International Journal of Advanced Trends in Engineering and Technology (IJATET)
Impact Factor: 5.665, ISSN (Online): 2456 - 4664
(www.dvpublication.com) Volume I, Issue I, 2016

DIRECTORY BASED CACHE COHERENCY, ORGANIZATION, OPERATIONS AND CHALLENGES IN IMPLEMENTATION - STUDY

Subrahmanya Bhat* & Dr. K. R Kamath**
* Department Computer Application, Srinivas Institute of Management Studies, Mangalore, Karnataka
** Professor, Department of Computer Science, Srinivas Institute of Technology, Mangalore, Karnataka

Cite This Article: Subrahmanya Bhat & Dr. K. R Kamath, "Directory Based Cache Coherency, Organization, Operations and Challenges in Implementation - Study", International Journal of Advanced Trends in Engineering and Technology, Page Number 30-33, Volume 1, Issue 1, 2016

Abstract:

Todays systems are designed with Multi Core Architecture. The idea behind this is to achieve high system throuput. Once the Processor clock speed reached its saturation, designers opted for having multiple cores. Each Core or Processor equipped with their own private cache memory. But under Chip Multiprocessor, where all the processor have access to shared memory, having respective cache memory will result with Cache Coherency Problem. In Directory Protocol, for each block of data there is a directory entry that contains a number of pointers. The purpose of this number is to mention the locations of block copies. The important advantage of directory based protocols is that they scale much better than snoopy protocols. In addition to this it has the advantage of ability to exploit arbitrary point-to-point interconnects. But mean time it also has the overhead in terms of the storage and manipulation of directory state. This paper discus different Directory Based implementation, operations along with and its implementation difficulties.

1. Introduction:

Todays systems are designed with Multi Core Architecture. The idea behind this is to achieve high system throughput. Once the Processor clock speed reached its saturation, designers opted for having multiple cores. Each Core or Processor equipped with their own private cache memory. A typical shared memory multiprocessor contains multiple levels of caches in the memory hierarchy. Each processor may read data and store it in its cache. This results in copies of the same data being present in different caches at the same time. In order to maintain the consistency, Cache Coherence Protocols have been imposed on such systems. Cache coherence protocols are classified based on the technique by which they implement as Snooping and Directory based protocols. In Directory based protocols, a main directory is maintained containing information on shared data across processor caches. The directory works as a look-up table for each processor to identify coherence and consistency of data which is currently being updated. A directory-based protocol is a smart way of implementing cache consistency on an arbitrary interconnection network.

2. Directory Organization:

Choosing the structure of the directory should be one of the criteria in order to get good system performance while implementing the cache coherency. Directory organization defines the storage structures that make up the directory and the different information stored in directory. While implementing the directory scheme one needs to concentrate on minimising the storage requirement for such directories. There are two major approaches in Directory Structure, like Limited Pointers Allocation Dynamic Pointer Allocation Directories and Sparse Directory Schemes [1]. These organizations provide good performance even though incurring some memory overheads in terms of pointer locations. At first we will analyse the characteristics of these two strategies, and compare them in terms of memory requirement and operations needed while implementing cache coherence.

In a traditional directory structure called Censier/Feautrier organization, the directory maintains a vector of valid bits, one bit per processor, for each data block at main memory. With this structure, the resulting memory overhead is exponentially increases with number of processors and hence does not support for the systems with very large number of CPUs. Furthermore, even if the memory overhead was reasonable in a large-scale machine, the large size of the valid bit vector will result with difficulties in implementation. So when such traditional directory organizations incur excessive memory overhead, other alternative methods needs to be explored. A general approach in this regard for reducing memory overhead is by either reducing the directory's length or the directory's width.

3. Implementation of Limited Pointers Directories:

In small-scale multiprocessors usually a few caches contain a given block of data whenever a write occurs. In such case, on a write operation, the data is invalidated in all except the one cache where write has been done. The valid bit vector in traditional directories has only a few of its bits set. It is therefore more efficient to replace the vector with several pointers, which are log n bit fields that encode the unique identities of those caches containing the data block. This overhead is calculated by dividing the number of bits in a directory entry by the number of data bits represented by that entry. Here the memory overhead is directly proportional to

the number of processors and the number of pointers in a directory entry, and inversely proportional to the number of bytes in a cache block. With this approach it has been found that it is possible to have the directory entry with at least three pointers for most large-scale systems, and keeping the memory overhead much lesser than traditional directory schemes [7]. With only a small number of pointers in each entry, it may happen that the directory entry run short for a given memory block. This occurs when a request due to a read miss arrives at memory and the directory finds no free pointers remaining in that entry to record this state. To handle this there are two basic strategies like broadcast scheme and no-broadcast scheme. In a broadcast scheme, a broadcast bit in the directory entry is set, indicating that the directory is no longer keeping track of the caches containing that data. When a processor eventually writes the data, the directory must resort to broadcasting an invalidation request to all caches. No-broadcast scheme insists that not more than ***k*** cached copies of a block of data may exist at any given time. Here ***k*** is the number of pointers in each directory entry. When a free pointer is needed and if it is not available, a pointer is randomly selected and freed by invalidating the data from that cache the pointer identifies. It has been found that limited pointers directories incur reasonable storage overhead, and its implementation is straightforward [7]. How-ever, in these systems performance may not be optimal especially in large-scale machines.

