International Journal of Hybrid Information Technology
Vol.8, No.2 (2015), pp.367-378
http://dx.doi.org/10.14257/ijhit.2015.8.2.34

Hybrid Shared-aware Cache Coherence Transition Strategy

Sun Sun, Hong An and Junshi Chen

School of Computer Science and Technology
University of Science and Technology of China
No.96, Jinzhai Road, Hefei, Anhui, China
sunsun@mail.ustc.edu.cn, han@ustc.edu.cn, cjuns@mail.ustc.edu.cn

Abstract

Chip-multiprocessors have played a significant role in real parallel computer architecture design. For integrating tens of cores into a chip, designs tend towards with physically distributed last level caches. This naturally results in a Non-Uniform Cache Access design, where on-chip access latencies depend on the physical distances between requesting cores and home cores where the data is cached. Therefore, data movement and management impact access latency and power consumption. Remote misses limit the performance of multi-threaded applications, so using data locality is fundamental importance in Chip-multiprocessors. In this work we observed that a shared data writing behavior dramatically wastes precious on-chip cache resource and seriously affects the whole system performance. Therefore, we emphasis on improving the performance of applications that exhibit high data sharing, and propose a new prediction mechanism to predict accurately the impact of shared data and a scalable, efficient hybrid shared-aware cache coherence transition strategy which collaborate with directory-based MESI cache coherence protocol. In order to evaluate our proposal transition strategy, we experiment with the NAS Parallel Benchmarks and a modern Intel Harpertown multi-core machine. Results show the whole performance gains of up to 21% opposed to the traditional write-invalidate cache coherence transition strategy.

Keywords: *Chip-multiprocessors; Non-Uniform cache access; Remote misses; Shared data; Cache coherence protocol*

1. Introduction

Emerging multi-core architecture provides parallel execution containing two or more cores to execute multi-threaded applications based on shared memory. Today, all major chip vendors have their multicore products on the market and trends indicate that future multi-core structure will have a large variety of core numbers, on-chip cache hierarchy, and interconnected structure. However, compared to computational resources, hardware caches occupy larger chip space, play a significant role in accelerating program executions and improving the throughput of the whole system [1, 2]. Unfortunately, hardware cache management cannot perceive cache contention and collaboration between co-running threads; uncontrolled execution of co-running threads would significantly deteriorate program executions [3-5] and system throughput [6, 7]. Additionally, multi-core in Chip

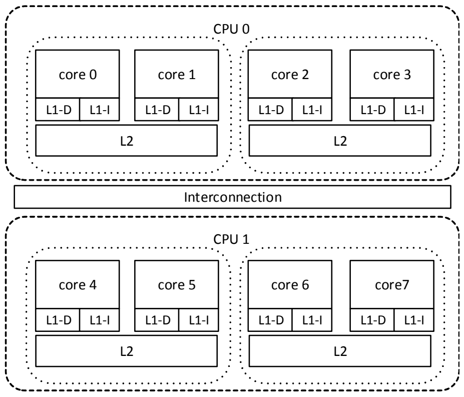


Figure 1. Distributed LLC Organization in Multi-core System

Multi-processors (CMPs) systems are usually organized into multiple groups, which are called core-groups or memory-domains [8]. This organizational style makes on-chip access latencies depend on the physical distances between requesting cores and home cores where the data is cached. Figures 1 shows the high-level view of a commercial multi-core architecture, which results in a Non-Uniform Cache Access (NUCA) design. When core 0 on CPU0 accessing a datum A, this data is placed in the last level cache (LLC) of CPU1, then it will cause a Remote Miss and injure performance of parallel shared memory application. As parallel applications becoming the standard to take advantage of multi-core architectures, it is important to consider the shared data of threads when executing a parallel shared memory application on distributed LLC organized CMPs architectures. Many works targeting at improving LLC efficiency focused on such general architectures [3, 6], including high performance interconnects, more effective threads placement, sophisticated caching and coherence mechanisms. This paper focus on the latter.

Typical cc-NUMA systems use directory-based MESI cache coherence protocol with write-invalidate transition strategy to maintain cache coherence [9]. Before a processor can modify a cache line of data, it must invalidate all remote copies. However, this write-invalided transition strategy is not efficient for those applications that seem to have a lot of shared data between threads. For example, when a subset of threads that are mapped into different core-groups access a shared data item, one of threads writes the datum, so it must invalidate all remote copies and a lot of invalidate requests are created. While another thread reads the same datum, a cache miss occurs and starts a cache-to-cache transfer to re-load the newest data due to the invalidation of that block. Therefore, traditional write-invalided cache coherence transition strategy results in plenty of replication, invalidation, and cache misses because of this shared data access pattern. In this paper, we propose a scalable, efficient hybrid shared-aware cache coherence transition strategy that collaborates with directory-based MESI cache coherence protocol to eliminate remote misses comparing to traditional write-invalidate cache coherence transition strategy. To exploit selective write shared-data update, we first reveal the write-shared-data pattern of parallel applications. Initial data access operation continues to be handled via native write-invalidate transition strategy, however, after we identify cache lines that exhibit a stable write shared data access pattern, we extended conventional write-invalidate strategy to write-update strategy to maintain cache coherence. For instance, when a core modifies a cache line, it requests exclusive access to the cache line. A normal directory-based write-invalidate cache coherence strategy invalidates all other cache lines that have a shared copy. In the directory, we use Strategy Counter to count the degree of shared data. If Strategy Counter of access cache line is higher than the Strategy Threshold, we will take the write-update transition strategy to process this access, or else the initial write-invalidate transition strategy is used. We evaluated our proposed technique with Sniper simulator

