WIKIPEDIA

Directory-based cache coherence

In computer engineering, **directory-based cache coherence** is a type of cache coherence mechanism, where directories are used to manage caches in place of bus snooping. Bus snooping methods scale poorly due to the use of broadcasting. These methods can be used to target both performance and scalability of directory systems.[1]

Contents

Full bit vector format

In the full bit vector format, for each possible cache line in memory, a bit is used to track whether every individual processor has that line stored in its cache.[2] The full bit vector format is the simplest structure to implement, but the least scalable.[1] The SGI Origin 2000 uses a combination of full bit vector and coarse bit vector depending on the number of processors.[3]

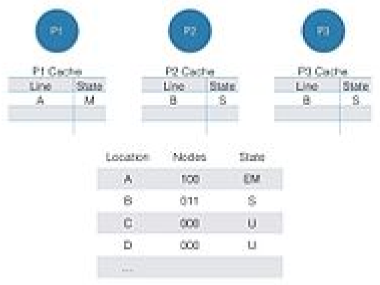

Diagram of full bit vector directory format, where E=Exclusive, S=Shared, M=Modified, and U=Uncached

Each directory entry must have 1 bit stored per processor per cache line, along with bits for tracking the state of the directory. This leads to the total size required being *(number of processors)×number of cache lines*, having a storage overhead ratio of *(number of processors)/(cache block size×8)*.

It can be observed that directory overhead scales linearly with the number of processors. While this may be fine for a small number of processors, when implemented in large systems the size requirements for the directory becomes excessive. For example, with a block size of 32 bytes and 1024 processors, the storage overhead ratio becomes 1024/(32×8) = 400%.[2]

Coarse bit vector format

The coarse bit vector format has a similar structure to the full bit vector format, though rather than tracking one bit per processor for every cache line, the directory groups several processors into nodes, storing whether a cache line is stored in a node rather than a line. This improves size requirements at the expense of bus traffic saving (processors per node)×(total lines) bits of space.[3] Thus the ratio overhead is the same, just replacing number of processors with number of processor groups. When a bus request is made for a cache line that one processor in the group has, the directory broadcasts the signal into every processor in the node rather than just the caches that contain it, leading to unnecessary traffic to nodes that do not have the data cached.[2]

In this case the directory entry uses 1 bit for a group of processors for each cache line. For the same example as Full Bit Vector format if we consider 1 bit for 8 processors as a group, then the storage overhead will be 128/(32×8)=50%. This is a significant improvement over the Full Bit Vector format.

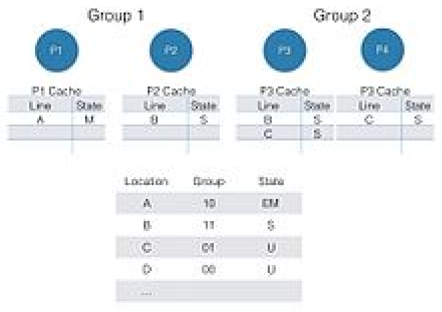

Diagram of coarse bit vector directory format

Sparse directory format

A cache only stores a small subset of blocks in main memory at a particular time. Hence most of the entries in the directory will belong to uncached blocks. In the sparse directory format the wastage is reduced by storing only the cached blocks in the directory.[2] Consider a processor with a cache size of 64KB with a block size of 32 bytes and the main memory size to be 4MB. The maximum number of entries that the directory can have in the sparse directory format is 2048. If the directory has an entry for all the blocks in the memory the number of entries in the directory will be 131072. Thus it is evident that the storage improvement provided by sparse directory format is very significant.

Number-balanced binary tree format

In this format the directory is decentralised and distributed among the caches that share a memory block. Different caches that share a memory block are arranged in the form of a binary tree. The cache that accesses a memory block first is the root node. Each memory block has the root node information (HEAD) and Sharing counter field (SC). The SC field has the number of caches that share the block. Each cache entry has pointers to the next sharing caches known as L-CHD and R-CHD. A condition for this directory is that the binary tree should be number balanced, i.e the number of nodes in the left sub tree must be equal to or one greater than the number of nodes in the right subtree. All the subtrees should also be number balanced.[4]

Chained directory format

In this format the memory holds the directory pointer to the latest cache that accessed the block and each cache has the pointer to the previous cache that accessed the block. So when a processor sends a write request to a block in memory, the processor sends invalidations down the chain of pointers. In this directory when a cache block is replaced we need to traverse the list in order to change the directory which increases latency. In order to prevent this doubly linked lists are widely used now in which each cached copy has pointers to previous and the next cache that accesses the block.[5]

Limited pointer format

The limited pointer format uses a set number of pointers to track the processors that are caching the data. When a new processor caches a block, a free pointer is chosen from a pool to point to that processor. There are a few options for handling cases when the number of sharers exceeds the number of free pointers. One method is to invalidate one of the sharers, using its pointer for the new requestor, though this can be costly in cases where a block has a large number of readers, such as a lock. Another method is to have a separate pool of free pointers available to all the blocks. This method is usually effective as the number of blocks shared by a large number of processors is not normally very large.[2]

References

1. Reihnhart, Steven; Basu, Arkaprava; Beckmann, Bradford; Hill, Mark (2013-07-11). "CMP Directory Coherence: One Granularity Does Not Fit All" (http://research.cs.wisc.edu/multifacet/papers/tr1798_region_coherence.pdf) (PDF).
2. Solihin, Yan (2015-10-09). *Fundamentals of Parallel Multicore Architecture*. Raleigh, North Carolina: Solihin Publishing and Consulting, LLC. pp. 331–335. ISBN 978-1-4822-1118-4.
3. Laudon, James; Lenoski, Daniel (1997-06-01). *The SGI Origin: a ccNUMA highly scalable serve* (http://dl.acm.org/citation.cfm?id=264206). Proceedings of the 24th annual international symposium on Computer architecture.
4. Seo, Dae-Wha; Cho, Jung Wan (1993-01-01). "Directory-based cache coherence scheme using number-balanced binary tree". *Microprocessing and Microprogramming*. **37** (1): 37–40. doi:10.1016/0165-6074(93)90011-9 (https://doi.org/10.1016%2F0165-6074%2893%2990011-9).
5. Chaiken, D.; Fields, C.; Kurihara, K.; Agarwal, A. (1990-06-01). "Directory-based cache coherence in large-scale multiprocessors". *Computer*. **23** (6): 49–58. CiteSeerX 10.1.1.461.8404 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.461.8404). doi:10.1109/2.55500 (https://doi.org/10.1109%2F2.55500). ISSN 0018-9162 (https://www.worldcat.org/issn/0018-9162). S2CID 683311 (https://api.semanticscholar.org/CorpusID:683311).

Retrieved from "https://en.wikipedia.org/w/index.php?title=Directory-based_cache_coherence&oldid=1105493837"

This page was last edited on 20 August 2022, at 14:33 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License 3.0; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.