CSCI 4717/5717
Computer Architecture

Topic: Symmetric Multiprocessors & Clusters

Reading: Stallings, Sections 18.1 through 18.4

Classifications of Parallel Processing

M. Flynn classified types of parallel processing in 1972 ("Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers) Types of Parallel Processor Systems (Figure 18.2)

- Single instruction, single data stream
- Single instruction, multiple data stream
- Multiple instruction, single data stream
- Multiple instruction, multiple data stream

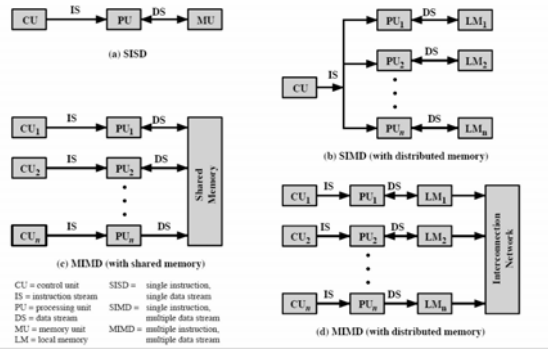
Classifications of Parallel Processing (continued)

- Single Instruction, Single Data Stream (SISD) – Single processor operates on a single instruction stream from a single memory (Uniprocessor)
- Single Instruction, Multiple Data Stream (SIMD) – Lockstep operation of multiple processors on single instruction memory with one data memory per processing element. (Vector/array processing)

Classifications of Parallel Processing (continued)

- Multiple Instruction, Single Data Stream (MISD) – Multiple processors execute different sequences of instructions on a single data set. Not commercially implemented
- Multiple Instruction, Multiple Data Stream (MIMD) – A set of processors simultaneously execute different instructions on different data sets.

Classifications of Parallel Processing (continued)



Multiple Instruction, Multiple Data Stream

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

Symmetric Multiprocessors (SMP)

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

Symmetric Multiprocessors (continued)

- All processors share access to I/O through either:
  - same channels
  - 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

SMP Advantages

Advantages only realized if O/S can provide parallelism

- Performance, but only if some work can 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

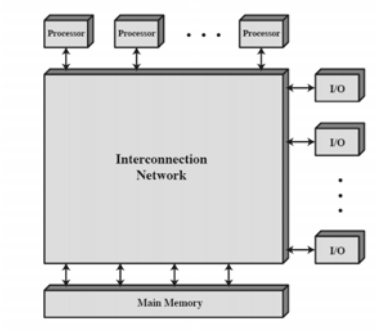
Organization of Tightly Coupled Multiprocessor

- 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

Organization of Tightly Coupled Multiprocessor (continued)

- Memory is often organized to provide simultaneous access to separate blocks of memory
- Bus
  - Time-shared or common bus
  - Central controller (arbitrator)
  - Multiport memory

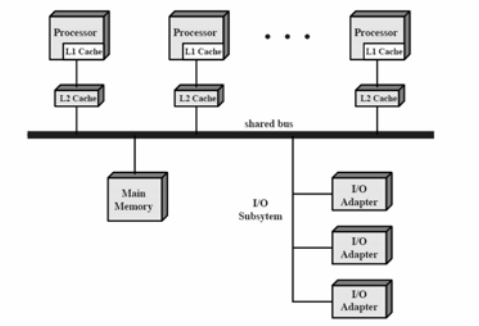
Organization of Tightly Coupled Multiprocessor (continued)



Time Shared Bus

- Structure and interface similar to single processor system (control, address, and data)
- Similar to DMA with single processor
- Following features provided
  - Addressing - distinguish modules on bus
  - Arbitration
    - Any module can be temporary master
    - Must have an arbitration scheme
  - 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

Time Shared Bus (continued)



Time Shared Bus (continued)

- 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

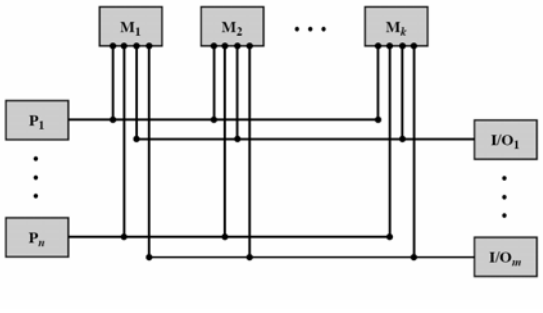
Time Shared Bus (continued)

- Disadvantages
  - Waiting for bus creates bottleneck
    - Can be helped with individual caches
    - Usually L1 and L2
  - Cache coherence policy must be used (usually hardware)

Multiport Memory

- Direct independent access of memory modules by each processor and I/O module
- Logic internal to memory required to resolve conflicts
- Little or no modification to processors used to single processor applications or I/O modules required

Multiport Memory



Multiport Memory (continued)