[21]. The system architecture we used in our simulations is presented in Figure 1. Besides, we used the OpenMP implementation of the NAS Parallel Benchmarks [10] to evaluate our proposal. Experimental results show that our mechanism outperforms conventional popular directory-base write-invalidate cache coherence protocol in all of tested benchmarks. It accelerates these programs by 21%, cache misses and cache-to-cache transfer was reduced by up to 25% and 30% in average.

The rest of this paper is organized as follows: Section II discusses the related researches. In section III we introduce our motivation and the background information of write-update and write-invalidate cache coherence transition strategy. Section IV details our proposal techniques and its implementation. Section V shows how our proposal was evaluated, and the results of our experiments. Section VI concludes our work.

2. Related Works

Effective use of on-chip hardware cache resource is important for improving the performance of CMPs system. There is a lot of cache management work aiming at more effective threads placement, and sophisticated caching and coherence mechanisms.

In the terms of threads placement, it has been studied in previous works [6, 11] that the execution time of a thread can vary greatly relying on which co-run threads in the same core-group. The main reason is the interaction of co-executing threads. In addition, this phenomenon is particularly true if several cores share the same LLC. Cruz [11] evaluated a technique to collect the communication pattern of the treads of parallel applications. With these patterns, they created a static thread mapping strategy to measure the performance, and place the threads that communicate a lot into a core-group. Azimi [12] showed that they estimate the communication pattern of threads based on stall cycles, cache misses and other performance events which can directly been acquired by Performance Monitor Unit provided in modern processors. Through this method, they can dynamically map threads of parallel applications roughly. Although sophisticated threads placement achieves some performance improvement, this is just a static placement strategy, still can’t dynamically adjust threads mapping following the real time communication pattern accurately.

For some coherence mechanisms, LLC organization incurs significant protocol latencies when a writer of a data block invalidates multiple readers [13] in CMPs; the impact is directly proportional to the degree of shared data block. Kurian [14] propose a locality-aware adaptive coherence protocol to manage the distributed private caches in CMPs. When a core requests a cache datum that misses the private cache, the coherence protocol either brings the entire cache data block using a native directory-based MESI protocol with write-invalidate transition strategy, or just remote acquires the requested word at the shared cache location. This method decreases the access bandwidth effectively. Cheng [15] proposes a directory delegation mechanism whereby the “home node” of a cache line can be delegated to another node. Other nodes that learn the delegation can send requests directly to the delegated node by passing the home node as long as the delegation persists. Mukherjee [16] was the first to utilize prediction mechanism in the context of shared memory. Additionally, Lai [17, 23] improved upon this mechanism by reducing the searching overhead. This works support speculative coherence operations. All these works do well from predicting the processor action to a particular cache line, but all require to change the processor die greatly, hence will lead to large hardware overhead, and they are difficult to apply to real commercial CMPs.

Our previous similar work [22] just combined the advantage of write-invalidate and write-update transition strategy; dynamically select the appropriate strategy to maintain cache coherence based on prediction mechanism with coarse-granularity and not provide a accurately policy to direct to choose a proper transition strategy. In this paper, we expand and improve our previous work. We further proposed a low overhead, just two bits for each directory entry, more accurate and effective prediction mechanism and a more comprehensive hybrid cache coherence transition strategy. At last, we give a more detailed implementations of our strategy.

3. Motivation and Background

In this section, we will introduce our motivation at first. Additionally, in order to illustrate mechanism of hybrid shared aware cache coherence transition strategy, we should firstly understand backgrounds of write-invalidate transition strategy and write-update transition strategy.

