ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

Cache Coherence Mechanisms

Sultan Almakdi, Abdulwahab Alazeb, Mohammed Alshehri
College of Computer Science and Information System, Najran University, Najran, Saudi Arabia

Abstract*— Many modern computing architectures that utilize dedicated caches rely on coherency mechanisms to maintain consistency across dedicated caches [2]. These mechanisms, which are to be the focus of this paper, rely on underlying hardware synchronicity to resolve the issue of the value of a particular piece of data at a given instant of time based on the discrete way in which processor instructions are executed with each clock cycle with corresponding memory accesses following[2], [4]. It should be clear that inconsistencies occur when data is written and noticed when data is read. It is the goal of this paper to explore the idiosyncrasies of the coherence mechanisms involved with dedicated caches via researching two common types of mechanisms, snoop-based and directory-based, and simulating their operations on top of a simulated architecture consisting of multiple processing cores and a layered cache system consisting of dedicated caches. In this paper, we implemented snoopy and directory protocols, and measure hit rate, compulsory miss rate, capacity miss rate, and coherence forces for each one. In addition, we show that how each scheme affected by block size and cache size.

***Keywords*—Cache Coherence, Coherency forces, Directory Protocol, Snoopy Protocol, Multi-cores.**

I. INTRODUCTION

According to Smith the properties of both the spatial and temporal locality comes from a sequence of memory, which is generated by a program. The temporal locality which is also known as locality in time means that there is a high likelihood of a program which was referred to be referred again [10]. Spatial locality on the other hand means that any address referenced by the program in a short span of time is more likely extend in a small portion of the address space. For instance any given program which operates on large data structures and have the elements of the structure located in the sequential memory location will have the memory generated clustered into a small range of address space. The private data caches are located near the processors and they are small and fast thus they are used to exploit the memory referencing properties and significantly reduce the time required accessing the main memory which is larger and more complex. Having the private cache copy reduces the time that is required to copy memory location thus making the referencing much faster in copying the memory in both the temporal locality and the spatial locality [15], [18].

Private data have been proven to be very effective especially in the shared memory multiprocessor where they reduce the overall delay in accessing the shared memory. The private data cache will improve performance more in uniprocessors as compared to multiprocessor because in a multiprocessor the data is spread across the processors thus reducing the locality of references [7], [3]. In a multiprocessor system, having different caches can lead to coherent problem, which means that the different caches will have different values simultaneously. The cache coherence mechanism is used to ensure that the processor only read the memory location with the correct value.

The two dominant classes of cache coherence protocols for hardware shared memory are snooping and directory protocols. In Bandwidth Adaptive Snooping the processor broadcasts a request for a block to all the nodes in the system looking for a specific owner. In directory protocol, processor unicast a request to the home directory for a block, next the directory forwards this request to the owner which can be trivial if the directory itself is the owner and the owner respond to the request. The key difference between snooping and directories is that snooping will work best when there is a lot of bandwidth available but when the bandwidth is limited directories is preferred [6]. In this paper, we provided a deeply study of cache coherence concepts and cache coherence protocols, which are Snoopy and Directory protocols under Background section. Then we explained in details the implementation and system design. Results and evaluation which show the experiments that we have done are given in evaluation and results section.

II. BACKGROUND

In cache coherence problem there are two important aspects, which we are going to discuss; we have the model of the memory system, which is presented to the programmer, and we also have the mechanisms, which are used by the system to maintain coherence between the main memory and the caches.

A. Consistency Models

The consistency model of a multiprocessor gives a clear definition of the programmer’s view of time-ordering events in the different parts of the processor. There are different definitions to the system with coherent cache: it can be defined as system whose value is returned to the Load instruction and the value is given from the same address by the store instruction [11]. Each processor and memory module will have different ordering of events that is caused by the buffering and delays in the processor-memory interconnection network. The events included in the processor-memory are the read-write and synchronization operations [15].

The system programmers usually have no assurances on the order of events, which mean there is a high likelihood of overlap in operations in the different processor sections. This

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

task falls to the programmer (compiler) who has to ensure that the dependency in the operation is not disrupted. The sequential consistency model is used to define the exact ordering of the sequence of execution of operation inside processor and between processors. In a given multiprocessor system when the results executed by the program are the same as if operations of all the processors inside were executed in the same sequential then the multi-processor system is then said to be sequentially consistent [12].

The consistency model means that each access in the shared memory must be complete before the next shared memory access can start. The program determines the order of execution in all the memory operations whereby this strict ordering of memory process can affect the performance. This is because there is high limitation in the accepted overlap between memory operations given by one processor and other processors in the system. The sequential consistency model allows for greater overlap in memory reads-writes while the weak-consistency model relaxes the strict ordering of events. In a weak-consistency model it’s only the memory that accesses the programmer defined synchronization variables thus it guarantees the sequential consistent order [18].

Memory references from the different processor to shared data variables can access the synchronization variables. It means that a developer is not able to make any assumptions on the ordering event between synchronization points while using the weak-ordering model [7], [8]. In efforts to prevent the nondeterministic operations each processor must then guarantee that the outstanding shared memory accesses are first completed before a synchronization operation can be issued. The release consistency also weakens the ordering constraint on synchronization variables by dividing its operation into many releases and acquires operations [10]. The acquire operation is issued by the processor in order to obtain exclusive access to shared memory object. Before the processor can grant the exclusive access it has to wait for the acquire operation to complete; this is done to reduce interference in the shared memory. The key function of the release memory is to allow for exclusive access to a shared memory objects; this is done to guarantee that any modifications made by the core to the shared objects are performed exclusively in the shared memory before any king of access is granted. The waiting for completion is known as the splitting of synchronization operations and it allows the consistency model achieve greater overlap of the memory operations that are issued by all the processors compared to both the weak or sequential consistent models [11]. Several studies have been carried out to measure the effect of the extra overlap where they have concentrated on the performance improvement that is achieved by using the relaxed consistency model [13].