- 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

Central Control Unit

Functions

- Funnels separate data streams between independent modules
- Can buffer requests
- Performs arbitration and timing
- Pass status and control
- Perform cache update alerting

Central Control Unit (continued)

- Uses same control, addressing, and data interfaces as typical processor, therefore, interfaces to modules remain the same
- Disadvantages
  - Very complex control unit
  - Control unit is possible bottleneck

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

SMP Operating System Design Issues

- Simultaneous concurrent processes
  - O/S routines should be reentrant
  - O/S tables and other management structures must be expanded to handle multiple processes and processors
- Scheduling
  - More than just order now, also which processor gets a process
  - Any processor should be capable of scheduling too

SMP Operating System Design Issues (continued)

- 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.

Cache Coherence

- One or two levels of cache typically 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

- 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

Software Solutions

- Compiler and operating system deal with problem
- Overhead transferred to compile time
- Design complexity transferred from hardware to software
- Software tends to make conservative decisions leading to inefficient cache utilization

Software Solutions (continued)

- Marked shared variables as non-cacheable – Too conservative
- Instructions added to enable/disable caching for variables. Then compiler can analyze code to determine safe periods for caching shared variables

Hardware Solution

- A.K.A cache coherence protocols
- Dynamic recognition of potential problems at run time
- Because it only deals w/problem when it occurs, more efficient use of cache
- Transparent to programmer and compiler
- Methods
  - Directory protocols
  - Snoopy protocols

Directory Protocols – Central control

- Central memory controller maintains directory of:
  - where blocks are held
  - in which caches they are held
  - what state the data is in
- Appropriate transfers are performed by controller

Directory Protocols – Write Process

- Requests to write to a line are made to controller
- Using directory, 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
- Request to read from another processor forces controller to issue command to processor with exclusive rights to update (write back) main memory.

Directory Protocols (continued)

- Problems
  - Creates central bottleneck
  - Communication overhead
- Effective in large scale systems with complex interconnection schemes

Snoopy Protocols – Distributed control

- Distribute cache coherence responsibility among cache controllers
- Cache recognizes that a line is shared
- Updates announced to other caches
- Suited to bus based multiprocessor
- Problem – possible to increase bus traffic to point of canceling out benefits
- Two types of implementations:
  - Write Invalidate
  - 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 (cheap) access until line required by another processor
- A state is associated with every line
  - Modified
  - Exclusive
  - Shared
  - Invalid

Write Update (a.k.a. write broadcast)

- Multiple readers and writers
- Updated word is distributed to all other processors

Snoopy Protocols – Implementations

- 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 most common – Used in Pentium 4 and PowerPC systems

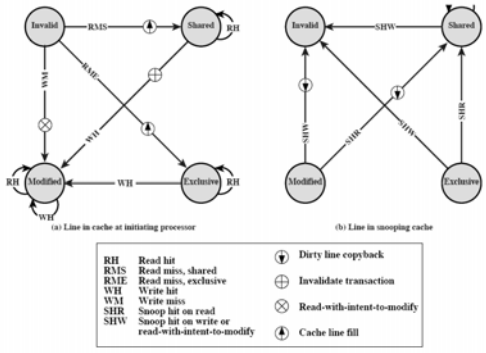
MESI Protocol

- Each line of a cache has associated with it two bits – four states
- 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) and may also be present in another cache
- Invalid – line in this cache contains bad data
- Write throughs from an L1 cache to an L2 cache makes it visible to the MESI protocol

MESI Protocol (continued)

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is…	out of date	valid	valid	—
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

MESI – State Transition Diagram



Clusters

- Defined
  - a group of interconnected, whole computers
  - working together as a unified computing resource
  - can create the illusion of being one machine
- Alternative to Symmetric Multiprocessing (SMP)
  - High performance
  - High availability
  - Server applications
- Each computer called a node

Cluster Computer Architecture

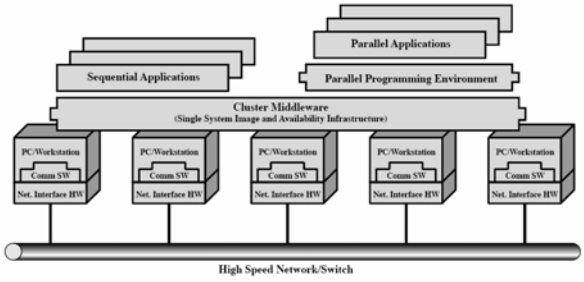


Figure 18.11 from Stallings, *Computer Organization & Architecture*

Cluster Benefits

- Absolute scalability – Almost limitless in terms of adding independent multiprocessing machines
- Incremental scalability – Can start out small and build as user acquires new machines

Cluster Benefits (continued)