3.1. Motivation

The utility of cached data block at LLC can be illustrated by cache data block Lifetime [18]. Lifetime is defined as the number of access to a cache line (at the LLC) from cores before it is invalidated or evicted. Figure 2 plots the distribution of Invalidated and Evicted cache line's Lifetime. An invalidated cache line indicates that at least one of accesses to this cache line is a write before reused. We observe that many cache lines residing in the LLC exhibit low Lifetime. For example, In EP over 95% of the cache lines Lifetime is lower than 3. Almost for all tested bechmarks, Lifetime is lower than 5 over 90% averagely. Greater the number of cache lines with high Lifetime in the LLC, greater is the utilization of LLC. Hence, most of our test benchmarks would not benefit from native write-invalidate cache coherence transition strategy. Due to this low utilization of LLC, great amount of remote access occurs. There are two ways to load the needed data for this remote access. One is that getting the available data directly from the main memory, and the other is to get the required data from the remote LLC in another core-group. Figure 3 shows distributions of this two access ways. We observe that about 60% of remote accesses on average get the required data from the remote LLC; especially in SP, almost 90% of required data got form remote LLC. The main reason is that there is great amount of data interaction among threads. In conventional write-invalidate cache coherence transition strategy, when a thread writes a cache line in a SHARED state [19], it must invalidate all copies in another core-group for the exclusive access. Therefore, the cache line in this core-group is not available for the next access. In this time when a thread in this core-group re-access the cache line, there is a remote miss, and the newest cache line will be updated from the remote LLC. However, the remote LLC access is not inevitable. Decreasing remote LLC access is a great opportunity to improve the whole system performance.

Because of this great amount of shared data between threads in scientific applications, the conventional write-invalidate cache coherence transition strategy is not optimized in such case, and leads to low whole system performance. Therefore, it is important to adopt a new method to decrease remote LLC access times. Our goal is to ensure that, if a stable access pattern of interactive read/write a shared data among threads, the write-update transition strategy will replace the conventional write-invalidate transition strategy. Immediately the interactive read/write pattern disappears, we again adopt the conventional write-invalidate strategy.

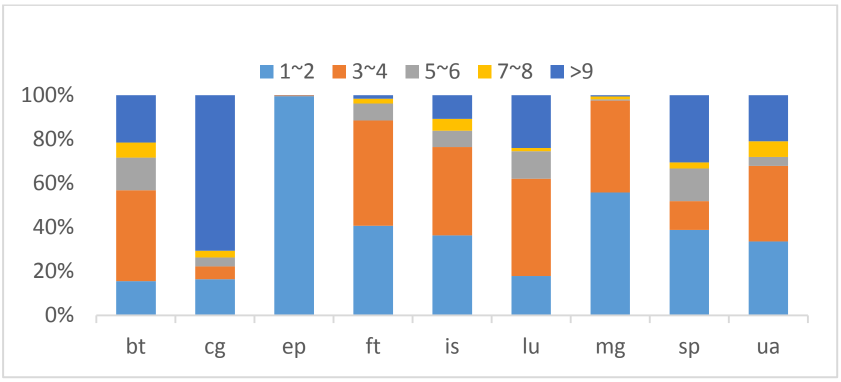


Figure 2. Distribution of Cache line Lifetime; Classification is Done at Cache Line Granularity

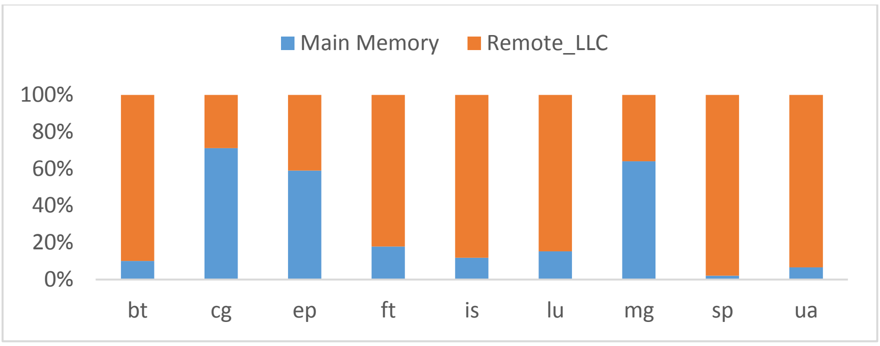


Figure 3. Distribution of Remote Access

3.2. Write-invalidate Transition Strategy Vs Write-update Transition Strategy

Most implementations of CMPs use write-invalidate transition strategy because early studies based on the system bus, and the write-invalidate strategy can effectively reduce traffic and acquire overall better performance. Every time a write operation is executed on shared cache line, write-invalidate cache coherence transition strategy send invalidation messages for exclusive priority. A common situation in multi-threaded applications is that two threads interleaving write or read the same area of memory. In this case, the write thread will successively invalidate the cache lines that will be accessed by the other thread. Therefore, write operations impact more on performance than read operations, as all writes to shared cache lines invalidate the corresponding lines on the other core-group caches [11]. Write-invalidate transition strategy have lower traffic because of the sequence of writes from the same core-group, only the first write needs to send invalidate request to corresponding copies in remote LLC of other core-groups. However, plenty of coherence misses occurred due to the invalidation of remote LLC. Coherency misses occurred when the requested data is continually invalidated due to write-invalidate cache coherence transition strategy. These misses can incur a significant access penalty. At first, the request is forwarded from directory to the remote LLC to keep the exclusive copy. Secondly, the remote LLC with the exclusive copy must flush the corresponding cache line to main memory and send it to the request cache. Thus, the total penalty of this coherence miss includes several cache-to-cache cache line transfers.

