Module 10: "Design of Shared Memory Multiprocessors"

Lecture 18: "Introduction to Cache Coherence"

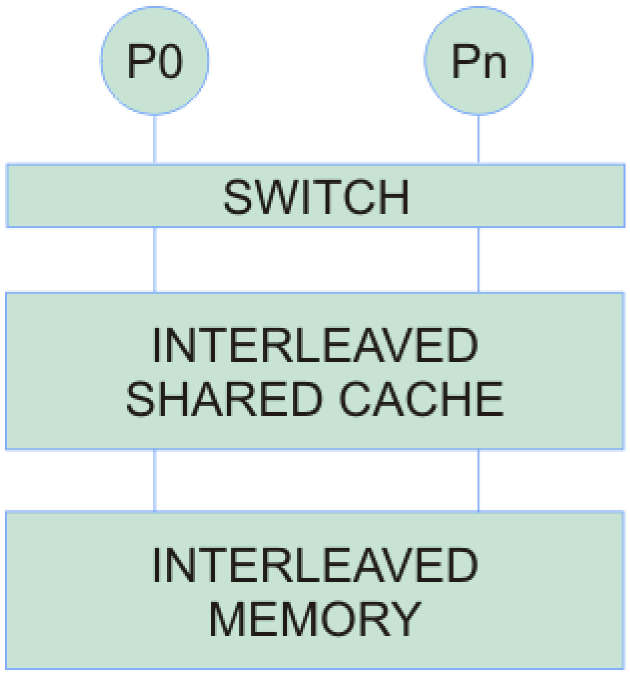
Shared Memory Multiprocessors

[From Chapter 5 of Culler, Singh, Gupta]

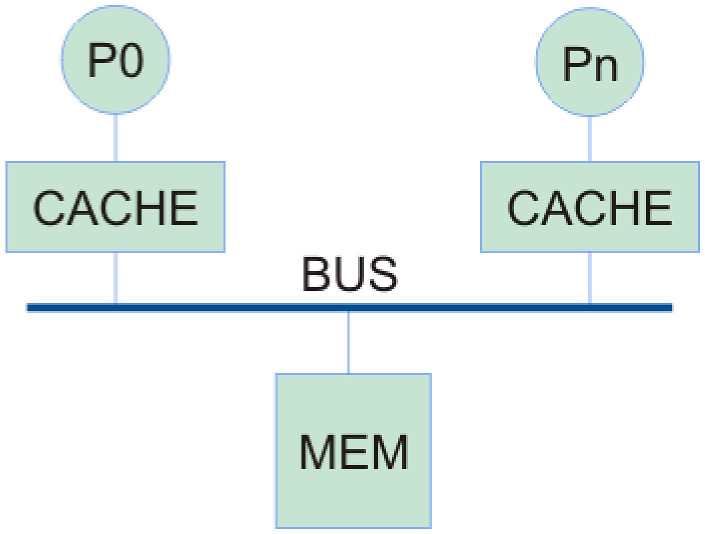
Previous Next

Four organizations

- Shared cache



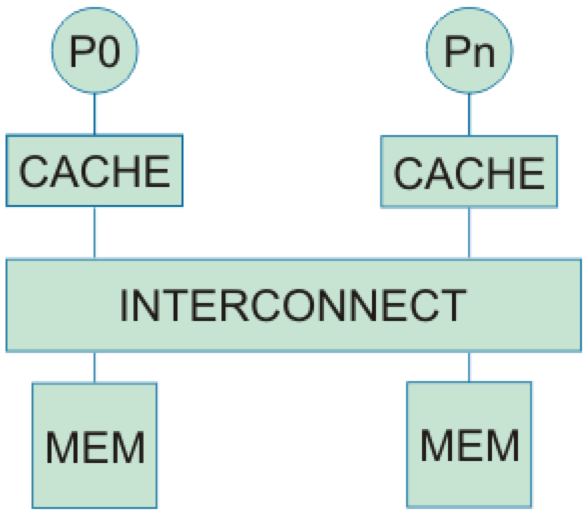
- The switch is a simple controller for granting access to cache banks
- Interconnect is between the processors and the shared cache
- Which level of cache hierarchy is shared depends on the design: Chip multiprocessors today normally share the outermost level (L2 or L3 cache)
- The cache and memory are interleaved to improve bandwidth by allowing multiple concurrent accesses
- Normally small scale due to heavy bandwidth demand on switch and shared cache
- Bus-based SMP



