Cache Systems: Significance And Evaluation In Multiprocessor Architectures

Sujit N. Deshpande
Lecturer,
Computer Science & Engineering Department
Walchand Institute of Technology
Solapur, Maharashtra, India

Dr. Mrs. S. S. Apte
Professor and Head,
Computer Science & Engineering Department
Walchand Institute of Technology
Solapur, Maharashtra, India

Abstract

Processor speed has increased dramatically than memory speed which results in processor-memory gap. To bridge the gap small amount of fast access memory called cache is introduced. It forms a bridge between processor and memory to provide data for execution. Cache plays an important role in organization of processor – may it be uniprocessor or multiprocessor. Inclusion of caches introduces several challenges such as coherence protocols, finding optimum cache line size, replacement policies etc. Evaluation of cache systems and innovations in it is tedious job. This will be carried out using trace driven simulators. In this paper influence of cache size on miss rate was tested. It has been observed that miss rate get decreased by increasing the cache size.

Keywords – Cache memory, coherence protocol, trace driven simulator etc.

1. Introduction

Multiprocessor demand has been increased in recent years. Uses of multiprocessors have grown from scientific and engineering applications to other areas also, such as databases, files and media servers. Multiprocessor architectures vary depending on the size of the machine and differ from vendor to vendor. Shared-memory architectures, become dominant in small and medium-sized machines, provide a single view of memory, which is shared among all processors referred as Centralized shared memory multiprocessor architectures. Like in uniprocessors, caching is used to achieve good performance in multiprocessors [3]. It reduces the latency of accesses by bringing the data closer to the processor [6] and it also reduces the communication traffic and bandwidth requirements in the network by satisfying requests without having to access the network. The presence of caches in achieving the shared memory model requires special mechanisms to maintain a coherent view of memory. These mechanisms enforce a cache coherence protocol and are usually implemented in hardware for performance. Hardware based cache coherence provides better results compare to software implemented coherence [1]. The hybrid software – hardware coherence mechanism has been proposed by [2].

Cache memory plays vital role in the performance of multiprocessor architectures. To achieve high performance it is very essential to select optimum cache size. But it is critical task to decide the size of caches in multiprocessor architectures. The size of cache is not decided randomly. Random selection of cache size affects heavily on the performance of multiprocessor architecture. There are many factors to determine the cache size like block size, associativity, number of blocks with much attention to miss ratio [8]. In this paper cache misses followed by types and cache coherence problem followed by protocols are discussed. Followed by this, idea of simulation based evaluation along with trace driven simulator is discussed. The paper is organized as follows. Section 2 gives a discussion on cache misses and its types. Cache coherence problem and protocols are discussed in section 3. Concept of simulation based evaluation along with trace driven simulation tools and memory traces is discussed in section 4. Results are given in section 5. Finally, conclusion and future work will be given in section 6.

2. Cache Misses and Types

Miss rate is one of the important metrics to measure the performance of cache systems. In the

uniprocessor architectures, cache misses are categorized into the "3 – Cs". They are compulsory, capacity, and conflict misses [7]. Many studies have examined how cache size affects each category of miss.

Compulsory misses

Compulsory misses are also referred as cold misses; occur on the first reference to a memory block by a processor. Since data cannot exist in the cache without first being brought into the cache, these misses cannot be avoided. Cold misses can only be reduced by increasing the block size, so that a single cold miss will bring in more data that may be accessed as well.

Capacity misses

They occur in fully associative mapped caches. When there is no room to map the block that is referenced by a processor during the execution of a program, based on replacement policy one of the blocks already mapped is replaced. Capacity misses are reduced by enlarging the cache.

Conflict misses

They occur in direct mapped and set associative mapped caches. If block from cache is replaced by another block and if processor makes the request for the replaced block conflict occurs. They are misses that would not have occurred in a fully associative cache. Conflict misses are reduced by increasing the associativity or increasing the number of blocks (increasing cache size or reducing block size).

Coherence misses

Cache-coherent multiprocessors introduce a fourth category of misses i.e. coherence misses. These occur when blocks of data are shared among multiple caches. Sharing is of two types: true sharing and false sharing. True sharing occurs when a data word produced by one processor is used by another. False sharing occurs when independent data words for different processors happen to be placed in the same block.

3. Cache coherence

Cache Coherence Problem

For faster access of data caches have been introduced. Inclusion of caches introduces inherent cache coherence problem. It may be possible that multiple processors have copy of same data in their cache either to perform read or write operation. Read will not create the problem but write operation to any data will change or modify the data in cache. While memory and other caches will not get the notification of data being modified. If some action is not taken, other processors will read stale copy of data. Modification to locally cached copies of a memory data block by one or more processors leads to data incoherence.

