مجلة جامعة سبها للعلوم البحتة والتطبيقية
Sebha University Journal of Pure & Applied Sciences

Journal homepage: www.sebhau.edu.ly/journal/index.php/jopas

Simulation Cache Coherence Protocols in Multicore Processors

*Anis Elgarduh[1], Abdalhamed Alkawash[2], Abdelmhsan Elbandac[3]

[1]Dept. of Computer Science, University of Derna, Libya
[2]Dept. of Computer, College Of Technical Sciences, Libya
[3]Higther Institue of Agricultural Techniques, Tarhunah, Libya

Keywords:
Cache Coherence
Directory Based
Multicore
Memory
Snooping

ABSTRACT

The cache coherence problem is the challenge of keeping multiple cache synchronized when one of the processors update its local copy of data which is shared among multiple cache. This paper discusses several different varieties of cache coherence protocols including with their pros and cons, and using simulation technique it will address this problem and compare between two protocols that use to solve it: Directory-based protocol and Snooping protocol. Simulation results have shown that snooping based systems are appropriate for high bandwidth systems while directory-based cache coherence protocols are suitable for lower bandwidth systems.

محاكاه لبرتوكولات الترابط بالذاكرة المخبأة في المعالجات متعددة النواه

*أنيس القردوح [1] و عبدالحميد الكواش [2] و عبدالمحسن البنداق [3]

[1] قسم. حاسب آلي ، جامعة درنة ، ليبيا
[2] قسم. حاسب آلي ، كلية العلوم التقنية ، ليبيا
[3] المعهد العالي للتقنيات الزراعية ، ترهونة ، ليبيا

الملخص

التحدي في معالجات متعددة النواه هي مشكلة ترابط الذاكرة المخبأة حيث الهدف هو الحفاظ على تزامن الذاكرة المخبأة لكل نواه وتحدث عندما يقوم أحد المعالجات بتحديث نسخته المحلية من البيانات والتي يشترك بها مع ذاكرات مخبأة لمعالجات اخرى. تناقش هذه الورقة نوعين من بروتوكولات ترابط الذاكرة المخبأة بما في ذلك مزاياها وعيوبها، وباستخدام تقنية المحاكاة سوف نقارن بين بروتوكولين يستخدمان لحل هذه المشكلة وهما بروتوكول القائم علي دليل وبروتوكول الاستطلاع. أظهرت نتائج المحاكاة أن الأنظمة القائمة على برتوكول الاستطلاع مناسبة للأنظمة ذات النطاق الترددي العالي بينما يعد برتوكول القائم علي الدليل مناسب للأنظمة ذات النطاق الترددي المنخفض.

الكلمات المفتاحية:
الترابط المنطقي للذاكرة المخبأة
برتوكول القائم علي الدليل
برتوكول الاستطلاع
الذاكرة
معالجات متعددة النواه

Introduction

CPU cache is a fast but small memory unit that stores frequently accessed memory blocks in order to minimize delays. According to Denning's locality principle [1], most computer programs most of the time work with a very small memory set. In a multicore environment cache coherence is a concern because of distributed L1 and L2 caches. due to each core has its own memory, the copy of the data may not always be the last up-to-date in that cache. For example, if we have a dual-core processor where each core has a block of memory, and then one core writes a value to a specific location. When the others core tries to read that value from its cache, it will not have the last up-to-date of data unless its cache entry is invalidated and a cache miss occurs. This cache miss forces the second core's cache entry to be updated. If this coherence policy was not in place, the wrong data would be read and invalid results would be produced, but whit a single core, cache works completely deterministically using these simple rules for loading and replacing cached blocks. However, for

Corresponding author:
E-mail addresses: anis.elgarduh@omu.ed.ly, (A. Alkawash) abdalhamed.alkawash@phid.edu.ly, (A. Elbandac) ayhimlara@gmail.com
Article History: Received 18 April 2022 - Received in revised form 04 July 2022 - Accepted 03 October 2022

multiprocessors [2] there arises a problem of maintaining cache coherency between the cores.

This is called the cache coherence problem, and a set of rules that governs how multiple caches interact in order to solve this problem is called a cache coherence protocol. In general, there are two protocols cache coherence which are a snooping protocol and a directory-based protocol. This paper gives an emphasis on the study and analysis of impact of various system parameters on the performance of the basic techniques to identify appropriate cache coherence protocol for various architectures.