4. Dynamic Pointer Allocation Directories:

Another directory organization proposed is the dynamic pointer allocation method. This method takes advantage of the fact that most data blocks are shared at any given time by only a few caches, while a few blocks may be widely shared. Similar to the limited pointers directories, this directory scheme uses pointers that contain the unique identities of those caches containing the data. However, the number of pointers available in an entry is not fixed, and it is rather allocated on-the-fly from a pool of available pointers. When a pointer is no longer needed, it is returned to the pool.

5. Implementation of Dynamic Pointer Allocation Directory:

Here the directory is containing a number of pointers, each paired with a link to form basic data structure - linked list of cache pointers. In addition, each block of main memory has an associated dirty bit, a link to a list of pointers allocated to the block, and an empty bit that indicates whether or not the block's list of pointers is empty. When the list is not empty, the last pointer links back to the main memory block, forming a circular list. At system start-up, a single linked list consisting of all the pointers is built. A special register known as the free list link is set to contain the address of the first pointer/link pair; this register is a link to the head of the free fist. On a cache miss, a pointer is removed from the free list and added to the head of the list for the desired memory block. When permission to write a block is requested by a cache, the directory steps through the block's pointer list, sending invalidations to each cache on the list and returning the pointers to the head of the free list. The end of the list is reached when the end-of-list bit associated with each pointer/link pair is found to be set. The dynamic pointer allocation directory adds less storage to each data block than a standard limited pointers directory with several pointers per entry. With this method of directory, memory overhead to be considered is in terms of the dirty bits, empty bits, and head links. Since caches are growing larger with each new generation of systems, the length of the pointer/link store must also grow accordingly. Therefore, the directory overhead will rise over time, due to increases in the width of the pointer list head link field. It has been found that with this scheme, even a increase in the size of the pointer/link store by a factor of 10 or higher results with only a modest overhead increase for a given cache line size[7]. The dynamic pointer allocation scheme therefore has resulted with good storage efficiency even in the presence of large caches. The pointer/link store would scale gracefully over the time. The number of pointers required on a node depends on the cache size. The number of pointers that can be provided will therefore scale at the same rate as the size of cache memories.

As in a limited pointers directory, here also it is possible to run short of free pointers. This occurs if a pointer is to be allocated from the free list but the free list is found to be empty. The action to take on such instance is to use some means to select a pointer and then free it by sending invalidation to the cache identified by that pointer. The selection of the pointer can be on random basis using some hardware register. Since the address of the block is needed to send an invalidation message, the list beginning at the pointer indicated by the register must be traversed until the last pointer on the list is found. This address and the pointer can be used to send invalidation, thereby freeing the pointer/link pair. An interesting effect occurs if caches are allowed to replace clean blocks without notifying the directory. Stale pointers indicating caches that no longer contain the data may accumulate in the pointer/link store. If they are not returned to the free list, these stale pointers could potentially occupy most of the pointers that would otherwise be free, perhaps causing the directory to run short of free pointers frequently. This is undesirable since processing a read miss which is the most frequent directory operation, will be considerably slower at the directory if no free pointers remain. So while not required for correctness, enhancing the protocol by having caches send replacement notifications to the appropriate directory when a clean block is replaced shell improve the system.

It has been seen that dynamic pointer allocation is more robust than the basic no-broadcast limited pointers directory, in that it can handle blocks that are shared by many processors without performance

degradation. The data blocks fall roughly into three categories. First, there are blocks for which the available pointers are rarely exhausted. All of the limited pointers schemes perform well for these blocks. Second, there are blocks with only moderately high read/write ratios that use all of their pointers with some frequency. Finally, there are blocks with very high read/write ratios. Standard no-broadcast schemes have only a minor negative impact for blocks with moderately high read/write ratios, but do not exploit the opportunity to significantly reduce miss rates for blocks with high read/write ratios [7]. On the other hand, the coarse vector schemes will perform well for blocks with high read/write ratios since writes are infrequent and miss rates are low. However, their performance suffers for blocks with moderately high read/write ratios. For such blocks, the coarse vector strategy will cause substantial traffic due to extra invalidations unless the number of caches represented by each bit can be kept small.

6. Implementation of Sparse Directory:

The sparse directories scheme is based on the following observation that at any point of time only a few memories can be resident in cache. Here usually cache size will be lesser than the memory size. Given that coherence protocol only needs sharing information for lines in the cache, most directory entries are emty or not used in most of the time. For example, 1MB cache with 1GB main memory leads to 99.9% waste of directory space. The basic idea of the method is to minimum the overhead of the empty entries in the directory. A typical way to achieve this is to leverage the link list tructure. For details, we have a linked list for each memory block. All cache lines of the specified block, probably in different nodes, are organized as linked list through an additional previous/next pointer in the cache line. To find the head of each list, we can have a single head pointer per line in memory. A more practical alternative would be to simply maintain a small lookup table storing the head of the list for cached lines. The sparse directory scheme can reduce the space overhead dramatically. We have some additional pointers in each node's cache, of which the size is proportional to the cache size. And we have a small lookup table in memory. However data write applications, we have to iterate through the list, which make the latency proportional to the number of sharers.