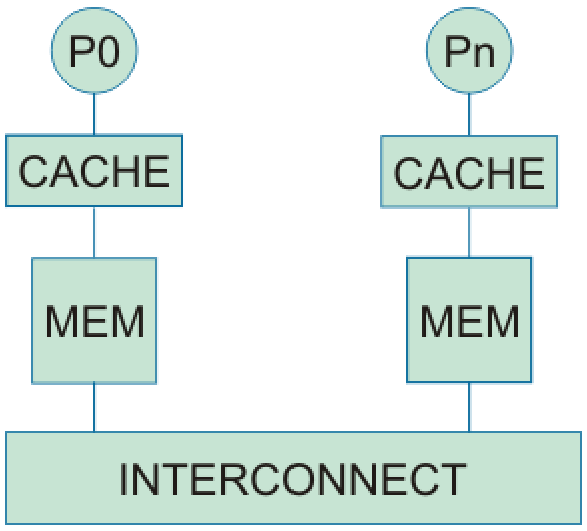
- Scalability is limited by the shared bus bandwidth
- Interconnect is a shared bus located between the private cache hierarchies and memory

controller

- The most popular organization for small to medium-scale servers
- Possible to connect 30 or so processors with smart bus design
- Bus bandwidth requirement is lower compared to shared cache approach
  - Why?
- Dancehall



- Better scalability compared to previous two designs
- The difference from bus-based SMP is that the interconnect is a scalable point-to-point network (e.g. crossbar or other topology)
- Memory is still symmetric from all processors
- Drawback: a cache miss may take a long time since all memory banks too far off from the processors (may be several network hops)
- Distributed shared memory



- The most popular scalable organization
- Each node now has local memory banks
- Shared memory on other nodes must be accessed over the network
  - Remote memory access
- Non-uniform memory access (NUMA)
  - Latency to access local memory is much smaller compared to remote memory
- Caching is very important to reduce remote memory access
- In all four organizations caches play an important role in reducing latency and bandwidth

requirement

  - If an access is satisfied in cache, the transaction will not appear on the interconnect and hence the bandwidth requirement of the interconnect will be less (shared L1 cache does not have this advantage)

- In distributed shared memory (DSM) cache and local memory should be used cleverly
- Bus-based SMP and DSM are the two designs supported today by industry vendors
  - In bus-based SMP every cache miss is launched on the shared bus so that all processors can see all transactions
  - In DSM this is not the case

Previous Next

Hierarchical design

- Possible to combine bus-based SMP and DSM to build hierarchical shared memory
  - Sun Wildfire connects four large SMPs (28 processors) over a scalable interconnect to form a 112p multiprocessor
  - IBM POWER4 has two processors on-chip with private L1 caches, but shared L2 and L3 caches (this is called a chip multiprocessor); connect these chips over a network to form scalable multiprocessors
- Next few lectures will focus on bus-based SMPs only

Cache Coherence

- Intuitive memory model
  - For sequential programs we expect a memory location to return the latest value written to that location
  - For concurrent programs running on multiple threads or processes on a single processor we expect the same model to hold because all threads see the same cache hierarchy (same as shared L1 cache)
  - For multiprocessors there remains a danger of using a stale value: in SMP or DSM the caches are not shared and processors are allowed to replicate data independently in each cache; hardware must ensure that cached values are coherent across the system and they satisfy programmers’ intuitive memory model

Example

- Assume a write-through cache i.e. every store updates the value in cache as well as in memory
  - P0: reads x from memory, puts it in its cache, and gets the value 5
  - P1: reads x from memory, puts it in its cache, and gets the value 5
  - P1: writes x=7, updates its cached value and memory value
  - P0: reads x from its cache and gets the value 5
  - P2: reads x from memory, puts it in its cache, and gets the value 7 (now the system is completely incoherent)
  - P2: writes x=10, updates its cached value and memory value
- Consider the same example with a writeback cache i.e. values are written back to memory only when the cache line is evicted from the cache
  - P0 has a cached value 5, P1 has 7, P2 has 10, memory has 5 (since caches are not write through)
  - The state of the line in P1 and P2 is M while the line in P0 is clean
  - Eviction of the line from P1 and P2 will issue writebacks while eviction of the line from P0 will not issue a writeback (clean lines do not need writeback)
  - Suppose P2 evicts the line first, and then P1
  - Final memory value is 7: we lost the store x=10 from P2

What went wrong?

- For write through cache
  - The memory value may be correct if the writes are correctly ordered
  - But the system allowed a store to proceed when there is already a cached copy

- Lesson learned: must invalidate all cached copies before allowing a store to proceed

- Writeback cache
  - Problem is even more complicated: stores are no longer visible to memory immediately
  - Writeback order is important
  - Lesson learned: do not allow more than one copy of a cache line in M state
- Need to formalize the intuitive memory model
  - In sequential programs the order of read/write is defined by the program order; the notion of “last write” is well-defined
  - For multiprocessors how do you define “last write to a memory location” in presence of independent caches?
  - Within a processor it is still fine, but how do you order read/write across processors?

