Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 1, January 2018, pp. 178-185
DOI: 10.23940/ijpe.18.01.p19.178185

A Cache Consistency Protocol with Improved Architecture

Qiao Tian*, Jingmei Li, Shuo Zhao, Fangyuan Zheng, Jiaxiang Wang

College of computer science and technology, Harbin Engineering University, Harbin, 150001, China

Abstract

With the technical innovation of microprocessors, the application of cache memory prevents the main memory from limiting the development of microprocessors. However, the introduction of Cache also brings the cache coherence problem while the microprocessor performance is being improved. Based on this problem, this paper designs an improved CMP architecture model and proposes a new Cache Coherence Protocol Based on Virtual Bus (CCPBVB). Firstly, the D-Cache virtual bus is added to the architecture to realize the point-to-point consistency transaction transmission, avoiding the bus idle phenomenon caused by the polling query method that the broadcast consistency transaction must follow, while also improving the effective utilization rate of the bus. Then, in order to reduce the access delay caused by the ping-pong phenomenon in the cache memory, a protocol is designed to improve the C hit rate, which combines the write invalid strategy and write update strategy between the private cache to reduce the delay time of the system. The experimental results show that the proposed protocol not only improves the bus utilization, but also reduces the C access delay.

Keywords: cache consistency problem; CMP architecture; virtual bus; access delay

(Submitted on October 22, 2017; Revised on November 19, 2017; Accepted on December 21, 2017)

1. Introduction

Cache consistency is one of the hot issues in the field of processor research. With the continuous optimization of processor architecture, the issue has become more serious, and it is a technical problem that determines whether multi-core technology can be further developed [5,10,11]. Therefore, the design of an effective cache consistency protocol to improve processor performance is of great significance.

Today, research on cache consistency has gradually matured, and it is constantly optimized and improved in its development. From the operating system level, Cuesta proposed the private data deactivate coherence, which distinguishes shared data and private data [1], thus avoiding the increment of directory entries by preventing the registration of private data in the directory. Valls proposed a Private Shared Directory (PS-Dir) mechanism [15]. PS-Dir has two completely separate caches. Because the two cache blocks are different, the two cache directories can convert to each other when PS-Dir works. From traffic, performance and power consumption, Menezo L G, Puente V and Gregorio J A proposed a coherence protocol called Flash with a listening mechanism and mixed directory characteristics [8]. The protocol effectively avoids the waste of communication traffic, reduces power consumption and improves system performance.

The traditional cache consistency protocol is divided into two categories: the consistency protocol based on bus listening and the consistency protocol based on directory [13]. As the focus of the protocols are different, the two have their own advantages and disadvantages. Based on the deep research and analysis of the bus listening protocol and directory protocol, this paper realized the point-to-point consistent transaction trans-mission by adding the D-Cache virtual bus in the architecture, and designed a new cache consistency protocol — the CCPBVB snoopy protocol, which improved the effective utilization of the bus and reduced the system access delay.

* Corresponding author.
E-mail address: tianqiao@hrbeu.edu.cn

This paper includes four sections. The first section explains the purpose of the research. In section two, we analyze the protocols based on bus listening and directory respectively, and then perform some optimizations on them. The third section introduces the designed cache consistency protocol CCPBVB. The last section contains the experiments done in order to verify the performance of CCPBVB.

2. Cache Consistency Protocol Optimization

2.1. The analysis and optimization of bus listening protocol

The bus listening protocol uses a bus to connect the private cache of the processor core with the main memory, propagating consistent transaction messages on the bus in broadcast. So, the bus is the ordering point of all consistent messages, and all nodes connected to the bus can observe the messages in the same order. The representative listening protocols are the Modified-Shared-Invalid (MSI) protocol [2] and Modified-Exclusive-Shared-Invalid (MESI) protocol [7,3].