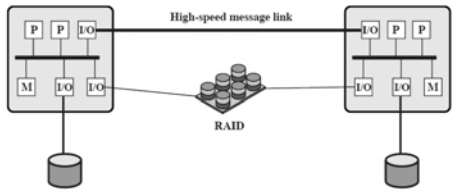
- High availability
  - Loss of one node only causes small decrement in performance
  - Software (middleware) handles fault tolerance automatically
- Superior price/performance
  - By using easily affordable building blocks, gets better performance at a lower price than a single large computer
  - Expanding design doesn't depend on PCB redesign

Cluster Configurations

- High-speed message link options/configurations
  - Dedicated LAN with at least one having connection to remote client
  - Shared LAN with other non-cluster machines
- Simplest way to classify clusters is based on whether computers share disk(s)
  - No shared disk – each machine has a local disk
  - Shared disk in addition to local disk – should use disk mirroring or RAID

Cluster Configurations – Standby Server with no Shared Disk

Cluster Configurations – Shared Disk



Cluster Configurations (continued)

- Secondary server – cluster functional classification
  - Passive Standby
    - Second computer will take over in the event of a failure on the part of the first
    - First computer sends "heartbeat"
    - Heartbeat stops, secondary takes over
    - Data must be shared or disks must be shared in order for secondary to take over database stuff too
  - Active Standby – Second computer participates in processing

Active Standby Configurations

- Separate Server
  - No shared disk
  - High performance and availability
  - Scheduling software is needed to assign client requests to servers to balance the load
  - If a computer fails in middle of application, another can take over
  - To do this, must have some method of copying data between at least neighboring computers

Active Standby Configurations (continued)

- Shared nothing
  - All computers share common RAID, but have partitions all to themselves.
  - If one fails, the cluster is reconfigured to reallocate failed computer's partitions
- Shared disk
  - All computers have access to all volumes of the same disk
  - Must use some type of locking facility to ensure that data can be accessed by one computer at a time

Comparison of Clustering Methods

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary:	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

Table 18.2 from Stallings, *Computer Organization & Architecture*

Cluster O/S Design Issues – Failure Management

- Two types of management: high availability and fault tolerant
- High availability
  - Independent processes
  - If one goes down, anything in progress is lost
  - Application layer must handle uncertainty of partially executed transactions
  - Process is taken over by next machine
- Fault tolerant
  - Redundancies
  - Mechanisms for handling partially executed transactions

Cluster O/S Design Issues – Failure Management (continued)

- Failover -- Switching applications & data from failed system to alternative within cluster
- Failback -- Restoration of applications and data to original system after problem is fixed

Cluster O/S Design Issues – Load balancing

- Incremental scalability of load with changes in number of nodes
- Automatically include new computers in scheduling
- Middleware needs to recognise that processes may switch between machines

Cluster O/S Design Issues – Parallelizing Computation

- Single application executing in parallel on a number of machines in cluster
- Three general approaches to the problem:
  - Parallelizing compiler
  - Parallelizing application
  - Parametric computing

Parallelizing Compiler

- Determines at compile time which parts can be executed in parallel
- Split off for different computers
- Performance depends on compiler

Parallelizing Application

- Application written to be parallel
- Message passing to move data between nodes
- Hard to program
- Performance depends on programmer
- Potential for best end result

Parametric computing

- If a problem is repeated execution of algorithm on different sets of data
- Example: simulation using different scenarios
- Depends on tools to organize/manage and execute

Cluster Middleware

Software installed on each node to enable cluster operation:

- Provides high availability through load balancing and failover control
- Creates unified image to user
  - Single point of entry – User logs onto cluster rather than a node
  - Single file hierarchy – User sees a single file structure
  - Single control point – single node acts as the interface to the user
  - Single virtual network visible to cluster nodes
  - Single memory space – programs are allowed to share variables across distributed memory
  - Single job management system – cluster assigns the jobs, not the user
  - Single user interface

Cluster Middleware (continued)

- Enhancement of availability
  - Single I/O space – I/O is accessible by any of the nodes regardless of the I/O device's location
  - Single process space
    - Processes are treated as if they are all operating on a single machine
    - This means that the process identification scheme should be uniform and independent of host node
  - Checkpointing – for recovery from a failure, each process should periodically save its state and intermediate variable values for failback
  - Process migration – to allow for load balancing

Cluster v. SMP

Positive points for both

- Both provide multiprocessor support to high demand applications.
- Both available commercially – SMP has been around longer

SMP benefits

- Easier to manage and configure since it is a single machine
- Closer to single processor systems for which nearly all applications are written
- Scheduling is main difference between SMP and single-processor system
- Less physical space
- Lower power consumption
- Well-established

Cluster benefits

- Superior incremental & absolute scalability
- Superior availability through redundancy of all components, not just processors
- Simpler to create from computers than SMP which is designed from PCB level
- With time, clusters are likely to dominate