In contrast, in a write-update transition strategy, a block loaded into a cache remains until it is replaced, which results in update actions from other cores. Write-update cache coherence transition strategy can eliminate all coherence misses due to all copies of a cache line in remote LLC are updated with a new value instead of invalidation while write a shared block. To guarantee correctness, all writes must keep a stable logical access order. In this write-update strategy, the update phase includes two transactions. In the first transaction, the cache lines are updated but the copies are locked. A core cannot access a locked cache line. During the second transaction, the copies are updated and unlocked, cores are allowed to access their copies again. The prices to pay for the updated new cache line is an increased number of write actions, and more traffic is needed.

As a result, write-invalidate transition strategy has lower write traffic and write penalty at the cost of a higher coherence miss rate whereas write-update strategy eliminates coherence misses at the cost of increased write traffic in the network. Our approach learned the advantages of these two methods and takes write-update transition strategy for those data blocks with high degree of sharing whereas original write-invalidate transition strategy for other data blocks without high degree of sharing. Through our method high coherence miss rate in write-invalidate transition strategy and high write penalty in write-update transition strategy can achieve a good trade-off.

4. Hybrid Shared-aware Cache Coherence Transition Strategy

In this section, we will first describe overview of our cache coherence transition strategy. At last, we give a detail introduction of the implementations of the transition strategy, and how to maintain the cache coherence by using read requests, write requests, eviction and invalidation, respectively.

4.1. Overview of the Transition Strategy

To achieve a good performance of our transition strategy and reduce the mechanism overhead, it is important to accurately predict which cache line will use effectively write-update strategy instead of original write-invalidate strategy while maintaining cache coherence. To limit the number of unnecessary updates, we should only send update to the LLC that most likely to consume the newly written data. In summary, we try our best to guarantee that the updated cache line will be accessed later. In the following, we describe the prediction mechanism that we employ.

In the directory-based MESI cache coherence protocol [9], the directory controller manages all memory access to all distributed LLC, so it has a global history of access of each cache line. To allow finding the usage of cache line, as shown in Figure 4, we extend directory entry with Strategy Counter (2 bits, saturating). This Strategy Counter tracks coherence misses corresponding to the access cache line, which is caused when the requested data is invalidated due to write-invalidate cache coherence transition strategy. Typically a directory cache only contains a small number of entries. e.g, 8K entries on SGI Altix systems[15], which only includes only a fraction of memory cached in the LLC. So for our mechanism, the overhead is limited.

The prediction mechanism logic we adopt is very simple and space overhead is limited and this mechanism can employed in the near-future designs. However, it is very effective to select a better transition strategy considering the whole performance between write-invalidate and write-update strategies. Our prediction mechanism starts out as a conventional write-invalidate transition strategy, and all Strategy Counters in directory cache are initializes to zero. For instance, when a core reads or writes a data block, it searches its private L1 cache at first. On a L1 cache miss, the core starts to look up its local LLC. In searching, if the required data tag matches a cache line tag that is invalidated (valid bits unset) by other cores, a coherence miss occurs and the request is forwarded to the directory. The directory starts to process this request and Strategy Counter (as shown in Figure 4) is incremented by 1. Immediately Strategy Counter is not lower than the Strategy Threshold which initialized by 2 (as shown in Figure 5), the cache coherence protocol is converted from write-invalidate transition strategy to write-update transition strategy. Because there are a lot of interactions among the executing threads for this cache line, write-update strategy is more suitable for reducing the coherence misses and improving whole system performance.

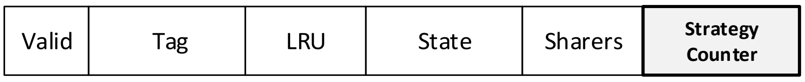


Figure 4. Extended Directory Entry

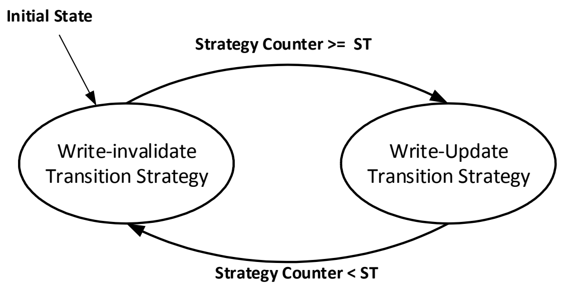

Initial State
Strategy Counter >= ST
Write-invalidate Transition Strategy
Write-Update Transition Strategy
Strategy Counter < ST

Figure 5. Strategy Transformation Diagram

On the eviction of a cache line in LLC, the corresponding Strategy Counter in directory entry is decrement by 1. If the number of the counter is lower than RT, write-invalidate strategy is taken to process this memory access. Through this mechanism, write-invalidate and write-update transition strategy are inter-changed with granularity of cache line. It effectively takes advantage of LLC cache resources and reduces the coherence misses, thus improving the whole system performance.