The cache coherence problem is illustrated in fig.1. Assume that memory contains location x with initial value 0. Both processors P1 and P2 read location x and cache it. Subsequently processor P0 writes 1 to x, data in processor P1's cache becomes stale. Processor P1 on reading x from cache gets old value.

Cache Coherence Protocols

Cache coherence protocols try to maintain the coherent view. A variety of cache coherence protocols have been implemented. They differ mainly by the action performed on write.

Cache coherence protocol are classified as invalidate or update depending on the notification of the changes conveyed to other processors. Consider the example with two processors P1 and P2 with private caches and shared memory containing block X. Both P1 and P2 have cached block X to perform read operation. The difference in an invalidate and update will be clear when P1 issues write operation on X. In an invalidate protocol processor P1 modifies the X from its cache and invalidates the other copies. Whereas in an update protocol it modifies the copy in cache and propagates the changes in the system. Other caches update the copy on receiving the changes.

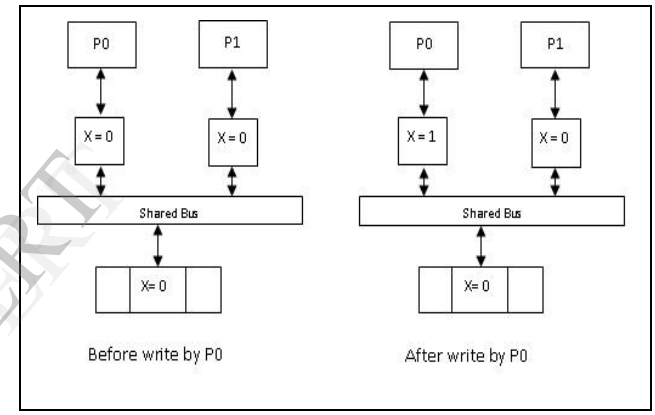

Figure 1 Cache coherence problem

Based on how memory is updated the cache coherence protocols have been classified into write – back and write – through protocols. In write – back protocol the memory is updated when any processor replaces the block from cache or when other processor requests the modified block. In write – through protocol whenever processor performs a write operation the memory is updated. Selection of an appropriate protocol plays an important role in multiprocessor systems.

To keep the track of blocks present in processor's cache coherence protocol uses states. Based on the states considered to keep the track protocols are classified as MSI, MESI, etc. MSI coherence protocol is the basic cache coherence protocol. Most of the multiprocessor systems use MSI as basic cache coherence protocol. Cache block can be in one of the three states

Modified (M): The cache block is in modified state means only this cache has the modified copy of the block. Block in memory is stale. This is also called as dirty block. The block must be written back to the memory before replacement.

Shared (S): The block is valid and is in processor's cache. Copy of the block in memory is up-to-date. The same block might be present in another processor's cache in valid state.

Invalid (I): The block is in invalid state. Block might be absent in cache and needs to be fetched from memory.
Exclusive (E) :E in MESI is for Exclusive state. The block is in current cache and status of the block is clean.
There are several cache coherence protocols used to maintain the coherent view of the system, includes snooping, directory based, token coherence etc.

Snooping Protocol
This protocol gives best performance with centralized shared memory multiprocessor architectures. This is bus based architecture. Snoopers (cache controllers) are associated with each cache memory. Cache controllers monitor the bus to check whether there is copy of the block in its cache requested by other processor. Reliance on the bus is the characteristic feature of the snopping protocol which distinguishes them from the other protocols.

Directory based Protocol
This protocol uses the concept of directory to maintain the sharing status of the block. Directory might be kept centrally or distributed. Request from the procesor might be remote or local, in case of distributed directory, is sent first to directory. Upon receiving the request directory checks the status of the block requested, and grants the request if the requested block is available. It uses several messages for the communication purpose. Interconnection network is usd as a medium.

Token – coherence Protocol
This protocol uses tokens to enfore the coherence. In this protocol each block should have fixed number of tokens. Processor should hold atleast one for read operation whereas it sholud hold all for write operation. In the designing of several cache coherence protocols framework provided by token coherence protocol can be used.

4. Trace driven simulator

It is beneficial to evaluate the performance of cache systems using simulation tools. Industry uses simulation more commonly during processor and system design because it is the simple and least expensive way to explore design options. Simulation is even more important in research to evaluate new ideas and characterize the nature of the design space. Use of trace-driven simulation in estimating the performance of computer system designs is a cost-effective method. In the designing process of caches, trace driven simulation is popular way to study and evaluate performance before builting the system.. Figure 2 explains the use of trace driven simulator.

