

Classification of MIMD Architectures

■ Multiprocessors (shared memory)

- Any process can use usual load / store operations to access any memory word
- ✓ Simple communication between processes — shared memory locations
- ✓ Synchronization is well-understood, uses classical techniques (semaphores, etc.)
- ✗ Complex hardware — bus becomes a bottleneck with more than 10-20 CPUs

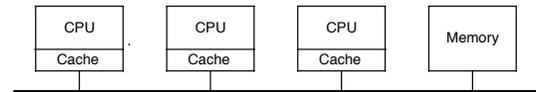
■ Multicomputers (distributed memory)

- Each CPU has its own private memory
- ✓ Simple hardware with network connection
- ✗ Complex communication — processes have to use message-passing, have to deal with lost messages, blocking, etc.
- RPCs help, but aren't perfect (no globals, can't pass large data structures, etc.)

1

Fall 2005, Lecture 18

Bus-Based UMA Multiprocessors



■ Symmetric Multiprocessor (SMP)

- Multiple CPUs (tens), one shared physical memory, connected by a bus

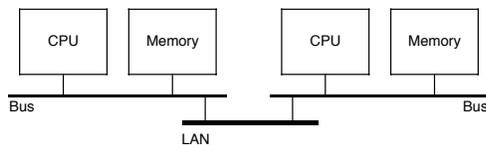
■ Caches must be kept consistent

- Each CPU has a “snooping” cache, which “snoops” what’s happening on the bus
 - On read hit, fetch data from local cache
 - On read miss, fetch data from memory, and store in local cache
 - Same data may be in multiple caches
 - (Write through) On write, store in memory and local cache
 - Other caches are snooping; if they have that word they invalidate their cache entry
 - After write completes, the memory is up-to-date and the word is only in one cache

2

Fall 2005, Lecture 18

NUMA Multiprocessors



■ NUMA = NonUniform Memory Access

■ NUMA multiprocessor

- Multiple CPUs, each with its own memory
- ...but the global memory space is shared
- Accesses to local memory locations are much faster (maybe 10x) than accesses to remote memory locations

■ Usually no hardware caching, so it matters where data is stored

- A reference to a remote page causes a hardware page fault, which traps to the OS, which decides whether or not to move the page to the local machine

3

Fall 2005, Lecture 18

Distributed Shared Memory (DSM) Overview

■ Basic idea (Kai Li, 1986)

- Collection of workstations, connected by a LAN, all sharing a single paged, virtual address space
- Each page is present on exactly one machine, and a reference to a local page is done in the usual fashion
- A reference to a remote page causes a hardware page fault, which traps to the OS, which sends a message to the remote machine to get the page; the faulting instruction can then complete

✓ To programmer, DSM machine looks like a conventional shared-memory machine

- Easy to modify old programs

✗ Poor performance — lots of pages being sent back and forth over the network

4

Fall 2005, Lecture 18

Comparison of Shared-Memory Systems

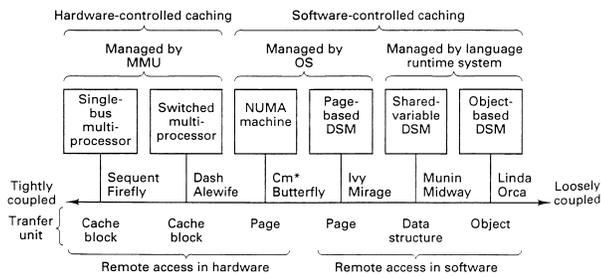


Figure from *Distributed Operating Systems*, Tanenbaum, Prentice Hall, 1995

- SMP = single bus multiprocessor
 - Hardware access to all of memory, hardware caching of blocks
- NUMA
 - Hardware access to all of memory, software caching of pages
- Distributed Shared Memory (DSM)
 - Software access to remote memory, software caching of pages

5

Fall 2005, Lecture 18

Consistency Models (Memory Coherence)

- Strict consistency (strongest model)
 - Value returned by a read operation is always the same as the value written by the most recent write operation
 - Requires global time to correctly order operations — not possible
- Sequential consistency (Lamport 1979)
 - All processes see all memory access operations in the same order
 - Order of operations doesn't matter, but all processes must see the same ordering
 - Global ordering of events, but not ordered by global time
 - Read operation may not return result of most recent write operation!
 - Running a program twice may give different results each time
 - If order matters, use semaphores!