B. Cache Coherence

In this paper, we focused on the mechanisms that are used by the system to ensure that processors cannot directly access the invalid data because this is a related problem to the memory consistent model whereby in a shared memory multi-processor individual processors can access any location in the common address space by using a single read or write instruction [4]. Each processor has its own private data cache which means that a similar copy of the shared memory location may be found in more than one cache at the same time. To ensure there is no stale version the value of the shared memory location must be broadcasted to all other processors with the cached copy of the location. For instance: we have a system with 2 processors and each has its own private data cache [1], [3], [5]. First of all, the value of block A in the main memory is 1 in the time 0, and no cache has this block cached yet. At time 1, P1 read block A, and gets it in to his cache form the main memory since this block has not been cache yet by any cache in the system. At time 2, P2 reads A also and gets it into his cache. At time 3, P1 write 0 to its copy of block A (Update the value of block A). At time 4, P2 reads again block, but it reads an invalid data since block A was updated by P1. Here the problem of cache coherence is happened. Therefore, we can define the cache coherence is the process of ensuring that any new value is broadcasted to all processors enabling them update to the new value changed by the new processor [5].

Time	Event	Cache-1	Cache-2	Memory
0		-	-	1
1	Processor1 reads A	1	-	1
2	Processor2 reads A	1	1	1
3	Processor1 writes 0 in A	0	1	0
4	Processor2 reads A	0	Read wrong Value (1)	0

Figure 1 Example of Cache coherence problem

C. Relationship between Consistency models and Coherence

The relationship between the two can be best understood by looking at the relationship that exists between the cache coherence and memory consistency model in which cache coherence is to ensure that all the caches follow a logical order when they write in a specific block thus reducing the likelihood of reading the wrong value [18]. The consistency model describes the order of writes to the different blocks depending on how each block perceives them. Once this is done it means that the coherence mechanism will force the value returned by any given load to be updated as guaranteed the by consistency model as long as the programmer in charge can follow the consistency model rules. The coherence mechanism will ensure that the effects of each write operation in a shared memory will be broadcasted to all caches before the next write by the processor to the same location [18].

According to Dubois the accesses are strongly ordered if used in a system with weakly-ordered consistency model and only the access to pre-defined synchronization variable [14].

He continues to say that the ordering of the access to shared data memory locations can occur in any order if ordered by different processors. Another key feature of the coherence mechanism is no processor can proceed with the synchronization process unless all the memory access has been identified by the coherence mechanism. The best way to ensure that the caches are only coherent at the synchronization point is by using the weak-ordering consistency model [15].

D. Four Primary Design Issues

The key consideration while choosing cache coherence mechanism for a multiprocessor system is its effectiveness and performance. You have to look at how the mechanism reduces the average delay while fetching data from the memory. One also has to consider the cost of implementation that can be measured by the size of the memory required and the complexity of the control logic. There are different coherent mechanisms that have different performance output and cost, which makes it hard to choose which one to select. One also needs to take into consideration some of the issues that affect the cache coherence mechanism, and in our implementation, we took care of all the following design issues [2], [18]:

1. Coherence Detection Strategy

This is one of the surest ways to determine the performance of the cache coherence mechanism. One has to look at the mechanism as it performs the statically at compile time or the dynamically at runtime. This strategy used by the coherence mechanism will solve the problems by first examining the memory addresses generated by a program at run-time while at the same timekeeping track the specific processors which must have the copy from the memory addresses. The static coherent scheme will attempt to predict the memory addresses that have the highest likelihood of becoming stale: analyzing the program referencing behavior does this. There is a clear difference between the implementation of the coherence mechanism and how it determines the location where memory is more likely to go stale. To be able to detect the coherence mechanism, software will be needed since it relies on a compiler and also hardware support in order to preserve the information about the location of the memory. Another key difference is that the mechanism that detect if there is need for coherence action uses the system memory addresses but it can also be combined with other software and compiler to create a hybrid scheme. What this means is that we get two major classes in the coherent mechanism where we have the statically detected and dynamically detected [18].

2. Coherence Enforcements Strategies

The strategy used by the coherent mechanism can adversely affect the performance of the multiprocessor memory system. The coherent mechanism must apply an effective coherent scheme, which will make sure that no processor can access stale memory location. Lilja gives a way to do that is by ensuring that all shared writable memory locations cannot be cached thus guaranteeing that there will be no multiple copies [15]. This approach is not an effective one because once you do not cache then there will be no referencing by the program and this can adversely affect the performance. There are two effective strategies, which can be used, and this will allow for memory location to be cached. The only difference between them is that they will update or invalidate stale cache before they can be referenced again. These two new strategies have advantages and disadvantages, for the update strategy the new value will be updated and broadcasted to all processors this preventing use if any processor which has a cached copy that has not been updated. The downside to this approach is that to update all the processors there will be need for additional system traffic. The invalidation strategy in which can be either direct invalidation or self-invalidation. It works by the use of a complier, which generates coded instructions to the processor forcing it to invalidate all or some of the data cache. This is usually done before the processor can attempt to access the stale data. There is also invalidation of all cached copies; this means all copies in the memory block are made invalid thus forcing the processor to miss them the next time it tries to reference the block. One advantage of using this approach is that it significantly reduces the system traffic when you compare it to the update strategy but on the downside it can have delays if the memory block is re-used and increase the miss rate in the system [18].

