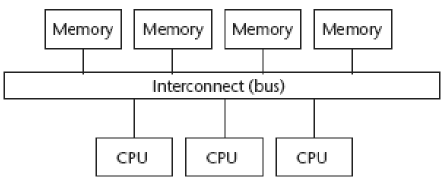
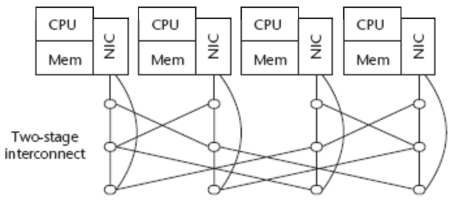
Memory Organization

Memory Parallelism

- One easy way to increase performance is to replicate entire computers and add way to communicate data
  - Advantage: increase mem bw
  - Cost: program complexity



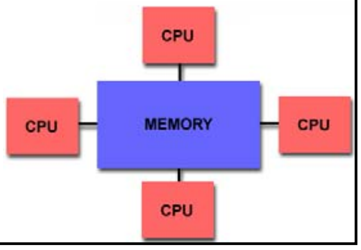
Shared memory system



Distributed memory system

Shared Memory

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space
- Multiple processors can operate independently but share the same memory resources.
  - Changes in a memory location effected by one processor are visible to all other processors.
  - Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** and ***NUMA***



Uniform Memory Access (UMA)

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
  - Identical processors
  - Equal access and access times to memory
- Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update.
  - Cache coherency is accomplished at the hardware level.

Non-Uniform Memory Access (NUMA)

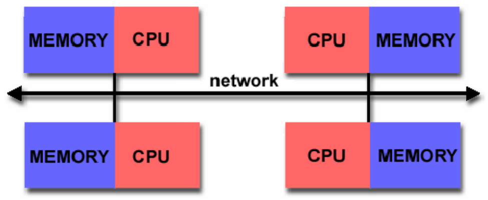
- Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - Not all processors have equal access time to all memories
- Memory access across link is slower
- If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

Advantages & Drawbacks

- Advantages:
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- Disadvantages:
  - Lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
  - Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory Architectures

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic.
- Distributed memory systems require a communication network to connect inter-processor memory



General Characteristics

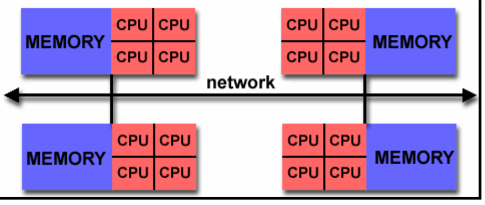
- Processors have their own local memory
  - Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently
  - Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated
  - Synchronization between tasks is likewise the programmer's responsibility
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet

Advantages & Drawbacks

- Advantages:
  - Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- Disadvantages:
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization.
  - Non-uniform memory access (NUMA) times

Hybrid Architectures

- The largest and fastest computers in the world today employ both shared and distributed memory architectures
  - The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
  - The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.
- Advantages and Disadvantages: whatever is common to both shared and distributed memory architectures.



Caches in Parallel Architectures

- Caches play key role in all cases
  - Reduce average data access time
  - Reduce bandwidth demands placed on shared interconnect
- But private processor caches create a problem
  - Copies of a variable can be present in multiple caches
  - A write by one processor may not become visible to others
    - They'll keep accessing stale value in their caches
  - *Cache coherence* problem
  - Need to take actions to ensure visibility

Shared Address Space: Cache Coherency Problems

- Coherency problem example

```
a = a + 1
b = 1
```

- Generic ISA translation:

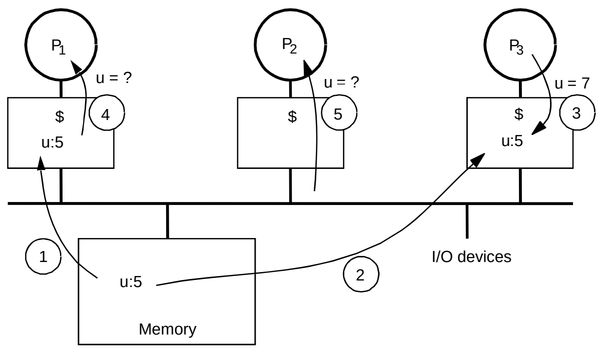
```
LOAD R12, %A10    ; Load a into register          <- loaded in cache of CPU0
ADD R12, #1       ; Add one to the value in R12
STORE R12, %A10   ; Store the result back into A  <- stored in cache of CPU0
```