Previous Next

Definitions

- Memory operation: a read (load), a write (store), or a read-modify-write
    - Assumed to take place atomically
- A memory operation is said to issue when it leaves the issue queue and looks up the cache
- A memory operation is said to perform with respect to a processor when a processor can tell that from other issued memory operations
    - A read is said to perform with respect to a processor when subsequent writes issued by that processor cannot affect the returned read value
    - A write is said to perform with respect to a processor when a subsequent read from that processor to the same address returns the new value

Ordering memory op

- A memory operation is said to complete when it has performed with respect to all processors in the system
- Assume that there is a single shared memory and no caches
    - Memory operations complete in shared memory when they access the corresponding memory locations
    - Operations from the same processor complete in program order: this imposes a partial order among the memory operations
    - Operations from different processors are interleaved in such a way that the program order is maintained for each processor: memory imposes some total order (many are possible)

Example

P0: x = 8; u = y; v = 9;

P1: r = 5; y = 4; t = v;

Legal total order:

x = 8; u = y; r = 5; y = 4; t = v; v = 9;

Another legal total order:

x = 8; r = 5; y = 4; u = y; v = 9; t = v;

- “Last” means the most recent in some legal total order
- A system is coherent if
    - Reads get the last written value in the total order
    - All processors see writes to a location in the same order

Cache coherence

- Formal definition
    - A memory system is coherent if the values returned by reads to a memory location during an execution of a program are such that all operations to that location can form a hypothetical total order that is consistent with the serial order and has the following two properties:

1. Operations issued by any particular processor perform according to the issue order
2. The value returned by a read is the value written to that location by the last write in the total order

- Two necessary features that follow from above:

A. Write propagation: writes must eventually become visible to all processors
B. Write serialization: Every processor should see the writes to a location in the same order (if I see w1 before w2, you should not see w2 before w1)

Bus-based SMP

- Extend the philosophy of uniprocessor bus transactions
    - Three phases: arbitrate for bus, launch command (often called request) and address, transfer data
    - Every device connected to the bus can observe the transaction
    - Appropriate device responds to the request
    - In SMP, processors also observe the transactions and may take appropriate actions to guarantee coherence
    - The other device on the bus that will be of interest to us is the memory controller (north bridge in standard mother boards)
    - Depending on the bus transaction a cache block executes a finite state machine implementing the coherence protocol

Previous Next

Snoopy protocols

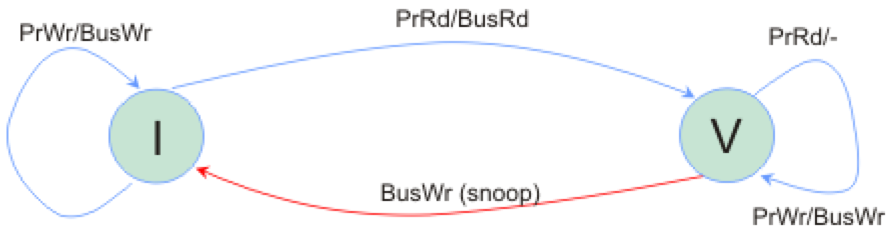
- Cache coherence protocols implemented in bus-based machines are called snoopy protocols
  - The processors snoop or monitor the bus and take appropriate protocol actions based on snoop results
  - Cache controller now receives requests both from processor and bus
  - Since cache state is maintained on a per line basis that also dictates the coherence granularity
  - Cannot normally take a coherence action on parts of a cache line
  - The coherence protocol is implemented as a finite state machine on a per cache line basis
  - The snoop logic in each processor grabs the address from the bus and decides if any action should be taken on the cache line containing that address (only if the line is in cache)

Write through caches

- There are only two cache line states
  - Invalid (I): not in cache
  - Valid (V): present in cache, may be present in other caches also
- Read access to a cache line in I state generates a BusRd request on the bus
  - Memory controller responds to the request and after reading from memory launches the line on the bus
  - Requester matches the address and picks up the line from the bus and fills the cache in V state
  - A store to a line always generates a BusWr transaction on the bus (since write through); other sharers either invalidate the line in their caches or update the line with new value

State transition

- The finite state machine for each cache line:



- On a write miss no line is allocated
  - The state remains at I: called write through write no-allocated
- A/B means: A is generated by processor, B is the resulting bus transaction (if any)
- Changes for write through write allocate?

Ordering memory op

- Assume that the bus is atomic
  - It takes up the next transaction only after finishing the previous one
- Read misses and writes appear on the bus and hence are visible to all processors

- What about read hits?
  - They take place transparently in the cache
  - But they are correct as long as they are correctly ordered with respect to writes
  - And all writes appear on the bus and hence are visible immediately in the presence of an atomic bus
- In general, in between writes reads can happen in any order without violating coherence
  - Writes establish a partial order

Write through is bad

- High bandwidth requirement
  - Every write appears on the bus
  - Assume a 3 GHz processor running application with 10% store instructions, assume CPI of 1
  - If the application runs for 100 cycles it generates 10 stores; assume each store is 4 bytes; 40 bytes are generated per 100/3 ns i.e. BW of 1.2 GB/s
  - A 1 GB/s bus cannot even support one processor
  - There are multiple processors and also there are read misses
- Writeback caches absorb most of the write traffic
  - Writes that hit in cache do not go on bus (not visible to others)
  - Complicated coherence protocol with many choices

Previous Next