Literature Review

The Directory-based cache consistency protocol displays superior performance for multicores over the Snooping-based cache consistency protocol. The Directory-based cache coherence protocol outperforms multicore on the Snooping-based cache coherence protocol. In the system of high bandwidth, Snooping doing well because of lower overhead and reduced congestion, but the directory-based system is not suitable for that class of systems. In contrast, directory in high bandwidth system is doing well, whereas the Snooping is not [3].

The mechanisms of directory are requests / forwards / responses messages. These massages are point-to-point

not broadcast so that it tends to have longer latency but use less bandwidth, while the mechanisms of Snooping are broadcast request to all processors in the systems. Since Each request must be broadcast to all nodes in the system, the system will become larger and bandwidth must be extended. Because directory-based protocol uses much less bandwidth than Snooping, it is widely used for larger systems to enhance their performance [4],[5].

The cache coherence problem with multiple cores

Clearly, each core has to have its own first-level cache, since cores may be working with different programs and thus the locality principle does not scale well to the core set as a whole.

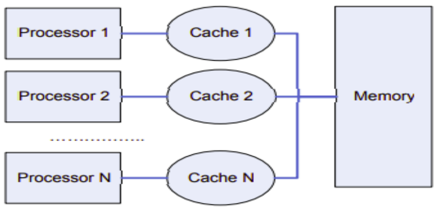

Fig. 1. Multiprocessors with shared Memory

Memory access itself is synchronized by the memory bus. However, it may happen those different caches store different versions of the memory block. This is referred to as cache coherence problem. In other word, the cache coherence problems come from that a copy of the same memory block may be store in more than one cache, and it is important to make sure that the data is consistent with each other in all caches. The next example shows the coherency problem in multicore processors.

TABLE I. COHERENCY PROBLEM EXAMLE

Time	Event	Cache content for CPU A	Cache content for CPU A	Memory content for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 int X	0	1	0

From the previous table, the cache coherency problem happens in the last step because the data in both caches are not consistent.

In the hardware-based method, there are two main approaches to solve this problem, which are invalidation and update. In the former, it is assumed that the last copy of cache block, which has changed is the only valid one. Figure 2 shows the invalidation. While in the latter, each modification of a cache block is transferred to all other caches. Figure 3 shows the update [6].

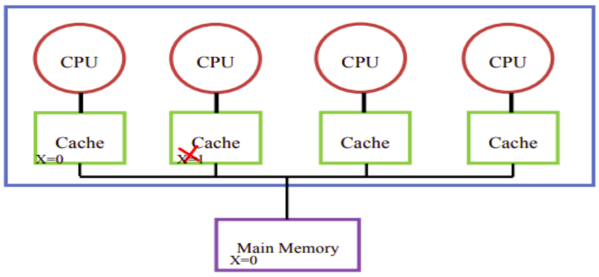

Fig. 2. Invalidation

In this case, core A writes 0 to X and sends invalidation request to all other caches, so X's value in core B will invalidate.

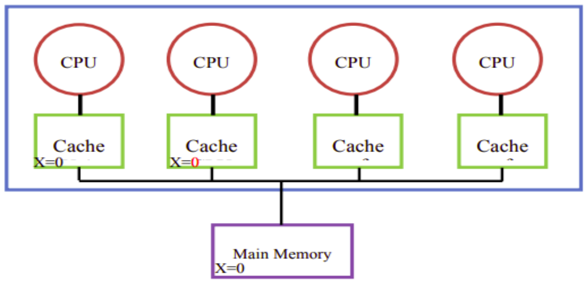

Fig. 3. Update

In this case, core A writes 0 to X and broadcast updated value to all other caches, so the value in core B will be updated.

Cache coherence protocols

There are two basic ways of dealing with the cached block that is about to become inconsistent: Update and Invalidation.

Updating assumes that the changes will be broadcasted to all caches which store the same block and/or the main memory. Invalidation postpones the updating and merely cache blocks as invalid. They will be updated later from the memory. For a practical implementation of these approaches, one needs a way to pass data between caches and notify them of changes.

There are two approaches for that [7]: Directory-based protocol [8] and Snooping protocol [9].

The latter approach is generally faster when the number of cores is small. However, with growing number of cores the line becomes busy to often and lots of local caches have to be invalidated each time, which slow down the whole system. For this reason, on large multiprocessor machines the directory-based approach is preferred.