- If CPU1 (on the same address space) performs:

```
if (b .eq. 0) goto 10
print *, a
```

- Ideally CPU1 sees the value after the increment as if the cache were not present, but this is not always true

Example of Cache Coherence Problem



- Processors see different values for u after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value when
  - Processes accessing main memory may see very stale value
- Unacceptable to programs, and frequent!

Cache Coherence

- Definition: reading a location should return latest value written (by any process)
- Easy in uniprocessors
  - Except for I/O: coherence between I/O devices and processors
  - But infrequent, so software solutions work
- Would like same to hold when processes run on different processors
  - E.g. as if the processes were interleaved on a uniprocessor
- But coherence problem much more critical in multiprocessors
- It’s worse than that: what is the “latest” value with independent processes?
  - Memory consistency models

Snooping Protocols

- Write invalidate (both for snooping and directory based)
  - One processor has exclusive access to a data item before writing it
  - It invalidates other cached copies on write (see below)
- Alternative: write update
  - Update all cached copies (too much overhead), not used in modern multiprocessors

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Implementation (1/3)

- Use a broadcast medium to invalidate
  - Acquire bus access
  - Broadcast the address to be invalidated
  - All processors continuously snoop on the bus watching for addresses, if an address is in the cache, it invalidates the data
- Bus enforces serialization of writes and invalidates

Implementation (2/3)

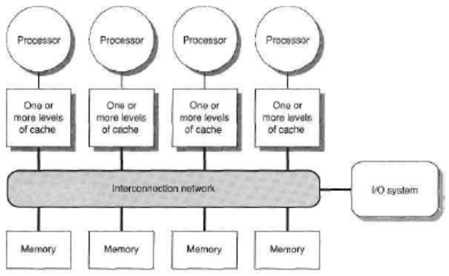
- After invalidation, a processor must locate the updated data upon cache miss
  - Easy in a write-through cache (most recent data always in memory)
  - Complex in a write-back cache (most recent data can be in cache)
- A snooping mechanism can be implemented
  - It snoops the bus for addresses, if it has a dirty copy of the requested cache block
    - It provides that block in response to a read
    - Cause memory access to be aborted
  - Additional complexity
    - Retrieve the block from the cache can be longer than memory if processors are in separate chips
    - But this approach is very common in CMP
  - In bus-based systems a line is dedicated to signal when invalidates are completed, in systems without a bus other techniques must be implemented…

Implementation (3/3)

- Track if a block is shared or not requires an extra bit on the cache tags
  - Decides whether a write generate an invalidate
  - Upon invalidation the block is marked as exclusive
  - Becomes shared again if another processor requests it
- Important: every bus transaction must check address tags
  - Interference with cache access
  - Solution:
    - Tags duplication
    - Redirect to L2 cache with inclusion property (every entry in L1 is present in L2): arbitration into L1 only when hit in L2

Limitations in SMM and Snooping Protocols

- Centralized resources are bottlenecks as memory demands increases
  - The bus must support also cache coherence traffic
- Solution: increase communication bw between proc and mem
  - Crossbar, p2p networks
  - Memory configured as multiple banks

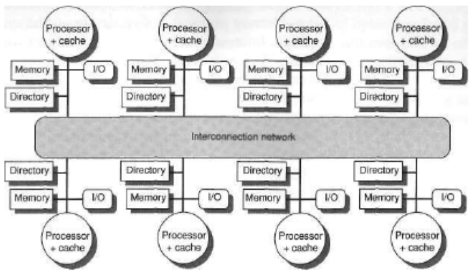

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Interconnection network
I/O system
Memory
Memory
Memory
Memory

AMD Opteron

- Memory is connected directly to each dual-core chip
- Up to four processor chips (8 in total)
- Cache coherency uses p2p network from one chip to the other three
  - Broadcast message to find shared copies (like snooping protocols)
  - Need explicit ack since comm medium is not shared (like directory protocol)
- Broadcast on writes remains a limit for scalability