3. Cache Block Size

The cache block size is also referred to as the line size; this is the exact number of memory words invalidated or updated as a single unit. There is also the fetch size, which is the number of words that are moved from the main memory to the cache on every miss [16]. One of the surest ways is to reduce the number of miss while increase the number of words in a cache block. This means that the memory locations, which are in close proximity also known as spatial locality to the recently referenced locations, will be referenced in the near future [18]. According to Przybylski and Smith there is a downside to the fetched data because the block size can become too large which means that the miss ratio will increase and the probability of re-using the data replaced becomes very small. They continue to add that the block size that is used to minimize the memory delay is smaller than the block that actually minimizes the miss ratio. This is because there is additional time that is required to transfer large memory block that can significantly overwhelm the power to receive the first word [16]. Cache block has another advantage because blocks, which are larger than a single word, can significantly reduce the memory overhead of the directory coherence mechanism. For the compiler-directed coherence mechanism like the version control you will need independent block size and dirty bit per word. The downside to this is any cache block that is larger than a single word can cause a false sharing where two words, which cannot be, shared end up in the same block [18]. A good example will be when the loop

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

scans an array; the array subscript increases from one iteration to the next. This means if there is a single stride then all the consecutive elements of the array will be accessed by the single iteration of the loop. This is different when the iterations are distributed to different processors then the cache block size becomes bigger than one array element meaning if arranged linearly a number of processor will have to have a copy of the same memory block. This copying lead to a large amount of sharing which is commonly referred to as false sharing since the processor sharing the memory block and not the actual data. The false sharing is not harmful as long as the processor read only the array but when the processor tries to write an element using an invalidation protocol the all the copies of the written block will be instantly invalidated. The key downside here is that every attempt to writer on a shared memory block will lead to an invalidation thus every read will be a miss making the blocks criss-cross between caches. This scenario will lead to the significant increase to the processor miss ratios and the increase in memory traffic [15], [16].

According to Argarwal, there is a way to eliminate false sharing, they say that it's better to use small blocks in the dynamic coherent schemes in case you lose the potential benefit in exploiting the spatial locality. When you have blocks, which are larger than one word, then the compiler needs to know the size of the block immediately control how the data is placed in the memory. False sharing is known to cause dependencies between two initially independent programs and this happens once the compiler completely ignores the block size. Having dependent program can lead to poor program execution. The only solution to this problem is to use one word blocks or to restrict the data placement thus making sure that each block will have one unique name that is variable. This restriction has little or no effect when it comes to array because the last block is allocated to the array. If there are large block then a huge amount of space is wasted on the scalar variables [17].

4. Precision of Block Sharing Information

The coherence scheme automatically determines which memory references need coherence action to gain access to the memory addresses and this depends on the program that generates them. The coherence mechanism monitors each memory block for its sharing characteristics because the hardware does not have the ability to determine how the memory block will be shared [2]. According to Censier one need an enormous amount of memory bits to store all the information which means there is need to be a mechanism of storing this information which states how the blocks will be shared and also to know which copy each processor has the mechanism has another function of sending invalidation messages to the processor of those cached copy that needs to be invalidated in the memory blocks [11]. Argarwal states that there is need to be a precise mechanism to achieve this plus it also reduces the size of the information stores [17]. He adds that the mechanism must also broadcast the invalidation messages to all the processors to make sure they are updates. The broadcast have been found to significantly increase traffic of the memory especially in the interconnection network. It also reduces the directory memory requirement because it will only maintain the sharing information that is the block cached or not [17].

E. Cache Coherence Protocols

A cache coherence protocol is the protocol that maintains the consistency between caches in a system where they are in distributed shared memory or centralized shared memory. The coherency that is maintained by coherence protocol will be according to specific consistency model. The older multiprocessors systems were maintaining the sequential consistency model. The modern shared memory systems on the other hand support the release consistency model. The transition of the two states may vary greatly where you find that the implementation may go with a different invalidation or update transition. You may find that you have update on write, update on read or even invalidate on read. The amount of inter-cache traffic can affect by the choice of transition. This may also affect the amount of cache bandwidth that is available for the actual work. When designing the distributed software this should be taken into consideration because it can lead to a strong contention between the multiple processors and the caches. There are two main protocols: Snoopy-Based Protocol and Directory-Based Protocol [2], [4], [18].

1. Snoopy-Based Protocol

The Snoopy cache protocol is used in bus-based multiprocessor system this is because they must use a broadcast medium bus. The bus-based multi-processor allows for all connected processors to observe every bus transactions that mean that each cache will monitor every memory transactions [2]. The function of the cache is to snoop and if it notices a transaction, which can affect the consistency of the block of data it, notify the cache coherence protocol, which intervenes. Snoopy cache coherence protocols behave in invalidated or updated manner, which was mentioned earlier so when the cache notices a write transaction on an address that is already in its cache it instantly invalidate the cached copy. The cache can also observe the read transaction where the state has to indicate whether it has the latest copy, if this is the case then the cache will instantly intervenes before the memory can reply [5].

The snoopy cache coherence protocols are used in a broadcast-enabled interconnects architecture, but there was a need for another different type of protocol for distributed shared memory. The Directory based cache coherence protocol was introduced for the DSM/NUMA architecture [6]. The most common structure of snoopy- based protocol is shown in the fig. below [5].