4.2. Implementations

In the following, we describe the implementation of our mechanism. We extend each directory entry with two bits saturating counter (as shown in Figure 4) to track the access pattern of cache line. We will introduce our implementation mechanism in aspects of read requests, write requests, evictions and invalidation, respectively.

4.2.1 Read Requests: When a core initiates a read request and misses in its private L1 cache, the request is forwarded to its local LLC. The core starts to search the local LLC for matching an appropriate cache line. There are three possible of matching results. (1) A cache line is found, and the cache line is inserted at the private L1 cache directly. (2) It finds a cache line, which is the same tag as the destination data block, but its valid bit is unset, then a coherence miss occurs. This kind of misses occurred due to invalidation of the requesting cache line by another core in write-invalidate cache coherence protocol. In this scenario, Rd_S_I request is sent to the directory. On receiving this transaction, the directory searches the whole directory cache for this cache line. If the cache line is found, the Rd_S request is forward to the owner of this cache line, and Strategy Counter of the cache line is incremented. On the other hand, if the cache line is not found, then the directory sends Rd_S request to memory controller and selects a directory entry with LRU replacement algorithm for the new entering cache line. (3) There is no corresponding cache line in the local LLC, we call a truly cache miss, Rd_S request is sent to the directory, the action of directory is the same as the conventional write-invalidate transition strategy and Strategy Counter in the directory entry is unchanged.

4.2.2. Write Requests: When a core makes a write request for an exclusive copy of a cache line and a miss occur in its private L1 cache line, the request is sent to its local LLC. In this case, if a cache hit occur in local LLC, it denotes the state of the cache line either Exclusive or Modify state, and LLC controller can directly operate on the cache line. This cache line's copy is inserted at the private L1 cache. On the other hand, when a cache miss occurs in its local LLC, the LLC controller performs the appropriate actions according to valid bit, cache line state and the tag bits of corresponding cache line. (1) The valid bit of the cache line is unset, but the tag bits is matched with the required data block. It sends Rd_E_I request to the directory. On receiving Rd_E_I request, the directory searches the directory cache, if the cache line is found, Rd_E request is forwarded to the Owner of this cache line, and Strategy Counter of the cache line is incremented. Otherwise, the directory sends this Rd_E request to main memory and allocates a directory entry with LRU replacement algorithm. (2) Cache line is matched, but the state of cache line is SHARED. In this case, it send UPGRADE request to the directory for the exclusive priority. In response to this request, the directory checks directory cache, and finds the corresponding directory entry for this cache line. If Strategy Counter of the directory entry is larger than the ST, we select write-update transiton strategy to process this cache line access. Then, UPDATE request is sent to the requester, telling the requester that the write-update transition strategy was selected. Then the local and remote LLC will be updated by the newly data. Otherwise, Strategy Counter of the directory entry is lower than the ST, we don't alter the original write-invalidate transition strategy.

4.2.3 Evictions and Invalidations: On an invalidation request, both the bottom LLC and the upper private L1 cache on a core-group are probed and invalidated as the native write-invalidate cache coherence transition strategy. When a LLC cache line is evicted due to eviction, the LLC controller ignores the directory, and decrement Strategy Counter of the corresponding directory entry.

5. Evaluation

5.1. Methodology

This section explains how we evaluated our proposed cache coherence transition strategy. In the next, we will detail introduce our methodology form the two aspects of hardware environment and workload.

5.1.1. Hardware Environments: We simulated our proposed mechanism in the Sniper full system simulator[21]. The base cache coherence protocol we choose is directory-based MESI cache coherence protocol. The system architecture we used in our simulations is presented in Figure 1. The system contains two physical processors, modeled after the Intel Hapertown architecture [20], each consisting of four cores. Each core has private L1 caches for data and instructions, but the L2 cache is shared between two cores. The detail configuration is listed in Table I

Table 1. Simulation Platforms Configuration and Performance Parameters

Parameters	Value
Configuration (chips * cores)	2 * 4
LLC size	4MB * 4
Local LLC access (cycles)	38 ~ 42
Remote cache access (cycles)	150 ~ 250
Memory access (cycles)	300 ~ 350
Cache coherence protocol	Directory-based MESI

5.1.2. Workloads: In the rest of the paper, we used the OpenMP implementation of the NAS parallel benchmarks (NPB)[10] to evaluate our proposal. The benchmarks were executed using the W input size, since it is the most appropriate size for simulation. We ran all the benchmarks except DC, which takes too much time to simulate.