A- The directory-based protocol

In the directory-based protocol a common directory of cached blocks is maintained. Processors must notify it when caching data from shared memory to the local cache. The directory controller is able to invalidate or update caches of other processors when a write operation occurs. This approach is generally slower due to the necessity of the directory notification. In short, the idea of this protocol is to keep track in a directory of which processors are caching a location and the state. This protocol is used in AMD processors, such as 12-core Opteron [10].

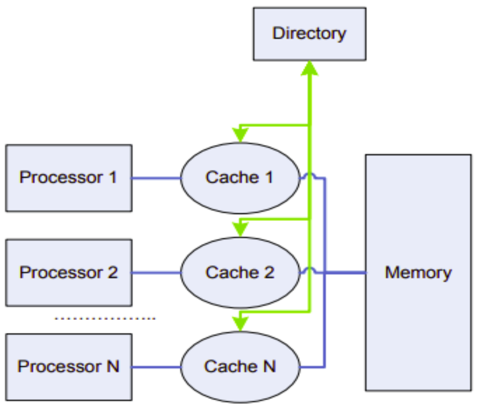

Fig. 4. Directory-based Protocol

The directory-based approach uses a shared global consistency object (the directory), which is able to communicate with caches in both directions. A cache can send a message to the directory requesting a block load. The directory maintains a list of caches, which store each entry and can notify them of changes in the cached entries. The directory-based protocol has three states for the cache blocks. These states are shared, uncached, and exclusive. Shared means one or more processors have a read-only copy of the data block and the main memory is up-to-date. Uncached means that the copy of data 7 block means no processor has a copy of the data block and the main memory is up-to-date. Finally, exclusive means only one processor has the copy of data, which is the owner, and the main memory is not up-to-date [11]. However, directory can be implemented as a linked list of cores (their indices) for each cached data block (address). Also, a full map directory structure is used, which contains a pointer for each cache in the system, so every cache can store a copy of any block of data in the global memory [12].

- Pros:
  - ✓ Small overhead
  - ✓ Fairness (directory can apply some fairness criteria to guarantee that every core receives access to the memory block in turn) •
  - ✓ Scalability (directory can be implemented e.g., hierarchically)
- Cons:
  - ✓ Requires separate data channel that connects caches and the directory
  - ✓ Slowdowns due to synchronization (only one cache can query the directory at a time)
  - ✓

Snooping protocol

The second approach is snooping protocol, which is the simplest protocol. It does not use any kind of global directory. Instead, it uses a shared bus between processors and memory called snoopy bus. Each processor monitors ("snoop") the bus through which all the cores communicate with the shared memory. All transactions are a request/response seen by all cache controllers and processors [13]. For each bus transaction, the snoopy protocol causes some rely on a bus to make all the cache controllers able to see the activities that all other processors did. Then the cache controller compares the block address in the bus with the address in the cache to find if it has a copy of that block or not, and then decide on which action should to do to prevent staling data [14]. In short, this protocol ensures that the processor has an exclusive access to the data, when write operation is performed. This protocol is used in Intel processors [15].

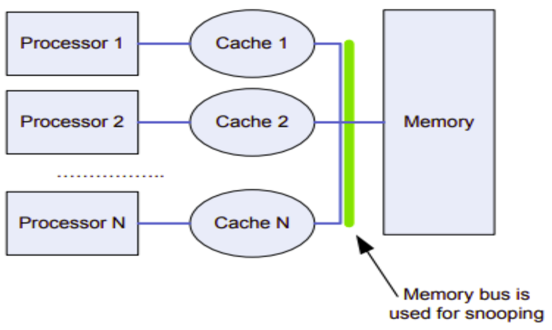

Fig. 5. Snooping Protocol

When a writing operation is broadcasted, A processor broadcast a write over the bus, and then all other caches have to invalidate their copy of a block before modifying it by a process. After invalidating, the processor, which requests the write, is sure that there is no processors received old data. This protocol called write invalidate and also known as a write once protocol. Most modern cache coherence multiprocessors use this protocol because it is easy to implement in hardware, generates less bus traffic and uses spatial locality, which means one transaction per cache block, but it could cause cache miss [16][17].

- Pros:
  - ✓ No separate data channel is needed (caches use only the bus, to which they already have access)
  - ✓ Low miss-latency (cache sees invalidation message on the bus immediately, without querying the directory)
- Cons:
  - ✓ Bad scalability
  - ✓ No fairness guarantees

To implement this protocol, cores must send all writes directly to memory in order to notify other cores about the update. Also, cores must "snoop" the bus and compare each address that passes though it with the list of cached pages and invalidate them as necessary. Alternatively, a finite state machine can be used to maintain Valid, Dirty and Shared states of the page. When requesting a page (due to hit miss), cache can receive it from the memory or from another cache that has modified it.