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

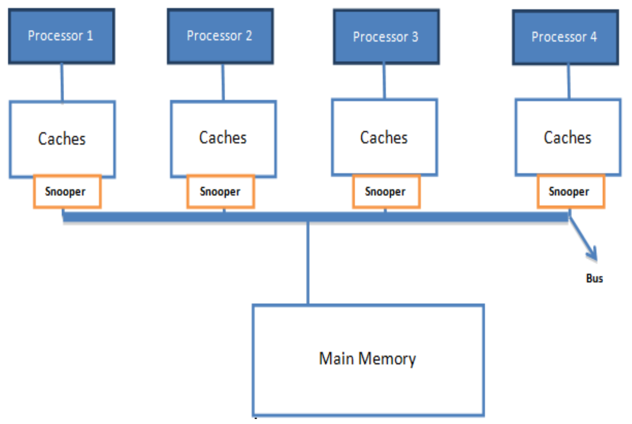

Fig. 2 The structure of Snoopy- based protocol

2. Directory-Based Protocol

This is a cache coherence protocol system that does not use the broadcasting strategy that is why it has to store all the location of cached copies of every block in the shared data whether is centralized or distributed in different processors. Every memory block is assigned to a directory entry that is the name of the structure that stores all the information about the different locations of the shared block. The directory-based cache will take any form of action depending on the current state of the directory and transactions. The directory-based protocol also needs to be distributed between the nodes in the given network. The cache coherent non-uniform memory access contains nodes and each of these nodes that are in the interconnection network has blocks of local memory that is always associated with local cache and directory. The advantage of distributing the directory-based protocol is that it reduces bandwidth issues and potential bottleneck [3], [18]. The most common structure of directory- based protocol is shown in the fig. Below.

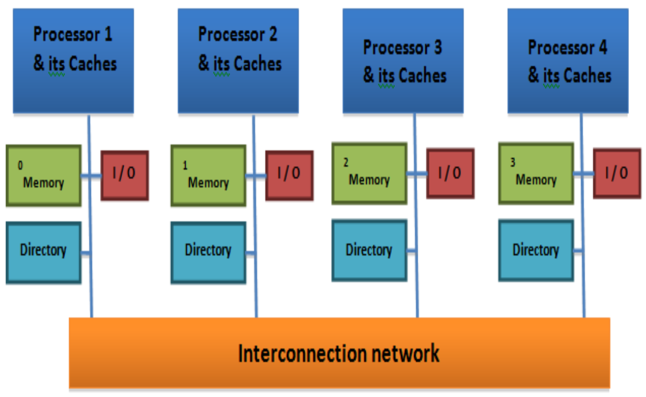

Fig. 3 The structure of directory-based protocol

In this structure, each processor has dedicated main-memory, and each memory has directory associated with it. Each processor has fast access to its local memory and slower access to remote memory that is resided at other processors. The nodes are connected with a scalable interconnect, causing in routing of the messages from sender to receiver instead of broadcasting. Consequently, this protocol cannot snoop anymore, thus records of sharing state is now kept in the directory in order to track them [4].

Directory based cache coherent protocol has two major components that are directory organization, and the set of messages types and message actions. First we have the directory organization that states to the structure that is used to store all the directory information. There is need for extra memory, which is known as the directory memory overhead, and it's used to store the directory information. To memory overhead is calculated as the fraction of the additional memory to the total memory in the directory. The key function of the directory based cache coherence protocol is to send messages through interconnect to other nodes that have the coherence actions. Each of the message sent has specific message type which include instructions and depending on the message receive each node can take coherence action independently [9].

In this protocol there are three processors involved which are the local node where a request creates, home node which has the memory location of an address, and remote node which holds a copy of the cache block, in either exclusive or shared [1]. Each block in any cache has one of the following states [1]

1. **Shared**: which means this block appears at least in cache of one processor and the memory is up-to-date also any processor is able to read the block.
2. **Exclusive**: that means only one processor which is the owner has the data cached and memory here will be staled. Therefore, only this processor will have complete access and right to write to this block.
3. **Invalid (Uncached)**: in this state no processor has the data cached.

Bit-vector is used by Directory-Based Protocol, and it works by associating one bit per processor in the directory for each of the shared block of local memory data [18]. Each bit in every vector will have information on if the shared block is present in similar nodes cache. The dirty bit is also associated with every block of data in the directory and the dirty bit will make sure that only one processor has a copy of block of data. What this means is that the processor will have ownership of the block where it can be able to write transactions to it. There is also the cache entity protocol whose function is to maintain two bits per block that is a part of its cache line. One of the bits will indicate if the block is valid and the other will show if the block can be written. The cache coherence protocol is meant to maintain the coherence of the indicator bits in all the directories and cache of the node in a given network. One thing to note is that the dirty bit is not set because no node has the write permission of the block [18].

F. Write Update vs. Write Invalidate

When we want to design any of the cache coherence schemes, we have to choose between the write update or write invalidate strategies. Most developers prefer to use the write invalidate because it's more popular and it consumes less bandwidth [4]. This is the reason why many multi processors

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

are already implementing the write invalidate protocol. This does not mean that that the write update is completely disregarded because it has a number of advantages worth exploring. Its algorithm and mechanism are usually less complex thus making it easy to implement [7]. If a modification has taken place then an updated version is immediately broadcasted to the bus. One of the disadvantages of using the write update is that it's associated with a lot of buffer due to the heavy signal traffic of the regular data update [4]. Another downside of this mechanism is that it spends a lot of time in updating and keeping all the copies valid. There is a low cache hit rate, which reduces the cache efficiency. This is some of the key reason as to why most people will go for the write invalidate but for us we have chosen the write update for our architecture since it has low miss rate and the hit rate in this mechanism is much better than the invalidate and all the copies of block is always up to date. Therefore, there is no coherence miss since all copies must be forced to be updated [16].

III. IMPLEMENTATION DETAILS

First of all we used the programming language C++ to implement our project since we are familiar with this language and its library (STL). We implemented two coherency schemes, which are Snoopy-based Protocol and Directory-based Protocol in addition to cores, caches, and blocks. In order to run and test our simulator, we had to use a separate code to generate Different Memory Access Patterns (up to 180 MB txt files). Each core is given a different set of traces, which are executed in parallel across all cores.