Trace driven simulator accepts two inputs. First supplies the configurations selected by applying different parameters and second memory traces. There is also a provision to store configurations as well as memory traces. In this paper trace driven simulator, 'SMP Cache', designed and developed by ARCO Research Group, University of Extremadura, Spain is disscused. SMP Cache is windows based cache simulator used to make analysis of cache memory systems on symmetric multiprocessors.

It has graphical interface. The version SMP Cache 2.0 supports centralized shared memory bus based architecture whereas DSM Cache an advanced version of SMP Cache 2.0 supports distributed shared memory architectures. The working of SMP Cache 2.0 is explained in this paper.

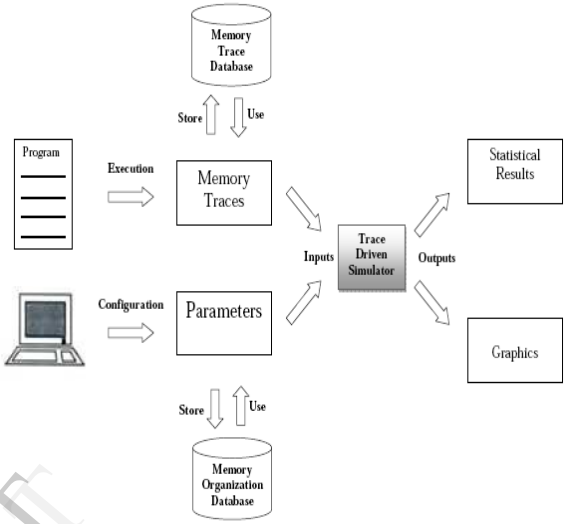

Figure 2 Use of Trace Driven Simulator

SMP Cache 2.0

Architecture and general working of the Simulator
This simulator allows users to select different configurations and work with them. It also allows storing of the selected choices for future use in file with extension " .cfg". It supports several architectural characteristics specified in Table1. User has to select one configuration by choosing various parameters. The configuration along with memory traces has to be loaded by the user. It allows running the simulation in three steps, one step by step, with break point and complete execution. It shows results in three ways, multiprocessor evolution, cache evolution and memory block evolution.

Memory traces
It consists of memory references made by processor during the execution. The extension of the trace file is ".prg". Trace files are constructed by using trace convertor. It accepts file in ".ref" format and convert the trace files. It consists of two fields, label and value. Label is decimal number specifies the memory operation requested by processor. '0' is used to represent instruction capture operation, '2' is used for memory read operation and '3' for memory write operation. Value represents effective address of the memory word referenced by processor in hexadecimal format.

No. of Processors	Upto 8
Cache coherence protocols	MSI, MESI, DRAGON
Bus arbitration schemes	LRU, LFU, Random
Bits per word	8, 16, 32 or 64
Blocks per word	1, 2, 4, 8, 16, 32, 64, 128, 256, 512 or 1024
Main memory blocks	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152 or 4194304
Cache blocks	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 or 2048
Cache Mapping	Direct, Set-Associative, Fully-Associative
No. of sets in cache	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
Replacement policies	Random, LRU, FIFO, LFU
Writing strategies	Write-Back
No. of Cache levels	1
Maximum size of block	8 KB
Max.size of main memory	32 GB
Maximum size of cache	16 MB

Table 1: Architectural characteristics [5]

Trace file is shown in Figure 3. It consists of 6 instruction captures along with 3 data reading and 1 data writing operation.

```
0 00001b08
0 00001ca5
2 00007951
0 00001d04
0 00001eb8
2 00007952
0 0000201c
2 00007c71
0 0000201f
3 00007b51
```

Figure 3. Example of trace file

Various snapshots of the loading configuration, loading memory traces, view during multiprocessor simulation, view during cache simulation and view during memory block simulation are shown in the figures (4 to 8) respectively.