The bus in the bus listening protocol is an exclusive resource that can easily decrease protocol efficiency, and the polling query method where the broadcast consistent transaction must be observed also produces bus idle occupancy [14]. Based on the above shortcomings, the D-Cache virtual bus architecture model is added to avoid invalid resource usage of the broadcast sending message mechanism and improve the equipment utilization based on the bus listening protocol. The D-Cache virtual bus structure is shown in Figure 1.

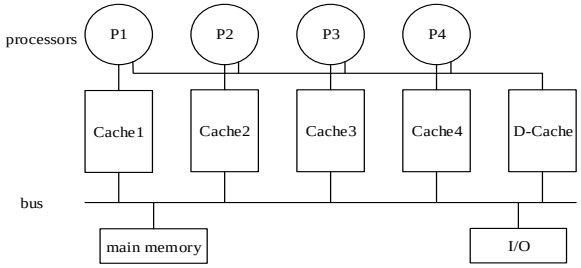

Figure 1. The system structure with D-Cache virtual bus

The D-Cache virtual bus is used to store directory entries that record data information. By designing and modifying the directory entry, the structure also constructs three units: request transaction collection unit, directory entry lookup and update unit, and listening response transaction unit. Therefore, in order to complete the protocol function with the above three units, the improved directory entry structure is shown in Figure 2.

Valid_Bit	Status_Bit	Ident_Bit	Share_Bit	Busy_Bit	Count_Bit

Figure 2. The directory entry structure

In Figure 1, raising the location of D-Cache to the private cache of each processor core not only speeds up the search for target data block, but also reduces the access delay. In Figure 2, the read and write requests from processor cores are first cached in the request transaction collection unit, and then they are implemented in chronological order by the directory entry lookup and update unit. The unit matches the directory entry in D-Cache with every request, which is identified by Ident_Bit: 1 for hit, 0 for miss, and after the hit, it checks whether Valid_Bit is 1. 1 represents that the entry is valid, otherwise it is invalid; then it checks Busy_Bit, as 1 represents that the data block is being used. The data block can only be read and written until Busy_Bit is 0; when Busy_Bit is 0, the Status_Bit and Share_Bit can get the state of target data and identify which processor core contains the data in their private cache. When the corresponding read and write requests are met, the entry information will be updated and the Count_Bit will be incremented by one, which is used as a reference bit when the data block is replaced. The smaller number is preferentially replaced.

2.2. Directory Consistency Protocol

The directory protocol uses the directory to store information about the cache data copy, and the directory serves as the ordering point. The requested data is obtained in point-to-point communication after finding the directory [4,9]. All consistent messages are forwarded through a directory structure. Therefore, the directory protocol reduces the network bandwidth requirements of consistent messages and avoids the blindness caused by broadcast messages. The directory protocol is represented by a fully associative directory, limited directory and chained directory.

Based on the advantages and disadvantages analysis of a fully associative directory, limited directory and chained directory, by combining the fully associative directory and chained directory, a new cache consistency protocol of a two-level directory structure is proposed. The system architecture is shown in Figure 3.

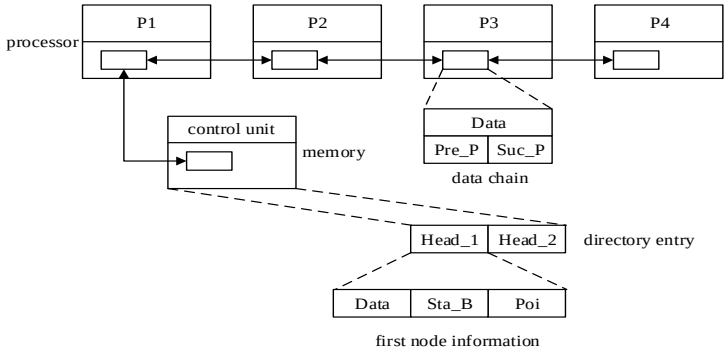

Figure 3. The two-level directory structure

Each directory entry in the main memory consists of two head nodes: Head_1 and Head_2, and each head node consists of three parts: Data, Sta_B and Poi. The head node points to the first address of the shared data block. The system adds a data chain to each data block in the private cache, which contains Pre_P and Suc_P.

