Bus snooping

Bus snooping or **bus sniffing** is a scheme by which a coherency controller (snooper) in a cache (a **snoopy cache**) monitors or snoops the bus transactions, and its goal is to maintain a cache coherency in distributed shared memory systems. A cache containing a coherency controller (snooper) is called a snoopy cache. This scheme was introduced by Ravishankar and Goodman in 1983.[1]

Contents

How it works

When specific data is shared by several caches and a processor modifies the value of the shared data, the change must be propagated to all the other caches which have a copy of the data. This change propagation prevents the system from violating cache coherency. The notification of data change can be done by bus snooping. All the snoopers monitor every transaction on a bus. If a transaction modifying a shared cache block appears on a bus, all the snoopers check whether their caches have the same copy of the shared block. If a cache has a copy of the shared block, the corresponding snooper performs an action to ensure cache coherency. The action can be a flush (https://forums.xilinx.com/t5/Embedded-Development-Tools/what-is-the-difference-between-cache-invalidate-and-cache-flush/td-p/74654) or an invalidation of the cache block. It also involves a change of cache block state depending on the cache coherence protocol.[2]

Types of snooping protocols

There are two kinds of snooping protocols depending on the way to manage a local copy of a write operation:

Write-invalidate

When a processor writes on a shared cache block, all the shared copies in the other caches are invalidated through bus snooping. This method ensures that only one copy of a datum can be exclusively read and written by a processor. All the other copies in other caches are invalidated. This is the most commonly used snooping protocol. MSI, MESI, MOSI, MOESI, and MESIF protocols belong to this category.

Write-update

When a processor writes on a shared cache block, all the shared copies of the other caches are updated through bus snooping. This method broadcasts a write data to all caches throughout a bus. It incurs larger bus traffic than write-invalidate protocol. That is why this method is uncommon. Dragon and firefly protocols belong to this category.[3]

Implementation

One of the possible implementations is as follows:

The cache would have three extra bits:

- *V* – valid
- *D* – dirty bit, signifies that data in the cache is not the same as in memory
- *S* – shared

Each cache line is in one of the following states: "dirty" (has been updated by local processor), "valid", "invalid" or "shared". A cache line contains a value, and it can be read or written. Writing on a cache line changes the value. Each value is either in main memory (which is very slow to access), or in one or more local caches (which is fast). When a block is first loaded into the cache, it is marked as "valid".

On a read miss to the local cache, the read request is broadcast on the bus. All cache controllers (http://www.cl.cam.ac.uk/research/srg/bluebook/32/mdh9/node6.html) monitor the bus. If one has cached that address and it is in the state "dirty", it changes the state to "valid" and sends the copy to requesting node. The "valid" state means that the cache line is current. On a local write miss (an attempt to write that value is made, but it's not in the cache), bus snooping ensures that any copies in other caches are set to "invalid". "Invalid" means that a copy used to exist in the cache, but it is no longer current.

For example, an initial state might look like this:

```
Tag  | ID | V | D | S
---------------------
1111 | 00 | 1 | 0 | 0
0000 | 01 | 0 | 0 | 0
0000 | 10 | 1 | 0 | 1
0000 | 11 | 0 | 0 | 0
```

After a write of address 1111 00, it would change into this:

```
Tag  | ID | V | D | S
---------------------
1111 | 00 | 1 | 1 | 0
0000 | 01 | 0 | 0 | 0
0000 | 10 | 1 | 0 | 1
0000 | 11 | 0 | 0 | 0
```

The caching logic monitors the bus and detects if any cached memory is requested. If the cache is dirty and shared and there is a request on the bus for that memory, a dirty snooping element will supply the data to the requester. At that point either the requester can take on responsibility for the data (marking the data as dirty), or memory can grab a copy (the memory is said to have "snarfed" the data) and the two elements go to the shared state. [4]

When invalidating an address marked as dirty (i.e. one cache would have a dirty address and the other cache is writing) then the cache will ignore that request. The new cache will be marked as dirty, valid and exclusive and that cache will now take responsibility for the address.[1]