5.2. Results Analysis

In this section, we present the results we obtained by using our hybrid shared-aware cache coherence strategy. Figure 6 shows the speedup of the tested parallel applications opposed to the traditional write-invalidate cache coherence transition strategy. Almost for all tested parallel applications, the performance has great improvement. The average speedup of all applications is up to 21%, and the IS applications is up to 47%. The application of MG is an exception, and there is a low performance due to low sharing of accessing data block. To get a more detailed understanding of the benefits of our mechanism, we also illustrate the number of LLC cache line invalidation, LLC cache misses, Cache-to-cache transfer. Cache-to-cache transfer occurs when a core access a data which is present neither in its private L1 cache nor local LLC and has to acquire the data from anther LLC. Figure 7, 8, 9 denote the normalized numbers of LLC cache line invalidation, LLC cache misses and cache-to-cache transfers, respectively.

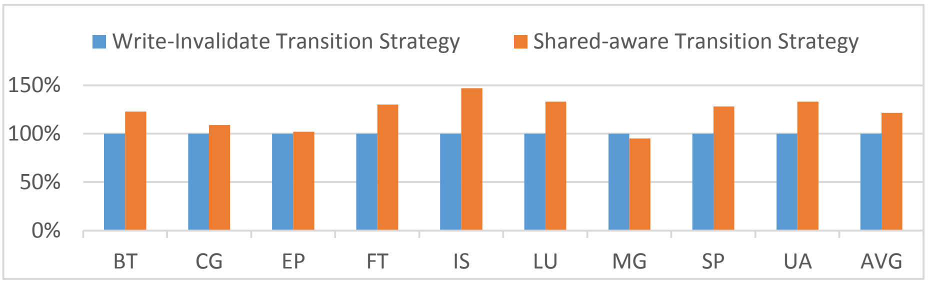


Figure 6. Normalized Applications Speedup

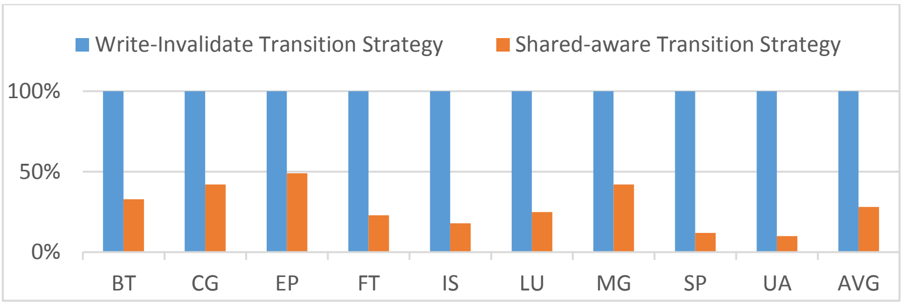


Figure 7. Normalized LLC Cache Line Invalidation

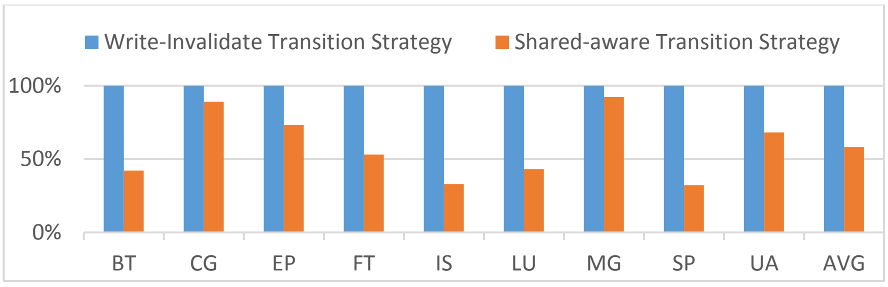


Figure 8. Normalized Cache-to-cache Transfers

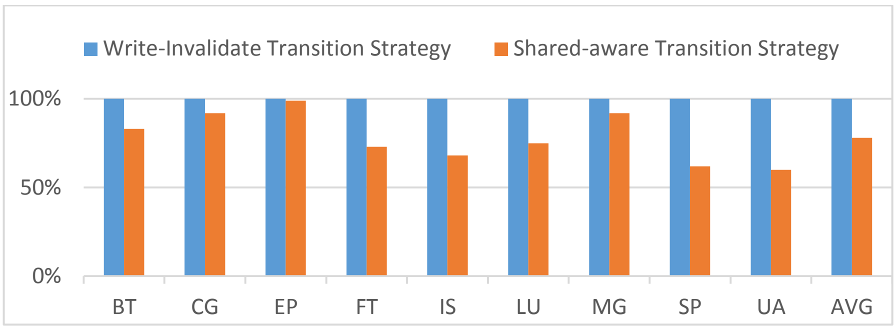


Figure 9. Normalized LLC Cache Misses

LLC capacity of a machine is very important for a programs performance. From Figure 7, we can investigate that the LLC cache line invalidation of all tested applications is dramatically decreased in our mechanism, about 72% in average because of the goal of our mechanism is to decrease the number of invalidation of cache line through selective using write-update strategy. On the other hand, Figure 7 can demonstrate that our mechanism achieves the goal of reducing the invalidation, and our mechanism works well too. The invalidation of IS is reduced about 80%, because IS is a bucket-based large integer sort, it require a lot of communication. LU exhibits good performance improvements (33% speedup, 75% LLC cache line invalidation).Because of the using data are assigned to individual processors, and directly result in amount of shared data that can be accesses by many cores. However, for the EP, there is a limited reducing, because EP is an embarrassingly parallel benchmark, which generates pairs of Gaussian random deviates according to a specific scheme and almost does not require communication between the processor in computing.