In this implementation, fully associative mapping is used, which prevents conflict misses completely. Block class contains a set of variables that define and track each block instance. Also, the Cache Replacement Model that we used is the least-recently used (LRU) model. Compulsory misses are tracked via a bit-array with every block in memory having 1 bit (true or false) if a cache ever accessed it.

In both schemes (Snoopy and Directory), write-through and update strategies are used in this implementation since they result in a high hit rate and low miss rate. The disadvantage of these strategies is producing traffic in the system, which leads to extra latency. In our project, we prefer to optimize cache performance in terms of hits and miss regardless of latency. The simulator measures hits and different kinds of misses. We classified misses as compulsory misses and capacity misses. Each time an update occurs, the simulator will count that as coherence forces. The simulator measures the execution time in seconds and shows the total time in the output.

In Snoopy-based Protocol, with each write process for any block, all copies of that block at all caches and main memory will be updated. Here, we have no coherence misses because caches and main memory are always updated and always have valid data. If we had to use the invalidation strategy, we could have coherence misses since any block written will invalidate other copies of the same block across other caches, and when any processor wants to access that block, it will find it invalid, which will lead to coherence misses.

In directory-based protocol, it was implemented in centralized, shared memory (one main memory). A vector of N bits, where N is number of cores, is associated with each block of memory to keep track each block (directory), and each bit is either true (block is cached) or false (block is not cached). When any write process occurs for any block, a copy of that block in main memory will be updated by write-through strategy. Then, according to directory information, the updating will be conducted only for caches that have true bits in the directory.

- *PARAMETERS*

The simulator takes the two parameters from the commend line which are number of cores and coherence scheme (0 for snoopy and 1 for directory). Then, the user can enter the block size and cache size. We used this flexibility to help us to see how these parameters affect coherency.

IV. EVALUATION AND RESULTS

A. *Hit Rate and Miss Rate using Snoopy Protocol*

Miss and hit rates in the cache are studied in order to check the percentage of hits and misses when distinct cache sizes are used, but with a constant block size and a constant number of cores of 8 byte and 4 cores respectively in the snoopy-based protocol. The following fig. shows that the hit rate is larger than the miss rate. In the case of the 32-byte cache size, the hit rate is roughly 70 percent, which is more than double the miss rate. It is obviously clear that the hit rate is increasing with the increasing cache size, while the number of misses has dropped.

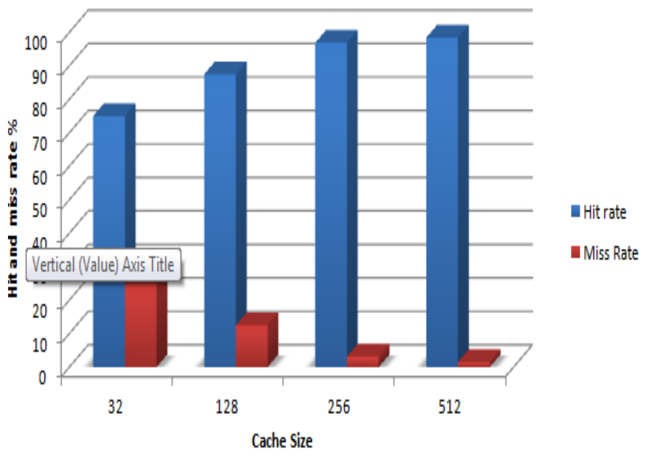

Fig. 4 Hit and Miss rate in Snoopy-Based protocol

B. *Hit Rate and Miss Rate using Directory Protocol*

We tried directory-based mechanism to study the hit and miss rate in case of various cache sizes, a constant block size of 16 B, and 32 cores. The following graph shows clearly that the hit rate is increasing, as the cache size grows larger. As long as the cache size is increasing, it is possible to contain more data so that the hit rate is increased and the miss rate is minimized.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

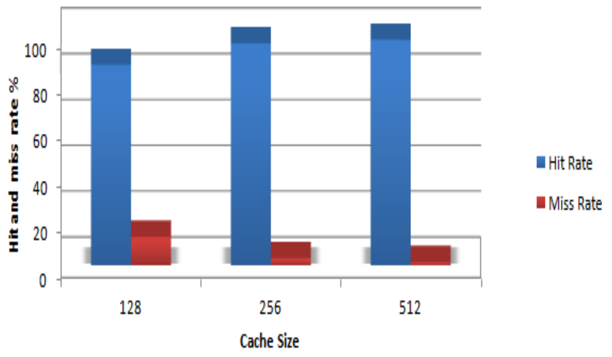

Fig. 5 Hit and Miss rate in Directory-Based protocol

C. *Cache Size Effectiveness*

Snoopy-based protocol is used with different cache sizes, constant block size (4B), and 8 cores, respectively. The following graph states that the number of hit rate increases as long as the cache size is maximized. The increase of the hit rate is attributed to the fact that the cache size plays a vital role in terms of the hit rate.