Benefit

The advantage of using bus snooping is that it is faster than directory based coherency mechanism. The data being shared is placed in a common directory that maintains the coherence between caches in a directory-based system. Bus snooping is normally faster if there is enough bandwidth, because all transactions are a request/response seen by all processors.[2]

Drawback

The disadvantage of bus snooping is limited scalability. Frequent snooping on a cache causes a race with an access from a processor, thus it can increase cache access time and power consumption. Each of the requests has to be broadcast to all nodes in a system. It means that the size of the (physical or logical) bus and the bandwidth it provides must grow, as the system becomes larger.[2] Since the bus snooping does not scale well, larger cache coherent NUMA (ccNUMA) systems tend to use directory-based coherence protocols.

Snoop filter

When a bus transaction occurs to a specific cache block, all snoopers must snoop the bus transaction. Then the snoopers look up their corresponding cache tag to check whether it has the same cache block. In most cases, the caches do not have the cache block since a well optimized parallel program doesn’t share much data among threads. Thus the cache tag lookup by the snooper is usually an unnecessary work for the cache who does not have the cache block. But the tag lookup disturbs the cache access by a processor and incurs additional power consumption.

One way to reduce the unnecessary snooping is to use a snoop filter. A snoop filter determines whether a snooper needs to check its cache tag or not. A snoop filter is a directory-based structure and monitors all coherent traffic in order to keep track of the coherency states of cache blocks. It means that the snoop filter knows the caches that have a copy of a cache block. Thus it can prevent the caches that do not have the copy of a cache block from making the unnecessary snooping. There are three types of filters depending on the location of the snoop filters. One is a source filter that is located at a cache side and performs filtering before coherence traffic reaches the shared bus. Another is a destination filter that is located at receiver caches and prevents unnecessary cache-tag look-ups at the receiver core, but this type of filtering fails to prevent the initial coherence message from the source. Lastly, in-network filters prune coherence traffic dynamically inside the shared bus.[5] The snoop filter is also categorized as inclusive and exclusive. The inclusive snoop filter keeps track of the presence of cache blocks in caches. However, the exclusive snoop

filter monitors the absence of cache blocks in caches. In other words, a hit in the inclusive snoop filter means that the corresponding cache block is held by caches. On the other hand, a hit in the exclusive snoop filter means that no cache has the requested cache block.[6]

References

1. Ravishankar, Chinya; Goodman, James (February 28, 1983). *Cache Implementation for Multiple Microprocessors* (http://www.cs.ucr.edu/~ravi/Papers/NWConf/ravishankar_83.pdf) (PDF). pp. 346–350.
2. Yan Solihin (2016). *Fundamentals of Parallel Computer Architecture*. pp. 239–246.
3. Hennessy, John L; Patterson, David A. (2011). *Computer Architecture: A Quantitative Approach* (https://archive.org/details/computerarchitec00henn_754). pp. 355 (https://archive.org/details/computerarchitec00henn_754/page/n380)–356. ISBN 978-0123838728.
4. Siratt, Adrem. "What is Cache Coherence?" (https://www.easytechjunkie.com/what-is-cache-coherence.htm). EasyTechJunkie. Retrieved 2021-12-01.
5. Agarwal, N.; Peh, L.; Jha, N. K. (December 2009). "In-Network Coherence Filtering: Snoopy coherence without broadcasts" (https://ieeexplore.ieee.org/document/5375372). *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*: 232–243. doi:10.1145/1669112.1669143 (https://doi.org/10.1145%2F1669112.1669143). hdl:1721.1/58870 (https://hdl.handle.net/1721.1%2F58870).
6. Ulfsnes, Rasmus (June 2013). *Design of a Snoop Filter for Snoop-Based Cache Coherency Protocols*. Norwegian University of Science and Technology.

External links

- Jim Plusquellic. Centralized Shared-Memory Architectures (http://www.ece.unm.edu/~jimp/611/slides/chap8_2.html).
- Snoop filter (http://www.eecg.toronto.edu/~moshovos/filter/doku.php?id=start).
- http://www.icsa.inf.ed.ac.uk/research/groups/hase/models/coherence/index.html
- http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/0650/bks/SGI_Developer/books/T_IRIX_Prog/sgi_html/ch01.html