7. Reducing Operations in Terms of Messages:

Another option where in the performance of Directory based cache coherence system that could be enhanced for optimization is in terms of minimising the total number of message exchanges that take place while implementation the coherency. Here mainly we take in to considerations the request for a cache line from requesting node to its home node, and the way how home node is going to serve this request. Mainly there are 2 approaches for serving this kind of request, Intervention Forwarding and Request Forwarding. While considering the case of a read miss on dirty block, in order to serve the request, first the requesting node has to send the request to Home node, and the Home node will forward the Owner node id to the requesting node. Later requesting node sends data request message to Owner node, and Owner reply's data to requesting node. Finally Data and directory update message has to be sent to Home node from the Owner node. Hence totally there will be 5 message needs to be exchanged in order to serve the request.

Message Reduction by Intervention Forwarding:

While considering the case of a read miss on dirty block, in Intervention Forwarding, first the request is sent to Home node, and the Home node will forward the Intervention Read message to the Owner node, the Owner node is going to reply with Data to Home node, and finally Home node will response to Request node with Data. Hence totally 4 message sequence will be required to serve this particular request with Intervention Forwarding method.

Message Reduction by Request Forwarding:

While considering the case of a read miss on dirty block, in Request Forwarding, the request node will first send the request to Home node, and the Home node will send the request to send data to the requesting node. The Owner node is going to reply to Home node as well as the requesting node with Data in single message. Hence totally 3 message sequence will be required to serve this particular request with Request Forwarding method.

8. Conclusion:

The directory-based cache coherence protocol is a scalable approach compared with snooping-based protocol. It avoids broadcasts by storing information about the status of the cache line in a directory and use point-to-point message communication. However, the naive implementation of directory-based cache coherence has much storage overhead, which limits its performance. With Limited Pointer scheme, they replaced the vector with several pointers, which are log n bit fields that encode the unique identities of those caches containing the data block. With this approach it is possible to have the directory entry with at least three pointers for most large-scale systems, and keeping the memory overhead much lesser than traditional directory schemes. Similar to the limited pointers directories, Dynamic Pointer Allocation Directory scheme uses pointers that contain the unique identities of those caches containing the data. The dynamic pointer allocation directory adds less storage to each data block than a standard limited pointers directory with several pointers per entry. This scheme has resulted with good storage efficiency even in the presence of large caches. Sparse Directory Scheme will simply maintain a small lookup table storing the head of the list for cached lines and can reduce the

space overhead dramatically. Here we have some additional pointers in each node's cache, of which the size is proportional to the cache size. However when writing data it needs to iterate through the list and may result with some latency factor. Techniques like Intervention forwarding and Request Forwarding will reduce the total number of message exchanges required while implementing the Cache Coherency and hence limit the network bandwidth.

9. References:

1. Design and Implementation of a Directory based Cache Coherence Protocol by Dimitris Tsaliagos Technical Report FORTH-ICS/TR-418 May 2011
2. Effects of cache coherency in Multiprocessors, By Michel Dubois, Member- IEEE, and Faye A. Briggs, Member-IEEE
3. Prof. M. Shaaban's EECC 756 Lecture notes on Cache Coherence Problem in Shared Memory Multiprocessor.
4. Parallel Computer Architecture (PCA) BY David E. Culler and Jaswinder P. Singh (1999 edition).
5. http://parasol.tamu.edu/~rwerger/Courses/654/cachecoherence1.pdf
6. CS252 Graduate Computer Architecture. A course by David A. Patterson in CS Department of UC Berkeley.
7. Cache Coherence Directories for Scalable Multiprocessors -Rrichard Simoni-Stanford, California - Technical report
8. Subrahmanya Bhat & K. R. Kamath, Directory Organizations in Cache Coherency Systems for High Performance Computation, International Journal of Current Research and Modern Education (IJCRME), Volume I, Issue I, p.p 868-871, (August,2016), ISSN (Online): 2455 – 5428
9. Subrahmanya Bhat & Dr. K. R. Kamath, Directory Based Cache Coherency Protocol In Multi-Core System For High Performance Computation, International Journal of Current Research and Modern Education (IJCRME) Volume I, Issue I, pp. 257-261, (May 2016), ISSN : 2455 – 5428
10. Subrahmanya Bhat B and Dr. K.R Kamath, Cache Hierarchy in Modern Processors and Its Impact on Computing, International Journal of Management, IT and Engineering (IJMIE), Volume 5, Issue 7, pp. 248-253, (July 2015), ISSN: 2249-0558, I.F. 5. 299
11. Subrahmanya Bhat & K. R. Kamath, Optimization Approaches in Directory based Cache Coherency Systems for High Performance Computation, International Journal of Current Research and Modern Education (IJCRME), ----Volume I, Issue I, p.p 868-871, (August,2016), ISSN (Online): 2455 – 5428