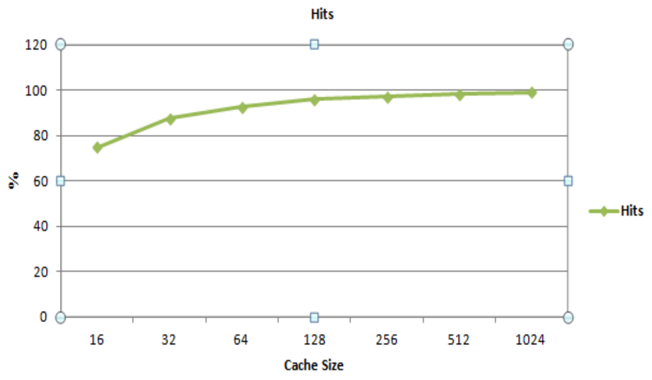

Fig 6 Hit rate vs. cache size

The graph above, Fig 6.clearly shows increasing interest as the hits rate increases as long as the cache size is increasing. On the other hand, even though the maximization of the cache size is taken advantage of with high hits rate, there is the drawback of the increase of the cache size. As a matter of fact, an increased cache size is going to lead to increased interval time to hit in the cache as we can observe that in Fig 7. Now, the implementation cost must be taken care of. There must be a tradeoff between cache size and time to hit in the cache. The following graph states that the needed time to hit in the cache increases as long as the cache size is increasing.

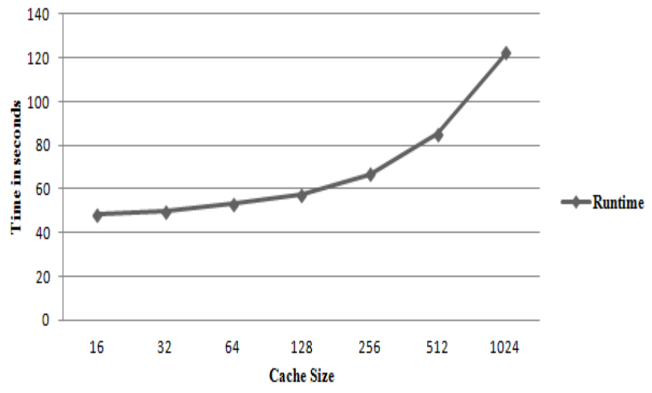

Fig 7 Runtime vs. Cache size

- **Compulsory Miss Vs. Cache Size**

There is a fact that compulsory misses cannot be avoided even in cases of very large cache size [1]. The following chart proves this fact (see figure 8). In our experiment, this fact is proven using the following parameters:

- Snoopy-based protocol.
- Different cache size.
- Block size of 4B.
- 8 cores.

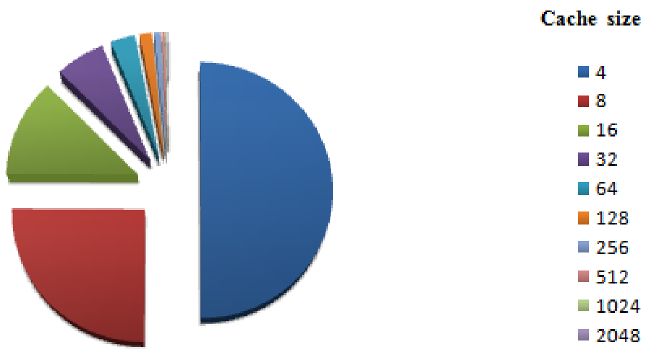

Fig 8 Compulsory Misses vs. Cache Size

- **Capacity miss Vs Cache Size**

In this experiment we used the following parameters to prove that larger cache size reduces capacity misses:

- Snoopy-based protocol.
- Different cache size.
- Block size 4B.
- 8 Cores.

As we can see in the fig.9 below, when we increase the cache size, the number of capacity misses is decreased. For example, when cache size is 4 B, the capacity miss is approximately 24%, and when we increase the cache size to 8 B, the capacity miss rate drops to approximately 12% and so on.

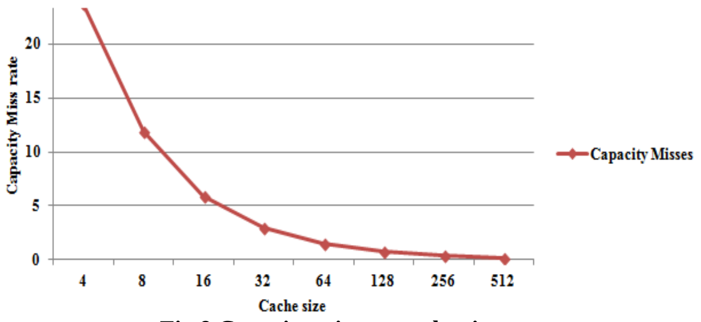

Fig 9 Capacity miss vs. cache size

D. *Block Size Effectiveness*

In this part of experiments, we studied how cache block size affects coherency forces and runtime. As a part of our implementation's goals, we were trying to verify the following facts:

- Small cache blocks can reduce the coherence forces but will lead to more time.
- Large cache blocks might cause unnecessary coherence forces due to false sharing [8].

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

We used the following parameters with different block sizes to verify the previous facts:

- Snoopy-based protocol.
- Different block size.
- Cache size of 256 B.
- 8 Cores.

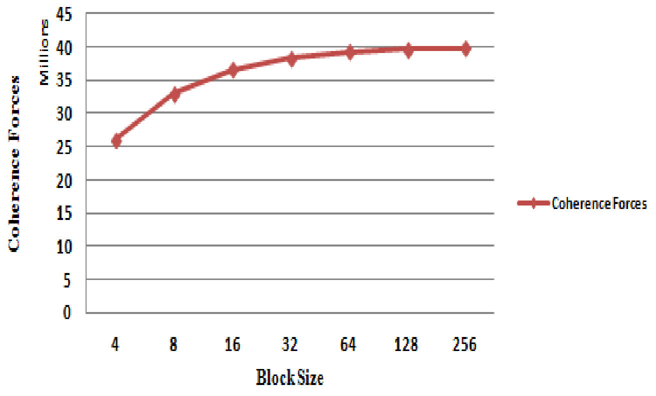

Fig 10 Coherence forces vs. block size

As we can note from the fig. 10 above, when we increase the block size, the number of coherence forces also increases. For example, when the block size is 4 B, the number of coherence forces is approximately 26 million times that, and when we increase the block size to 8 B, the number of coherence forces is also increased to approximately 33 million times. Consequently, the large cache blocks might cause unnecessary coherence forces due to false sharing.