Figure 8 and Figure 9 show that the LLC cache misses and cache-to-cache transfer are improved about 22% and 40% in average. The reason for this reduction is that the reduction of invalidation of shared cache line lead to less coherence misses, and converts this coherence misses to cache hit almost. For almost tested applications, the cache-to-cache transfer are reduced, all required data is directly acquired form the home core-group. The utility of hardware cache resource are improved dramatically. In summary, our performance improvement largely depends on the thread writing of shared data, but almost tested applications satisfy for our proposal cache coherence transition strategy.

In the next, we select some representative applications to give more detail analysis. LU solves a finite difference discretization of the 3D compressible Navier-Stokes equations through a block-lower block-upper approximate factorization of the original difference scheme. The vertical columns of data are assigned to individual processors, and directly result in amount of shared data that can be accesses by many cores. Therefore, LU exhibits good performance improvements (33% speedup, 75% LLC cache line invalidation, 25% LLC cache misses reduction). CG uses a Conjugate Gradient method to compute an approximation to the smallest eigenvalue of a large, sparse, unstructured matrix. CG exhibits a limited communication due to the data it used is the sparse and unstructured data matrix. Although the cache line invalidation is reduced about 60%, there is a little improvement in speedup (5%), LLC cache misses (8%). In addition, the sparse matrix used in CG leads to plenty of false sharing which confine performance improvement.

6. Conclusions

Our work focuses on the design mechanism that improves the performance of multi-threaded applications by eliminating coherence misses and coherence traffic. In this paper, we propose a scalable, efficient hybrid shared-aware cache coherence transition strategy that collaborates with directory-based MESI cache coherence protocol that can be used to improve the performance of parallel applications that exhibit large amount of data sharing between threads. Through detecting instance of coherence misses of a cache line using a simple directory-based predictor, we can demonstrate that these shared cache line are accessed frequently. In this scenario, we alter the original write-invalidate cache coherence transition strategy to write-update cache coherence transition strategy, which can update the cache line in real time instead of invalidation, and covert next coherence misses to cache hit. We evaluated our proposal mechanism using OpenMP implementation parallel applications from the NPB. We demonstrate that our proposal hybrid shared aware cache coherence transition strategy mechanism can reduce the number of coherence misses compared to conventional write-invalidate cache coherence transition strategy. We evaluate our hybrid cache coherence transition strategy by the execution time, number of cache line invalidation, cache misses and cache-to-cache transfer, the results show the performance gains of up to 21% averagely opposed to the native cache coherence transition strategy, cache misses and cache-to-cache transfers were reduced by up to 25% and 30%. This shows that our hybrid transition strategy is a better method for managing precious cache resource.

Acknowledgments

This paper is a revised and expanded version of a paper entitled Cache Coherence Method for Improving Multi-threaded Applications on Multicore Systems presented at MulGrab, Hainan, China, December 20-23, 2014. This work is supported financially by the National Basic Research Program of China under contract 2011CB302501, the National Natural Science Foundation of China grants 60970023, the National Hi-tech Research and Development Program of China under contracts 2012AA010902 and 2012AA010901, the National Science & Technology Major Projects 2009ZX01036-001-002 and 2011ZX01028-001-002-3.

References

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious", ACM SIGARCH computer architecture news, vol. 23, no. 20, **(1995).**

[2] X. Yang, S. M. Blackburn, D. Frampton, J. B. Sartor, and K. S. McKinley, "Why nothing matters: the impact of zeroing", Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications, **(2011)** October 22-27, Portland, OR, United State.

[3] X. Ding, K. Wang, and X. Zhang, "ULCC: a user-level facility for optimizing shared cache performance on multicores", Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, **(2011)** February 12-16; San Antonio, TX, United States.

[4] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems", Proceedings of the 14th International Symposium on High Performance Computer Architecture, **(2008)** February 16-20; Salt Lake City, UT, United States.

[5] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page coloring-based multicore cache management", Proceedings of the 4th ACM European conference on Computer systems. **(2009)** May 5-9; New York, NY, USA.

[6] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling", Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, **(2010)** March 13-17; Pittsburgh, PA, United States.