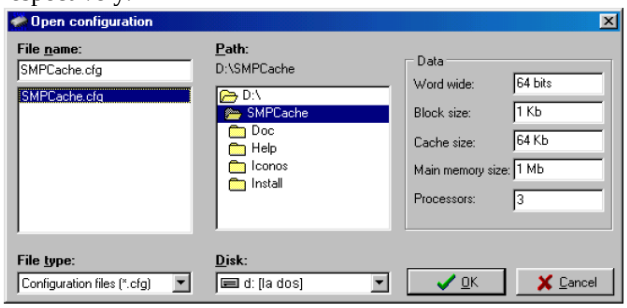

Figure 4. Loading configuration

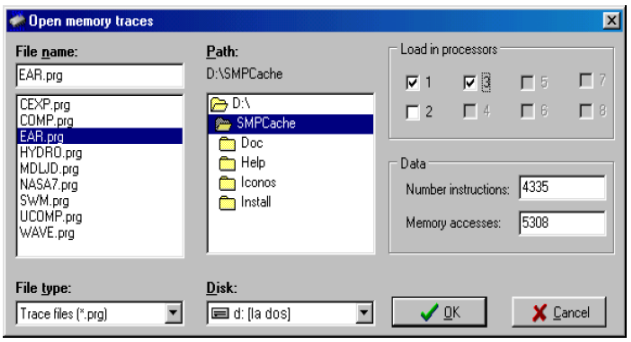

Figure 5 Loading memory trace files

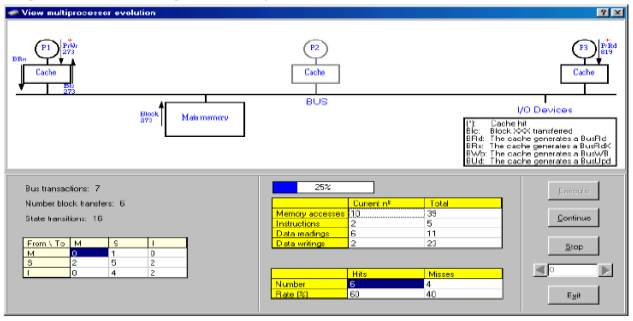

Figure 6 View during multiprocessor simulation

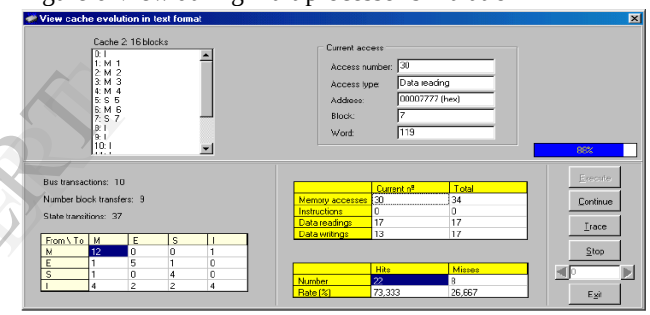

Figure 7 View during cache simulation

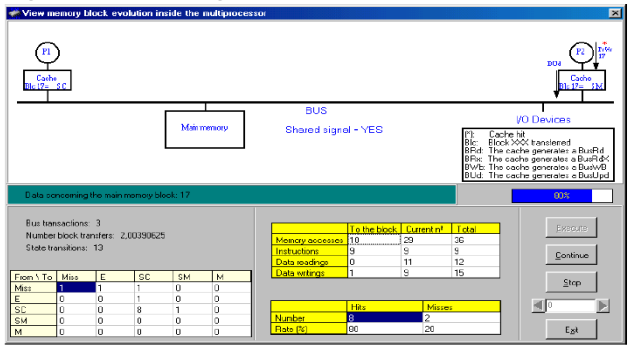

Figure 8 View during memory block simulation

5. Results

The set of experiments were carrie out with unified cache. The configuraion was selected from Table1. Selected configuration o carry out the experiments is shown in Table 3. By varying number of blocks the influence of cache size over miss rate was examined. Figure 9 displays global miss rate for two traces SPEECH and SIMPLE by varying cache size from (1KB to 128 KB). The Table 4 and Table 5 shows the detailed results for SPEECH and SIMPLE memory trace files respectively including number of hits, hit rate, number of misses, miss rate for the cache size 1KB to 128 KB.

Table 3 and Table 4 show that for SPEECH and for SIMPLE traces 128 KB cache size shows minimum miss rate.

Number of Processors	8
Coherence Protocol	MESI
Bus Arbitration	LRU
Word Wide (Bits)	16
Word by Block	32
Blocks in Main Memory	262144
Block Size	64 Bytes
Main Memory Size	32 Mbytes
Blocks in Cache	Varies
Cache Size	Varies
Mapping	Set Associative
Number of Cache Sets	Varies
Replacement Policy	LRU
Cache Levels	1
Writing Strategy	Write Back

Table 3. Selected configuration.

