights to that data

Read Process for Directory Protocols – When a processor makes a request to read, the controller must issue a command to the processor with exclusive right to update (write back to) main memory.

Summary – The directory protocols method is effective in large scale systems with complex interconnection schemes. The problem is that it creates an ***additional communications burden*** and the ***central controller becomes a bottleneck***.

Snoopy Protocols – Distributed Control for Management of Multiple Caches

Another option is to distribute the cache coherence responsibility to the cache controllers of each processor.

- Cache recognizes that a line is shared
- Updates announced to other caches
- Suited to bus based multiprocessor
- Problem – it's possible to increase bus traffic to the point of canceling out benefits
- Two types of implementations: write invalidate and write update

Write Invalidate (a.k.a. MESI)

- Multiple readers, one writer
- When a write is required, command is issued and all other caches of the line are invalidated
- Writing processor then has exclusive (easy) access until the line is required by another processor
- A state is associated with every line: ***Modified, Exclusive, Shared, or Invalid*** (hence M.E.S.I.)

Write Update

- Multiple readers and multiple writers
- When a word is updated, the cache controller of the processor that updated it must distribute updated word to all other cache controllers.

Comparing Write Invalidate to Write Update

- Performance of these two implementations depends on number of caches and pattern of read/writes
- Some systems use adaptive protocols to use both methods
- Write invalidate (MESI) most common – Used in Pentium 4 and PowerPC systems

MESI Protocol

This protocol adds two bits to each line of a cache identifying the block contained in the line as:

- Modified – line in this cache is modified and ***only valid in this cache***
- Exclusive – line in this cache is same as that in memory (unmodified) and ***not present in any other cache***
- Shared – line in this cache is same as that in memory (unmodified), but ***may also be present in another cache***
- Invalid – line in this cache contains bad data, i.e., ***another processor has updated it and this processor has yet to load a valid copy***

Coherence can be maintained between L1 caches by utilizing write throughs. In other words, any time a block in an L1 cache is updated, the value is copied to the L2 which then handles the MESI protocol.

The table and state diagram below attempt to describe the operation of the MESI protocol.

Table 18.1 MESI Cache Line States from Stallings

	M Modified	E Exclusive	S Shared	I Invalid
This cache line is valid?	Yes	Yes	Yes	No
The copy in memory is…	out of date	valid	valid	can't tell
Do copies exist in other caches?	No	No	Maybe	Maybe
A write to this line…	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

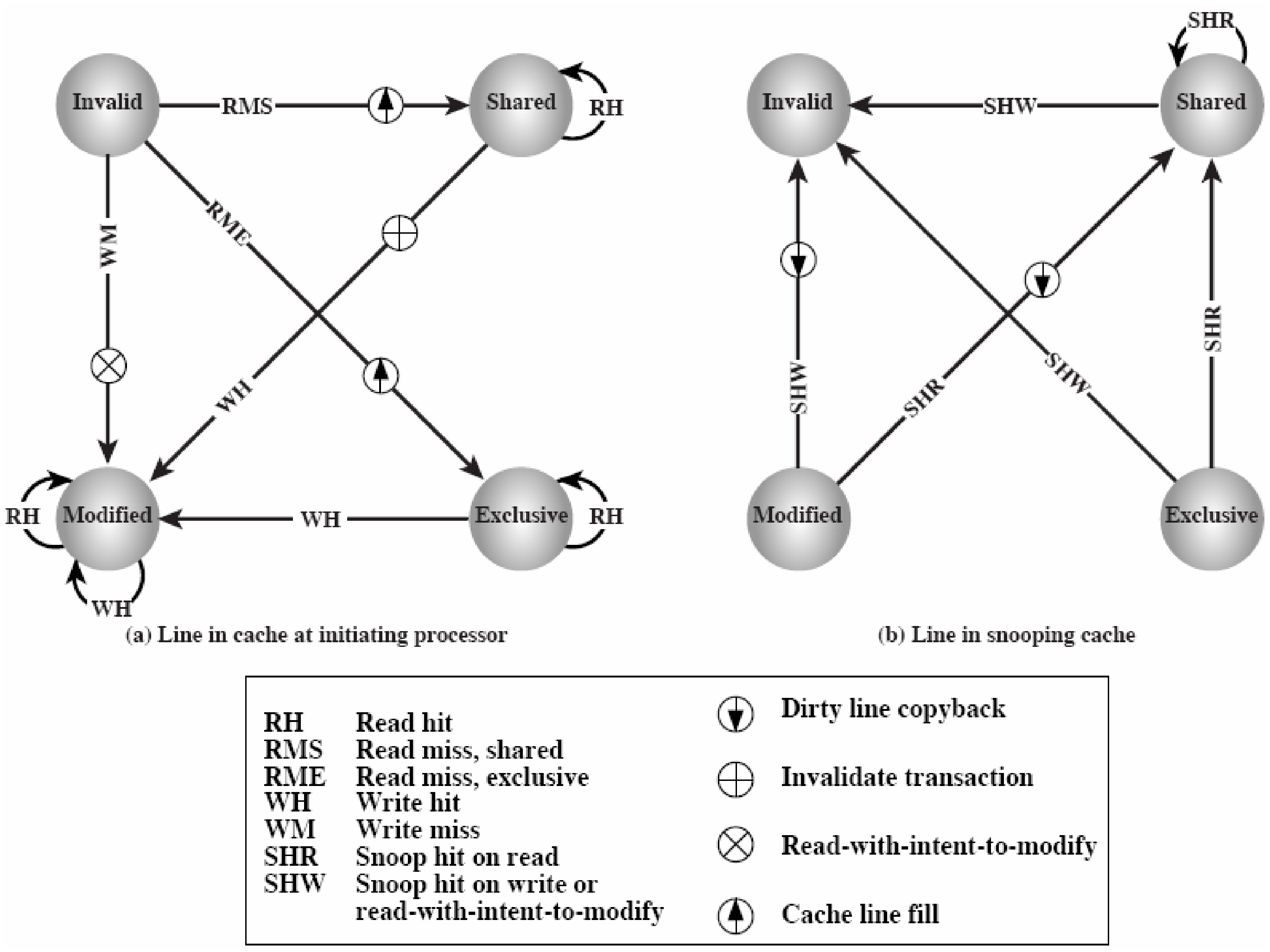


Figure 18.7 MESI State Transition Diagram from Stallings