When a processor core sends a read request, the request information first reaches the main memory directory. After matching the data block, the head node sends the data to the processor core that requests data, and the private cache of the processor core is added to the chain of the head node.

When a processor core sends a written request, the information also reaches the main memory directory. If Sta_B of the head node is the state except "M", all the data blocks connected to the node should be discarded. Then a write operation is done as well as a modification of the Sta_B. Finally, the private cache with the latest data is connected to the head node. If Sta_B of the head node is "M", it can be written directly after transferring the data. Tt is not necessary to modify the Sta_B after the completion of the write invalidate. The private cache of the processor core is connected to the chain that the head node is in.

The MSI protocol has three states, which are not enough to accurately describe the whole data block state, easily increasing the number of consistency transactions in the system. Therefore, it is not ideal to improve the protocol performance by unilaterally optimizing the architecture model [6]. Moreover, although the MESI protocol adds an "E" on the basis of the MSI protocol, it effectively reduces the number of conformance transactions. However, the MESI protocol does not use the cache ping-pong problem as the design direction, and so the system delay has not been solved. Finally, the optimized directory protocol is simple and has good scalability, but because the head node in the main memory causes each read and write request to be indirectly accessed, the operation time increases, thereby reducing the efficiency of the entire system.

To summarize, the optimization of cache consistency protocol should not be only from the protocol itself, but we also need to consider the importance of the architecture model, so as to effectively maintain the consistency of the data in the system.

3. Design of CCPBVB protocol

Based on the protocol optimization scheme summarized above, this chapter applies D-Cache to the improved architecture model in order to achieve point-to-point consistent transaction transmission and avoid the blindness caused by the broadcast message. Then, a CCPBVB (Cache Consistency Protocol Based Virtual Bus) based on write-back strategy is designed, and the state set and state transition graph of CCPBVB protocol are described.

3.1. Improved system structure model

In this section, the D-Cache virtual bus mechanism is added into the system structure based on the second part of the paper. The additional D-Cache component has the following advantages.

On one hand, its specifications are the same as other private caches, maintaining the correct execution of the entire system as the ordering point of consistent transaction. The frequently used data is stored in D-Cache in the form of directory entry, which not only shortens the access delay time, but also speeds up locating the target data block.

On the other hand, the improved system structure model draws on the idea of the bus listening protocol and directory protocol. By combining the advantages of two traditional protocols, we can aim to reduce the system delay time and improve the effective utilization rate of the bus. The mechanism of the processor core mastering the target data block information is optimized, and the point-to-point sending function is realized in the consistent transaction transmission. The controller in the private cache does not always need to be in the listening state, as the shared bus can also get relief in heavy affairs, releasing the system pressure to a certain extent and improving the system performance.

3.2. Directory organization structure and D-Cache operation steps

The architecture with D-Cache constructs three structural units by designing and modifying the directory entry in the directory protocol, generating a D-Cache virtual bus with physical proximity distance. With the virtual bus, the point-to-point consistent transaction processing mode is implemented when the local read and write requests miss. The specific steps are as follows:

Firstly, the read and write requests of the processor core are sent to the local cache and D-Cache; if the cache hits, do the corresponding read and write operation while interrupting the D-Cache read and write requests. When the read and write operations of the processor core are finished, the corresponding directory entry does an update operation by the directory entry lookup and update unit.