However, as we know that small block size will lead to increased time to hit, we have been able to show in the following figure (fig.11) how cache block size affects runtime. We can also observe that an increase of the block size decreases the runtime. For example, when the block size is 4 B, it takes approximately 83 second to run, and when we increase the block size to 16 B, the runtime is dropped to approximately 60 second.

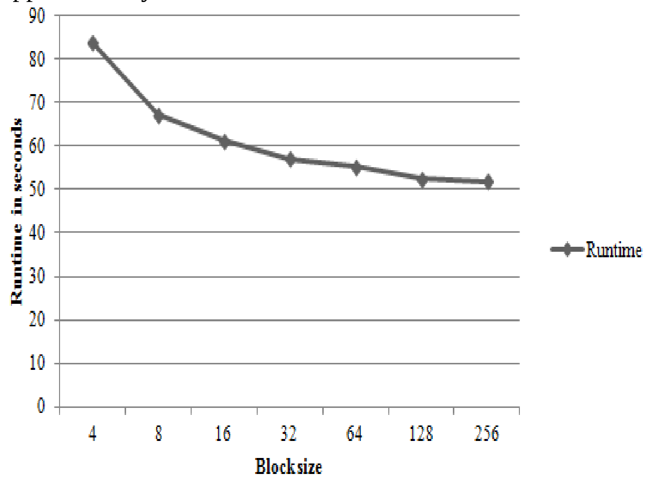

Fig 11 Runtime vs. block size

E. Different Behaviors Studies

In fig.13 bellow, we can see the relationships among the hit rate, capacity, and compulsory miss's rate. It is clear from this figure that when we increment the cache size, the hit rate also increases, and both the capacity and compulsory miss's rates are decreased. From here we can address the issue that an increase of the cache size will reduce the compulsory and capacity cache misses. We used the following parameters with different cache sizes to understand the relationship between hits and misses:

- Snoopy-based protocol.
- 4 cores.
- Different cache sizes.
- Block size of 4 B.

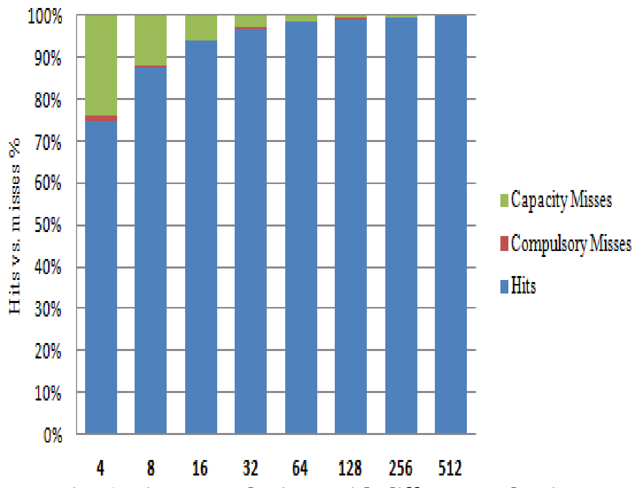

Fig 12 Hits vs. total misses with different cache sizes

In the following fig.14 (a) and (b), we can observe that the type of trace plays a large role. The miss rate almost stays the same with a different number of cores because of the update strategy that we used in our implementation and the type of trace we used.

In this experiment, we tried the following:

- Snoopy-based protocol then Snoopy-based protocol.
- Different number of cores.
- Cache size of 128 B.
- 16 B block size.
- Large trace files (180 MB for each core).