Comparing the performance of protocols

In order to compare the protocols a Java program simulating them was developed for this project. Each core is simulated as a thread and performs a sequence of randomized memory reads/writes [18]. Simulation is run for 10 seconds. Then the following two quantities are compared:

• Total number of reads
• Number of writes relatively to the number of reads
• Total amount of data processed by all threads

In the first two items only reads/writes that pass through the global bus are counted, i.e. cache-misses. These measurements approximate the well-known Average Memory Access Time (AMAT) statistics [19] in a manner adapted to the task of comparison of scalability of two algorithms.

The results we get of running the program are not realistic since the cores are simulated using Java threads, and Java thread run sequentially while in the real system, the cores run in parallel.

Of course, the true slowdown factor and critical number of cores are hardware dependent and are likely to be significantly different in reality. The simulation only demonstrates general trends. A precise simulation would require replacing Java threads and locks with a sequential model to make it independent from the OS scheduler. This model should use interlocking rules based on the processor and bus circuits.

TABLE II. FIRST SIMULATION RESULRS

Input Configuration				Output Configuration	
Memory Size	**Number of Cores**	**Cache Size**	**Protocol Mode**	**Fraction of write**	**Miss Rate**
1024	16	512	2	43%	87%
1024	4	512	2	65%	67%
1024	4	512	1	44%	64%
1024	16	512	1	34%	85%

The miss rate and the fraction of write in the previous table were changed depending on the number of cores and the protocol that was used. We found improvements in the miss rate and fraction of write when the number of cores becomes smaller for the snoopy protocol. Thus, our expectation that the snoopy protocol is good for small systems is true. However, we did not find any improvements in the miss rate and the fraction of write when the number of cores becomes larger for directory-based protocol. Thus, our expectation that the directory-based protocol is good for the large systems is not true in our simulator since a shared directory has been used in the simulation.

TABLE III. SECOND SIMULATION RESULRS

Input Configuration				Output Configuration	
Memory Size	**Number of Cores**	**Cache Size**	**Protocol Mode**	**Fraction of write**	**Miss Rate**
1024	8	32	1	30%	93%
1024	8	32	2	37%	97%
1024	8	256	1	34%	80%
1024	8	256	2	44%	85%
1024	8	512	1	34%	78%
1024	8	512	2	48%	82%

From the second table, we found that when the cache size increases, the miss rate decreases because the cores will be able to store more data in their caches, and that decrease read/write misses. In addition, we found that the fraction of write becomes better when we increase the cache size. Figures 7, 8, and 9 visualized these results.