Cache Size (in KB)	# Hits	Hit Rate (%)	# Misses	Miss Rate (%)
1	1426490	12.118	10345174	87.882
2	1426490	12.118	10345174	87.882
4	4148893	35.245	7622771	64.755
8	5017752	42.626	6753912	57.374
16	5521651	46.906	6250013	53.094
32	7035183	59.764	4736481	40.236
64	7214731	61.289	4556933	38.711
128	7287619	61.908	4484045	38.092

Table 4. Hit and Miss rate for SPEECH trace file

Cache Size (in KB)	# Hits	Hit Rate (%)	# Misses	Miss Rate (%)
1	10481957	38.779	16548135	61.221
2	10481957	38.779	16548135	61.221
4	11967766	44.276	15062326	55.724
8	14888574	55.081	12141518	44.919
16	17203493	63.646	9826599	36.354
32	17491493	64.711	9538599	35.289
64	17767655	65.733	9262437	34.267
128	17835789	65.985	9194303	34.015

Table 5. Hit and Miss rate for SIMPLE trace file

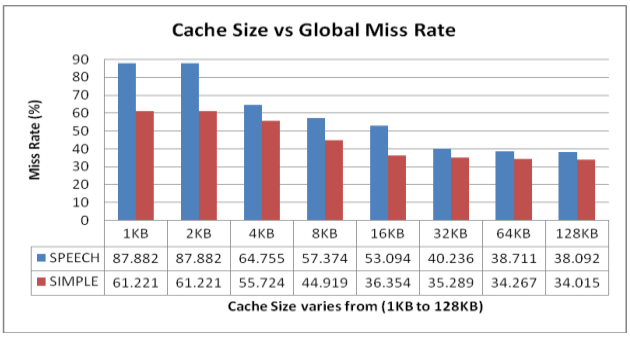

Figure 9 Miss Rate vs Cache Size

6. Conclusion and Future Work

Study of cache systems and its evaluation can be easily carried out by using trace driven simulator SMP Cache2.0. Trace driven simulators are really helpful in getting the results with wide variety of configurations and analyze them than the actual system implementation. In this it has been observed that miss rate decreases by increasing the cache size. But it gets saturated at one point further increasing the cache size it won't affect on miss rate.

By understanding the use of simulator any one can use it to carry out the pracical work. It can be used to perform the evaluation of cache systems in multiprocessor architectures and to find the optimum performance of the system.

7. Acknowledgements

We are very much thankful to Vega-Rodriguez, Miguel A., Associate Professor and Member of ARCO Research Group at Department of Technologies of Computer and Communications, University of Extremadura (Spain), for providing SMPCache 2.0 – trace driven simulator for research work.

8. References

[1] Milo M.K. Martin, Mark D. Hill, and Daniel J. Sorin. "Why On-Chip Cache Coherence is Here to Stay" In communications of the ACM July 2012, vol. 55, no. 7. Page(s): 78-89.

[2] Thomas J. Ashby, Pedro Dı´az, and Marcelo Cintra. "Software-Based Cache Coherence with Hardware-Assisted Selective Self-Invalidations Using Bloom Filters", IEEE Transactions On Computers, VOL. 60, NO. 4, April 2011 Page(s): 472 – 483.

[3] Sujit Deshpande, Priya Ravale, Sulabha Apte, "Cache Coherence In Centralized Shared Memory And Distributed Shared Memory Architectures", International Journal on Computer Science and Engineering (IJCSE), ISSN : 0975-3397, February 2011 Page(s): 39 – 44.

[4] Miguel A. Vega-Rodríguez, R. Jorge Gil-Ramos, Juan A. Gómez-Pulido, Juan M. Sánchez-Pérez. "A Versatile Simulator for Cache Memories on DSM Systems", In the proceedings of 19th European Conference on Modelling and Simulation ISBN 1-84233-112-4, 2005.

[5] Miguel Ángel Vega Rodríguez , Juan Manuel Sánchez Pérez, Juan Antonio Gómez Pulido. "An Educational Tool for Testing Caches on Symmetric Multiprocessors", Microprocessors and Microsystems, vol. 25, Page(s)187–194, June 2001.

[6] "A New Approach for the Verification of Cache Coherence Protocols" Fong Pong, Member, IEEE, and Michel Dubois, Senior Member, IEEE Computer Society IEEE Transactions On Parallel And Distributed Systems, VOL. 6, NO. 8, AUGUST 1995 Page(s): 773-787.

[7] Mark D. Hill and Alan Jay Smith. Evaluating Associativity in CPU Caches. IEEE Transactions on Computers, vol. C-38, no. 12, December 1989, Page(s) : 1612-1630.

[8] A.J. Smith, "Line (block) size choice for CPU cache memories", IEEE transaction on Computers, vol. 100, no 9, September 1987, Page(s): 1063-1075.