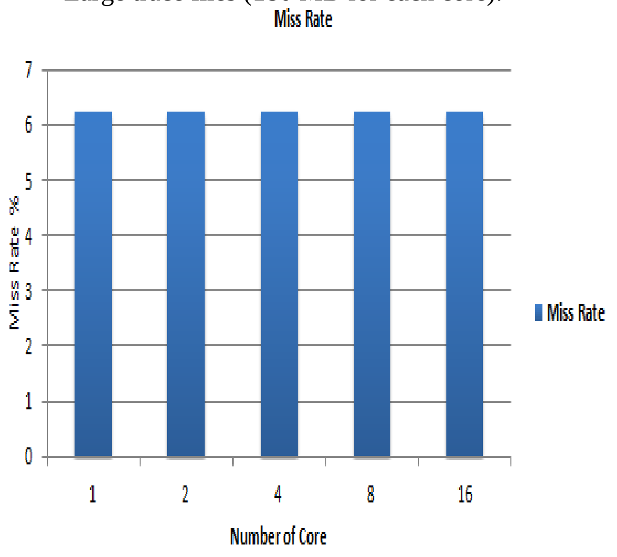

Fig 13 (a) Miss rates of Snoopy with different numbers of cores

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 4, Issue 7, January 2015

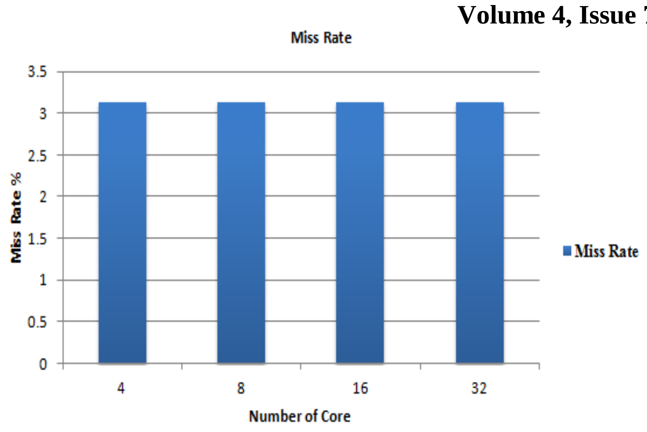

Fig 13 (b) Miss Rates of Directory with different numbers of cores

V. PROBLEMS FACED AND FIXED THROUGHOUT THE IMPLEMENTATION STAGE

- Always updated the cache write, even if the block was not present when coherence was forced (unintentional replacement).
- Counted cache coherence updates as hits, resulting in an incorrect hit count.
- Directory protocol implementation consumed a huge amount of memory, so we had to use a bit array rather than Boolean data.
- Compulsory misses were measured every time data was brought back into the cache rather than just the first time of access.
- Used bool array, which used a huge amount of memory. Reduced by using vector<bool> instead for bit field (one bit per data instead of 8 bits with 7 wasted).

VI. CONCLUSION

Today's computing systems depend in using multiprocessing systems that rely on using shared memory in order to speed up the computing process [2]. However, there is a need to a coherency protocol to control and manage the results of data in shared caches. The most common protocols that assure coherency among all shared caches are Snoopy-Based Protocol and Directory-Based Protocol. In the first part of this paper, we deeply study and investigate those two protocols and the how they are working. In the second part of this paper, we show how we implemented these two coherency schemes with update and write through strategies. In addition, we classified misses into capacity misses and compulsory misses. The coherence forces are counted instead of coherency misses since update strategy is used which keeps all caches up to date. According to the experiments we have done, it is obvious that larger cache size does not affect coherence forces, but it reduces the capacity misses. Also, the larger block size results in more coherence forces due to false sharing, which comes from independent data in same coherence block. Moreover, miss rates mostly fall with increasing block size. As well as we were able to verify the fact that said compulsory misses cannot be avoided even with an infinite cache size. Additionally, the small cache blocks can reduce the coherence forces but will lead to longer execution time. Also, increasing the cache size leads to increase the time to hit in a cache.

VII. FUTURE WORK

- Implement the invalidation and write back strategies and compare their results to our existing work.
- Improve our implementation, and use real workloads to test it.
- Implement an intelligent protocol to act as a snoopy if there is enough bandwidth and as a directory if there is less bandwidth.

REFERENCES

[1] J. Hennessy, D. Patterson. Computer Architecture: A Quantitative Approach (5th ed.). Morgan Kaufmann, 2011.

[2] Hashemi, B., "Simulation and Evaluation Snoopy Cache Coherence Protocols with Update Strategy in Shared Memory Multiprocessor Systems," Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on , pp.256,259, 26-28 May 2011.

[3] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, ''The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture,'' International Conference on Parallel Processing, pp. 764-771, 1985.

[4] Ahmed, R.E.; Dhodhi, M.K., "Directory-based cache coherence protocol for power-aware chip-multiprocessors," Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference, pp.001036, 001039, 2011.

[5] Emil Gustafsson and Bruno Nilbert,"cache coherence in parallel Multiprocessors", Uppsala 24th February 1997, Department of computer science, Uppsala university 1997.

[6] Milo M. K. Martin, Daniel J. Sorin, Mark D. Hill, and David A.: "Bandwidth Adaptive Snooping," 8th Annual International Symposium on High-Performance Computer Architecture (HPCA-8). (2002) 2-6.

[7] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir, ''The NYU Ultra computer -- Designing a MIMD, Shared-Memory Parallel Machine,'' International Symposium on Computer Architecture, pp. 27-42, 1982.

[8] Cezary D.and Thomas J.: The Effects of Block Size on the Performance of Coherent Caches in Shared-Memory Multiprocessors, UR research at university of Rochester, Technical Report (2012)

[9] David A. Patterson:. "Snooping vs. Directory Based Coherency", Lecture notes given in University of California, Berkeley" Fall 1996.

[10] Alan Jay Smith, ''Cache Memories,'' ACM Computing Surveys, Vol. 14, No. 3, pp. 473-530, September 1982.

[11] Lucien M. Censier and Paul Feautrier, ''A New Solution to Coherence Problems in Multicache Systems,'' IEEE Transactions on Computers, Vol. C-27, No. 12, pp. 1112-1118, December 1978.

[12] Leslie Lamport, ‘‘How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs,’’ IEEE Transactions on Computers, Vol. C-28, No. 9, pp. 690-691, September 1979.

[13] Richard N. Zucker and Jean-Loup Baer, ‘‘A Performance Study of Memory Consistency Models,’’ International Symposium on Computer Architecture, pp. 2-12, 1992.

[14] Michel Dubois, Christoph Scheurich, and Faye A. Briggs, ‘‘Synchronization, Coherence, and Event Ordering in Multiprocessors,’’ Computer, Vol. 21, No. 2, pp. 9-21, February 1988.

[15] David J. Lilja, David M. Marcovitz, and Pen-Chung Yew, ‘‘Memory Referencing Behavior and a Cache Performance Metric in a Shared Memory Multiprocessor,’’ Center for Supercomputing Research and Development Report No. 836, University of Illinois, Urbana, IL, 1989.

[16] Steven Przybylski, Mark Horowitz, and John Hennessy, ‘‘Performance Tradeoffs in Cache Design,’’ International Symposium on Computer Architecture, pp. 290-296, 1988.

[17] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz, ‘‘An Evaluation of Directory Schemes for Cache Coherence,’’ International Symposium on Computer Architecture, pp. 280-289, 1988.

[18] David J. Lilja,” Cache Coherence in Large-Scale Shared Memory Multiprocessors: Issues and Comparisons,” ACM Computing Surveys, Vol. 25, No. 3, September 1993, pp. 303-338.