BITS Pilani
Pilani Campus

Cache Coherence

K Hari Babu
Department of Computer Science & Information Systems

Symmetric Shared-Memory Architectures

- The use of large, multilevel caches reduce the memory band-width demands of a processor
- Symmetric shared-memory machines support the caching of both shared and private data
  - Private data are used by a single processor
  - Shared data are used by multiple processors
    - providing communication among the processors through reads and writes
- When private data is cached, behavior is same as uniprocessor
- When shared data is cached,
  - It is replicated in caches of multiple processors. Reduces latency, requirement of memory bandwidth, and contention for the same data item
  - BUT introduces a new problem: **cache coherence**

What Is Multiprocessor Cache Coherence?

- The view of memory held by two different processors is through their individual caches

Time	Event	Cache Contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

- The cache coherence problem for a single memory location (X), read and written by two processors (A and B)
  - Assume that neither cache contains the variable and that X has the value 1
  - Also assume a write-through cache; a write-back cache adds some additional but similar complications
  - After the value of X has been written by A, A’s cache and the memory both contain the new value, but B’s cache does not, and if B reads the value of X, it will receive 1!

Shared Memory Systems – Cache Coherence

BITS Pilani

- Cache Coherence is a problem in Shared Memory Multiprocessor (UMA) systems:
  - Multiple processors may read copies of the same data and store it in private caches
  - Any changes in private caches have to be synchronized
- Cache Coherence in multi-core systems
  - This is a specific case of the general problem:
    - some levels of cache are private (per core),
    - some are locally shared (within a group of cores – possibly on chip),
  - some are globally shared (across all cores – possibly on chip)
  - Some multi-core systems do not provide cache coherence:
    - e.g. IBM Cell

Coherence

BITS Pilani

- A memory system is coherent if
  1. A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.
  2. A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
  3. Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors.
  - The first property simply preserves program order—we expect this property to be true even in uniprocessors
  - The second property defines the notion of what it means to have a coherent view of memory: If a processor could continuously read an old data value, we would clearly say that memory was incoherent.

Coherence

BITS Pilani

- A memory system is coherent if
  1. A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.
  2. A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
  3. Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors.
  - Third property ensures that all writes to the same location are seen in the same order by all processes. This property is called write-serialization.

Coherence and Consistency

- Coherence defines the behavior of reads and writes to the same memory location
- Consistency defines the behavior of reads and writes with respect to accesses to other memory locations
- Two conditions to make coherence happen
  - First, a write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
  - Second, the processor does not change the order of any write with respect to any other memory access
- These two conditions mean that if a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A
- These restrictions allow the processor to reorder reads, but forces the processor to finish a write in program order

Basic Schemes for Enforcing Coherence

- In a coherent multiprocessor, the caches provide both migration and replication of shared data items
- Multiprocessor caches support
  - Data Migration:
    - moving a data item to a local cache reduces the latency to access a remote memory and the bandwidth demand on the shared memory
  - Data Replication
    - allowing multiple copies of a shared data items to be copied in multiple local caches reduces both latency and memory contentions
- Migration and replication are critical for performance, but give rise to coherence issues

Cache Coherence Protocols

- Small-scale multiprocessors adopt a hardware solution by using a protocol to maintain coherent caches
  - Key is tracking the state of any sharing of a data block
- There are two classes of protocols using different techniques to track the sharing status
- Snooping
  - Every cache that has a copy of a data from a physical block, is responsible of keeping track of its sharing status by monitoring (snooping) shared-memory bus transactions
- Directory based
  - The sharing status of each physical block is keptin a unique directory

Snooping

- Write invalidate
  - Provides a processor exclusive access to a data item before it writes that item
  - This is done by invalidating (on a write) other copies of the data item to be written
  - It is by far the most common protocol, both for snooping and for directory schemes
  - Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs: All other cached copies of the item are invalidated.
- Write update(or write broadcast)
  - When a data item is written, all cached copies of that item are updated

Snooping

BITS Pilani

- Write invalidate
  - To see how this protocol ensures coherence, consider a write followed by a read by another processor: Since the write requires exclusive access, any copy held by the reading processor must be invalidated
  - When the read occurs, it misses in the cache and is forced to fetch a new copy of the data
  - For a write, we require that the writing processor have exclusive access, preventing any other processor from being able to write simultaneously
  - For the other processor to complete its write, it must obtain a new copy of the data, which must now contain the updated value.
    - Therefore, this protocol enforces write serialization.
- Write update(or write broadcast)
  - When a data item is written, all cached copies of that item are updated
    - consumes considerably more bandwidth, therefore not used in practise

Directory Protocol

- A directory keeps the state of every blok that may be cached
- Information in the directory includes
  - Which caches have the block
  - Whether it is dirty …
- Within multicore system L3 cache is shared and easy to implement directory there
  - Simply keep a bit vector of the size equal to the number of cores for each L3 block
  - The bit vector indicates which private caches may have copies of a block in L3
  - Invalidations are sent only to those private caches
  - This is the scheme used in Intel i7

Cache Misses

- The uniprocessor cache misses are classified into capacity, compulsory, and conflict
- Similarly, the misses that arise from inter-processor communication(coherence misses), can be broken into two separate sources
  - True sharing misses
    - The first write by a processor to a shared cache block causes an invalidation to establish ownership of that block. When another processor attempts to read a modified word in that cache block, a miss occurs and the resultant block is transferred. These misses are classified as true sharing misses since they directly arise from the sharing of data among processors
  - False sharing
    - Arises when there are multiple words per cache block/line
    - With a single valid bit per cache block, false sharing occurs when a cache line is invalidated because some word in the block, other than the one being read, is written into

False Sharing

BITS Pilani

- Assume that words x1 and x2 are in the same cache block, which is in the shared state in the caches of both P1 and P2.
  - Any miss that would occur if the block size were one word is designated a true sharing miss

Time	P1	P2	
1	Write x1		This event is a true sharing miss, since x1 was read by P2 and needs to be invalidated from P2
2		Read x2	This event is a false sharing miss, since x2 was invalidated by the write of x1 in P1, but that value of x1 is not used in P2
3	Write x1		This event is a false sharing miss, since the block containing x1 is marked shared due to the read in P2, but P2 did not read x1.
4		Write x2	This event is a false sharing miss for the same reason as step 3.
5	Read x2		This event is a true sharing miss, since the value being read was written by P2.

References

BITS Pilani

- J. Hennessey and D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 5th Edition (Chapter 5)

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You