Directory-Based Protocols

- Directory keeps state of each memory block that can be cached
  - Which caches have the copy
  - Whether they are dirty
- Directories are distributed along the memories



Example Protocol

- States of a cache block
  - Shared: on or more processors have the value cached and the memory is updated
  - Uncached: no processor has a copy of the block
  - Modified: exactly one processor (the owner) has a copy of the block and the memory is not updated
- The status must be tracked as well as the processors having the copies
  - Bit vector for each memory block, indicating which processor has a copy

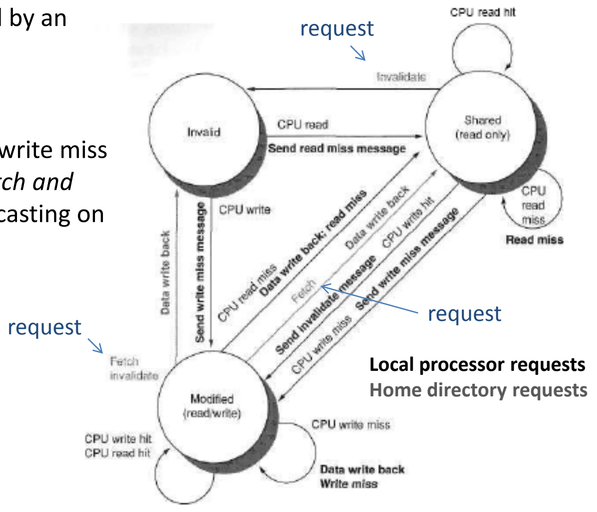
Implementation

- Type of nodes
  - Local node: where a request originates
  - Home node: where the memory location and the directory entry resides
  - Remote node: it has a copy of the cache block (exclusive or shared)

Message type	Source	Destination	Message contents	Function of this message
Read miss	local cache	home directory	P,A	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	local cache	home directory	P,A	Processor P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	local cache	home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	home directory	remote cache	A	Invalidate a shared copy of data at address A.
Fetch	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared. (read miss)
Fetch/invalidate	home directory	remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache. (write miss)
Data value reply	home directory	local cache	D	Return a data value from the home memory.
Data write back	remote cache	home directory	A,D	Write back a data value for address A.

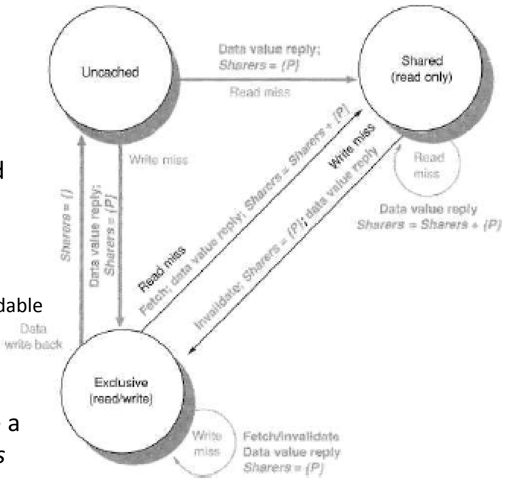
Example Directory Protocol (Local Cache Side)

- Protocol actions performed by an individual cache
  - Requests in gray
  - Actions in bold
- Differently from snooping, write miss operation is replaced by *fetch and invalidate* instead of broadcasting on the bus



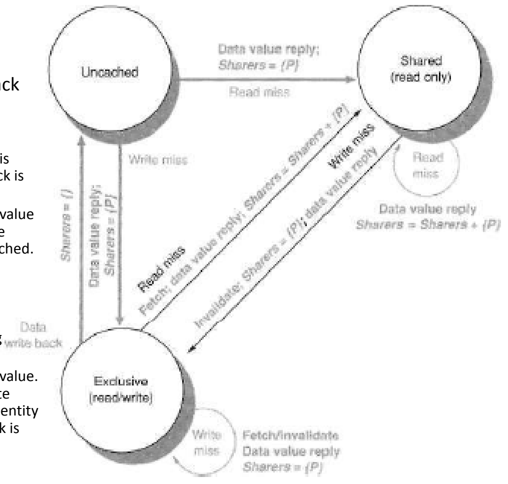
Example Directory Protocol (Directory Side)

