International Journal of Current Research and Modern Education (IJCRME)
ISSN (Online): 2455 - 5428
(www.rdmodernresearch.com) Volume I, Issue I, 2016

DIRECTORY ORGANIZATIONS IN CACHE COHERENCY SYSTEMS FOR HIGH PERFORMANCE COMPUTATION

Subrahmanya Bhat* & K. R. Kamath**

* Department of MCA, Srinivas Institute of Management Studies, Mangalore, Karnataka
**Department of CS, Srinivas Institute of Technology, Mangalore, Karnataka

Abstract

Caching the required Data or Instruction is one of the common practices used for improving the system performance. But in Multiprocessor Systems, Caching of shared data requires a cache coherency mechanism to maintain the shared memory model. In small-scale multiprocessors where all of the processing nodes sit on a common bus, the coherency problem could be solved with a Snoopy Protocols, but they do not scale up for large number of Processors. To support large Multiprocessors Directory Based Protocols are the best option. In these schemes a separate directory memory is maintained that identifies the caches containing each cache-line-sized block of memory. When data is written, the directory information is used to direct invalidation or update messages, depending on the type of protocol, to only those caches that must receive them. A primary step in designing a directory-based coherence mechanism is selecting a directory organization. This paper discus the directory organization, the storage structures that make up the directory and the nature of the information stored within. We discus on Limited Pointers and Dynamic Pointer allocation directories. These organizations potentially provide good performance while incurring modest memory overhead, even in large-scale machines. We discuss the characteristics of both strategies, and compare them with different directory-based approaches.

Index Terms: Multiprocessor Systems, Snoopy Protocols, Limited Pointer & Dynamic Pointer

1. Introduction:

Today's 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 Prtocols 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 for getting good system performance in terms of achieving memory consistency. Directory organization defines the storage structures that make up the directory and the nature of the information stored in it. There are two major approaches in Directory Structure, like Limited Pointers Allocation and Dynamic Pointer Allocation Directories [1]. These organizations provide good performance even though incurring some memory

overheads in terms of pointer locations. This paper analyse the characteristics of these two strategies, and compare them.

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 exponentailly 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. 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. 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 inversly proportional to the number of bytes in a cache 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 [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 straight forward [7]. However, in these systems performance may not be optimal especially in large-scale machines.

4. Dynamic Pointer Allocation Directories:

The 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. Directory Organization:

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 an 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 an 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.

6. Comparisons between Directory Schemes:

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.

7. Conclusion:

Since the directory has limited resources, there will always be a sequence of references that causes the directory to perform poorly. All of the limited pointers schemes incur some performance penalty on any block for which the pointers are exhausted. By sharing all of the pointers available to a memory module across all of the blocks in that module, dynamic pointer allocation drastically reduces the probability of running short of pointers. This feature ultimately makes dynamic pointer allocation more robust to different types of reference behaviour, allowing it to perform equally well for each of the three categories of data blocks.

8. 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. Wikipedia.com