[7] K. K. Pusukuri, D. Vengerov, A. Fedorova, and V. Kalogeraki, "FACT: a framework for adaptive contention-aware thread migrations", Proceedings of the 8th ACM International Conference on Computing Frontiers, **(2011)** May 3-5; Ischia, Italy.

[8] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "AKULA: a toolset for experimenting and developing thread placement algorithms on multicore systems", Proceedings of the 19th international conference on Parallel architectures and compilation techniques, **(2010)** September 11-15; Vienna, Austria.

[9] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the DASH multiprocessor", Proceedings of the 17th Annual International Symposium on Computer Architecture. **(1990)** May 28-31; Seattle, WA, USA.

[10] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance", Technical Report NAS-99-011, NASA Ames Research Center **(1999)**.

[11] E. H. Molina da Cruz, Z. Alves, A. Carissimi, P. O. A. Navaux, C. P. Ribeiro, and J. Mehaut, "Using memory access traces to map threads and data on hierarchical multi-core platforms", Proceeding of 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, **(2011)** May 16-20; Anchorage, AK, United States.

[12] R. Azimi, D. K. Tam, L. Soares, and M. Stumm, "Enhancing operating system support for multicore processors by using hardware performance monitoring", ACM SIGOPS Operating Systems Review, vol. 43, no. 56, **(2009).**

[13] K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance evaluation of memory consistency models for shared-memory multiprocessors", Proceeding of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, **(1991)** April 8-11; Santa Clara, CA, USA.

[14] G. Kurian, O. Khan, and S. Devadas, "The locality-aware adaptive cache coherence protocol", Proceedings of the 40th Annual International Symposium on Computer Architecture, **(2013)** June 23-27; Tel-Aviv, Israel.

[15] L. Cheng, J. B. Carter, and D. Dai, "An adaptive cache coherence protocol optimized for producer-consumer sharing", Proceeding of the 13th International Symposium on High Performance Computer Architecture. **(2007)** February 10-14, Scottsdale, AZ, United States.

[16] S. S. Mukherjee and M. D. Hill, "Using prediction to accelerate coherence protocols", ACM SIGARCH Computer Architecture News, vol. 26, no. 179, **(1998).**

[17] A. C. Lai and B. Falsafi, "Memory sharing predictor: The key to a speculative coherent DSM", Proceedings of the 26th International Symposium on Computer Architecture, **(1999)** May 2-4; Atlanta, USA.

[18] A. Krishna, A. Samih, and D. Solihin, "Data sharing in multi-threaded applications and its impact on chip design", Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems and Software, **(2012)** April 1-3; New Brunswick, NJ, United States.

[19] M. S. Papamarcos and J. H. Patel, " A low-overhead coherence solution for multiprocessors with private cache memories", ACM SIGARCH Computer Architecture News, vol. 12, no. 348, **(1984).**

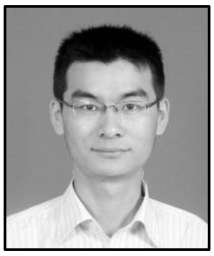
[20] Q.-C. I. Intel, Xeon® Processor 5400 Series. Intel Corporation, California **(2008)**

[21] W. Heirman, T. Carlson, and L. Eeckhout, "Sniper: scalable and accurate parallel multi-core simulation", Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, **(2012)** November 12-18; Seattle WA, United States.

[22] S. Sun, H. An, and J. Chen, "Cache Coherence Method for Improving Multi-threaded Applications on Multicore Systems", Proceedings of the 6th International Conference on Multimedia, Computer Graphics and Broadcasting, **(2014)** December 20-23; Hainan, China.

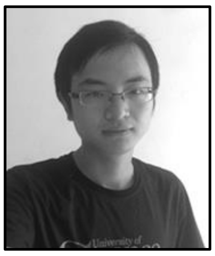
[23] J. Liu, J. Chen, W. Tong, C. Chen, "Design and Implementation of Metadata Cache Management Strategy for the Distributed File System", International Journal of Grid and Distributed Computing, vol. 7, no. 89, **(2014).**

Authors



Sun Sun, he received B.S in Computer Science from Anhui University. Currently he is pursuing his doctoral degree in Computer Architecture form University of Science and Technology of China. His major research interest is chip multiprocessors(CMPs) with an emphasis on the last-level cache(LLC) designs, Shared resource management for CMPs, Performance characterization, analysis and prediction for CMPs

Hong An, Professor, she is working at University of Science and Technology of China and also a member of HPC Advisory committee, China Computer Federation Architecture special committee. Her major research interest is architecture of chip multiprocessors(CMPs), Parallel computer architecture, High-performance computing application, Parallel programming environments and tools.



Junshi Chen, he received B.S in Computer Science from Anhui University of Technology. Now he is pursuing doctoral degree in Computer Architecture from University of Science and Technology of China. His major research interest is Performance characterization, analysis and prediction for chip multiprocessors (CMPs).