- Message to a directory lead to:
  - Update dir status
  - Send additional messages
- Unlike snooping, the dir state indicates the state of all cached copies of a memory block
- Memory block can be:
  - Uncached by any node
  - Cached in multiple nodes and readable (shared)
  - Cached exclusively and writable in exactly one node
- The set of processors that have a copy of a block is called *Sharers*


Uncached
Shared (read only)
Exclusive (read/write)
Data value reply; *Sharers* = {*P*}
Read miss
Write miss
Data value reply; *Sharers* = {*P*}
Sharers = {}
Data write back
Read miss
Fetch; data value reply; *Sharers* = *Sharers* + {*P*}
Write miss
Invalidate; *Sharers* = {*P*}; data value reply
Read miss
Data value reply *Sharers* = *Sharers* + {*P*}
Write miss
Fetch/invalidate Data value reply *Sharers* = {*P*}

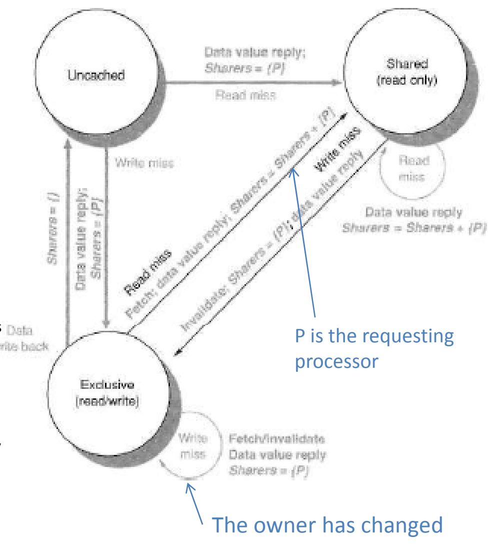
Example Directory Protocol (Directory Side)

- Directory receives three types of messages:
  - Read miss, write miss and data write back
- Block: uncached state
  - *Read miss* - The requesting processor is sent the requested data from memory, and the requestor is made the only sharing node. The state of the block is made shared
  - *Write miss* - The requesting processor is sent the value and becomes the sharing node. The block is made exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner
- Block: shared state
  - *Read miss* - The requesting processor is sent the requested data from memory, and the requesting processor is added to the sharing set.
  - *Write miss* - The requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, and the Sharers set is to contain the identity of the requesting processor. The state of the block is made exclusive.


Uncached
Shared (read only)
Exclusive (read/write)
Data value reply; *Sharers* = {*P*}
Read miss
Write miss
Data value reply; *Sharers* = {*P*}
Sharers = {}
Data write back
Read miss
Fetch; data value reply; *Sharers* = *Sharers* + {*P*}
Write miss
Invalidate; *Sharers* = {*P*}; data value reply
Read miss
Data value reply *Sharers* = *Sharers* + {*P*}
Write miss
Fetch/invalidate Data value reply *Sharers* = {*P*}

Example Directory Protocol (Directory Side)

- Block: exclusive state (memory is not updated)
  - *Read miss* - The owner processor is sent a data fetch message, which causes the state of the block in the owner's cache to transition to shared and causes the owner to send the data to the directory, where it is written to memory and sent back to the requesting processor. The identity of the requesting processor is added to the set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy).
  - *Data write back* - The owner processor is replacing the block and therefore must write it back. This write back makes the memory copy up to date (the home directory essentially becomes the owner), the block is now uncached, and the Sharers set is empty
  - *Write miss* - The block has a new owner. A message is sent to the old owner, causing the cache to invalidate the block and send the value to the directory, from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to the identity of the new owner, and the state of the block remains exclusive



NUMA, UMA & Cache Coherency

- UMA and NUMA are cache coherent shared memory architectures
- NUMA are also called:
  - Distributed shared memory systems
  - Virtual shared memory
  - Virtual distributed shared memory

References

- Introduction to Parallel Computing
  - https://computing.llnl.gov/tutorials/parallel_comp/#MemoryArch
- Dongarra, Foster, Fox, “The Sourcebook of Parallel Computing”, 2003.
- Hennessy Patterson, “Computer Architecture, A Quantitative Approach, 4 Ed.”, 2007.