6

Fall 2005, Lecture 18

Consistency Models (cont.)

- Causal consistency (Hutto & Ahamad 1990)
 - All processes see only potentially causally-related memory access operations in the same (correct) order
 - Two memory references are potentially causally-related if the first one potentially influences the second one
 - Write a variable, then read that variable
 - Read a variable, then write that variable
 - Write a variable, then write it again
 - Non-causally-related (“concurrent”) memory accesses may be seen in different orders by different processes
 - Write a variable, then write another totally separate variable
 - Must maintain a dependency graph to keep track of which operations are dependent on which other operations

7

Fall 2005, Lecture 18

Consistency Models (cont.)

- PRAM consistency (Lipton & Sandberg 1988)
 - All processes see all memory writes done by a particular process in the same (correct) order
 - Writes done by different processes may be seen in different orders by different processes
 - For a particular memory location, different processes may see the memory writes to that location in different orders
 - PRAM = pipelined RAM
 - Think of the process queuing or pipelining its writes to improve performance
- Processor consistency (Goodman 1990)
 - PRAM consistency + memory coherence
 - For a particular memory location, all processes see all memory writes to that location in the same order

8

Fall 2005, Lecture 18

Consistency Models (cont.)

- Weak consistency (Dubois 1988)
 - *Consistency need only apply to a group of memory accesses rather than individual memory accesses*
 - Use *synchronization accesses*
 - Synchronization accesses are sequentially consistent, and act as a barrier
 - Finish previous accesses, synch to propagate data, then start next accesses
 - Reads and writes between synch. accesses can occur in any order
- Release consistency (Gharachorloo 1990)
 - *Consistency applies only to lock acquire and release operations*
 - Use acquire before accessing a set of data, release afterwards
 - Finish previous accesses before release, start next accesses after acquire

9

Fall 2005, Lecture 18

Comparison of Consistency Models

- Differ in restrictiveness, implementation difficulty, ease of use, and performance
 - Strict consistency — most restrictive, but impossible to implement
 - Sequential consistency — intuitive semantics, but doesn't allow much concurrency
 - Used in DSM systems
 - Variation called *relaxed memory* is used in commercial memory systems (allows reads and writes to be reordered if they access different memory locations)
 - Causal, processor & PRAM consistency — allow more concurrency, but have non-intuitive semantics, and put extra burden on the programmer (to avoid doing things that require more consistency)
 - Weak & release consistency — intuitive semantics, but put extra burden on the programmer

10

Fall 2005, Lecture 18

Implementing Sequential Consistency in a Page-Based DSM

- Can a page move? ...be replicated?
- Nonreplicated, nonmigrating pages
 - All requests for the page have to be sent to the owner of the page
 - Easy to enforce sequential consistency — owner orders all access request
 - No concurrency
- Nonreplicated, migrating pages
 - All requests for the page have to be sent to the owner of the page
 - Each time a remote page is accessed, it migrates to the processor that accessed it
 - Easy to enforce sequential consistency — only processes on that processor can access the page
 - No concurrency

11

Fall 2005, Lecture 18

Implementing Sequential Consistency in a Page-Based DSM (cont.)

- Replicated, migrating pages
 - All requests for the page have to be sent to the owner of the page
 - Each time a remote page is accessed, it's copied to the processor that accessed it
 - Multiple read operations can be done concurrently
 - Hard to enforce sequential consistency — must invalidate (most common approach) or update other copies of the page during a write operation
- Replicated, nonmigrating pages
 - Replicated at fixed locations
 - All requests to the page have to be sent to one of the owners of the page
 - Hard to enforce sequential consistency — must update other copies of the page during a write operation

12

Fall 2005, Lecture 18

Granularity

■ Page-based DSM

- Single page — simple to implement
- Multiple pages — take advantage of locality of reference, amortize network overhead over multiple pages
 - Disadvantage — false sharing

■ Shared-variable DSM

- Share only those variables that are need by multiple processes
- Updating is easier, can avoid false sharing, but puts more burden on the programmer

■ Object-based DSM

- Retrieve not only data, but entire object — data, methods, etc.
- Have to heavily modify old programs