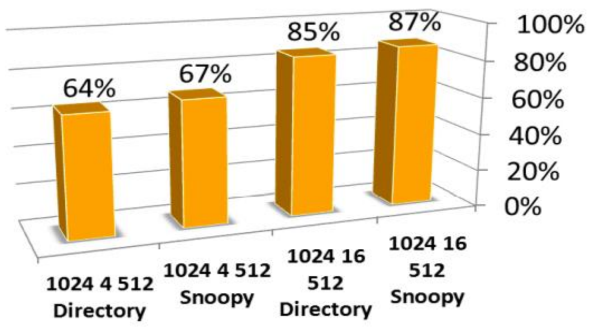

Fig. 6. Miss Rate Chart

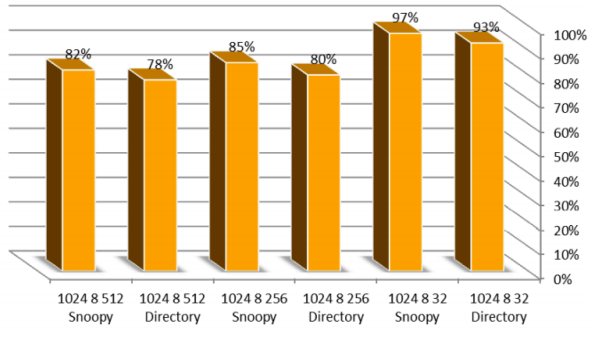

Fig. 7. Miss Rate Chart

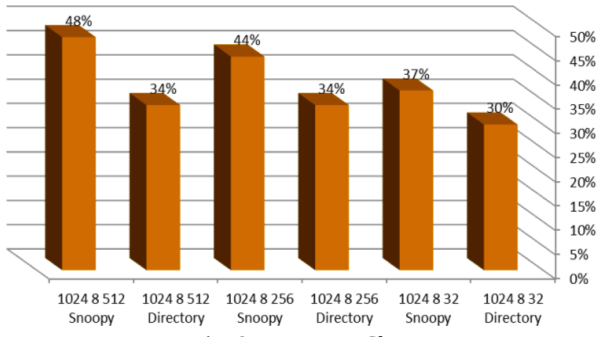

Fig. 8. Miss Rate Chart

Conclusion

Although the snoopy protocol tends to be faster if there is enough bandwidth available, it is not scalable and no longer appropriate due to limitation of the bandwidth that requires to broadcast messages to all processors. That means, when we have a large system, the size of the bus and the bandwidth have to be large enough.

However, the directory-based protocol is designed because the state of a block can no longer be determined by placing a request on shared bus as in snoopy protocol. Therefore, the main advantage of using this protocol is using less bandwidth since messages are point to point. Due to this reason, most of large systems use the distributed directory base protocol.

References

[1] P. Denning, “The Locality Principle,” Communications of the ACM, vol. 48, no. 7, pp. 19– 24, Jul. 2005.

[2] J. Hennessy and D. Patterson. Computer Architecture: A Quantitative Approach, 4th ed. San Francisco: Morgan Kaufmann, 2006.

[3] Amit Joshi, Satyanarayana Vollala, Shameedha Begum, N. Ramasubramanian.“Performance Analysis of Cache Coherence Protocols for Multi-core Architectures: A System Attribute Perspective” the International Conference. AICTC ’16, August 12-13, 2016, Bikaner, India.(3)

[4] Samaher Al-Hothali, Safeeullah Soomro, Khurram Tanvir, Ruchi Tuli.” Snoopy and Directory Based Cache Coherence Protocols: A Critical Analysis” Journal of Information & Communication Technology Vol. 4, No. 1, (Spring 2010) 01-10. (4)

[5] ManojJadhav, G. Gopichand. “Directory Based Cache Coherence Modellar in Multiprocessor using Scalable Cache Coherence(SCI)”. International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682 Vol.5, No.3, march 2016.(5)

[6] M. Marty, M. Hill, “Cache coherence techniques for multicore processors,” Ph.D dissertation, Univ. Wisconsin., Madison, WI, 2008. (6)

[7] D. Patterson and J. Hennessy. Computer Organization and Design (The Hardware/Software Interface), 4th ed. San Francisco: Morgan Kaufmann, 2008. (7)

[8] M. Bauman, E. Rodi, D. Morrissey, “Directory based cache coherency system supporting multiple instruction processor and input/output caches,” U.S. patent 6 438 659, August 20, 2002. (8)

[9] M. Koster, C. Johnson, B. O'Krafka, “Snooping-based cache-coherence filter for a pointto-point connected multiprocessing node,” U.S. patent 7 698 509, April 13, 2010. (9)

[10] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, B. Hughes, "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor," Micro, IEEE, vol.30, no.2, pp.16,29, March-April 2010 (10)

[11] B. Schauer. (2008, 11). Multicore Processors: A Necessity [online]. Available: http://www.csa.com/discoveryguides/multicore/review2.php?SID=s0s5lp3hcg6fa6pf2ba2 8u84d0 (11)

[12] D. Chaiken, C. Fields, K. Kurihara, A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," Computer, vol.23, no.6, pp.49,58, June 1990. (12)

[13] S. Lametti. (2010, Desember 1). Cache Coherence Techniques [PDF]. Available: http://www.di.unipi.it/~vannesch/SPA%202010-11/Silvia.pdf

[14] J. Archibald, J. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Transactions on Computer Systems (TOCS), vol.4, no.4, pp.273,298, Nov, 1986.

[15] D. Levintha. (2009). Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors. [PDF]. Available: http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide. pdf (15)

[16] W. Yen, D. Yen, King-Sun Fu, "Data Coherence Problem in a Multicache System," Computers, IEEE Transactions on , vol.C-34, no.1, pp.56,65, Jan. 1985. (16)

[17] K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hill, New York, NY, 1998. ISBN 0-07-031798-4. (17)

[18] S. Asmussen and P, Glynn. Stochastic Simulation: Algorithms and Analysis. Springer, 2007. (18)

[19] C. Kozyrakis. (2008). Advanced Caching Techniques. [PDF]. Available: http://www.stanford.edu/class/ee282/08_handouts/L03-Cache.pdf (19)