Secondly, if the local private cache misses, the read and write request messages sent to D-Cache are cached to the request transaction collection unit in chronological order. If the unit is full, the request message is not buffered and the listening response transaction unit does not send a reply transaction message, indicating that the request message has been correctly received. Therefore, the processor core requesting the data cannot get the target data. If the unit is sufficient, the processor core read and write request message is cached in this unit. The request transaction is selected through the arbitration mechanism of the request transaction collection unit. Then the target data block is found through the directory entry lookup and update unit. If successful, a target data request message is formed from the listening response transaction unit and it is sent to the processor core with a target data block, while the corresponding directory entry is updated; otherwise, a corresponding miss message is formed to the processor core with request data by the listening response transaction unit, and then the next level memory is needed to find the target data.

Finally, if the data is offered from the main memory, the entire data block containing the target data is first transferred into the corresponding private cache. The read and write operations of the processor core are then completed, and the update operation is done by the directory lookup and update entry unit.

3.3. The state set of CCPBVB

The CCPBVB protocol proposed in the paper adopts the write-back strategy, as well as the advantage combination of the write-invalid and write-update strategies. The protocol contains five states, as shown in Table 1.

Table 1. Consistent transaction request

transaction	description
LR	local read request
LW	local write request
RR	remote read request
RW	remote write request
RP	replacement request of cache block

3.4. State transition process

In CCPBVB, the operation for the data block has local read and write request, remote read and write request, and a replacement request of the cache block. The remote read and write request is from the generation and forwarding of the listening response transaction in the D-Cache. The consistent transaction message is sent to the processor core described in Share_Bit in a straightforward way through Ident_Bit in the directory entry. The process optimizes the forwarding and

transport mechanism of the consistent transaction and implements the stable, accurate features of message transmission, improving the protocol efficiency. The specific consistent transaction request is shown in Table 2.

Table 2. States of CCPBVB protocol

State	Description
LM	Latest modified state, the current data block copy has been modified, only the private cache contains the latest data, the block needs write back to the main memory if it is replaced.
ME	Mutex exclusive state, cache in this state is "unique" which has the same data with main memory, the data in other caches is invalid.
ES	Effective shared state, the cache in this state has the same data with the main memory and other processor cores, the number of processor cores with the data is two or more.
RS	Rewritten shared state, the data block copy in this state exists only in two processor cores of the system. In this case, the data in cache with this state does not match the main memory, the read and write operations are made after writing back to the main memory.
DI	Data invalid state, the cache in this state is invalid, so the read and write operations of cache miss, it should read the required data block from other cache containing the valid data or the main memory.

Figure 4 shows the state transition graph when the data block copy is only one in multi-core processor system. The details are as follows:

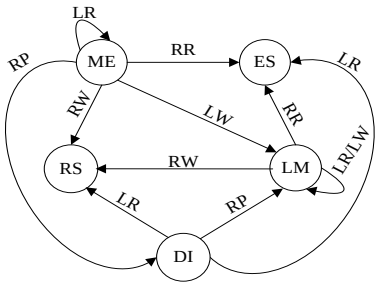

Figure 4. The state transition graph when the data block is only one

(1) Local read operation

When there is only one data block copy in the system, the data block may only be in ME or LM. The copy state does not change after the local read operation.

(2) Local write operation

- When the data block existing in the system is ME, the cache controller needs to modify the data block content when it receives the write request from local processor core. The data block state is transformed from ME to LM.
- If the data block existing in the system is LM, the cache controller also changes the data block content when it receives a write request from the local processor core, but the modified data is still the only valid data in the system. That is, the copy state does not change.

(3) Remote read operation

- The remote read operation hits the data block copy in ME. Its state is changed from ME to ES, and the requestor's copy is changed from DI to ES.
- The remote read operation hits the data block copy in LM. Its state is changed from LM to ES, and the requestor's copy is changed from DI to ES. Since the data block in the latest modified state is the only valid copy of the system, when the remote processor hits, the data block is written back to the main memory and the data block state is changed.

(4) Remote write operation

- When the data block state is ME, a receiving remote write operation makes the state RS, and the requestor's copy state is changed from DI to RS.
- When the data block state is LM, the remote write operation hits. The copy state becomes RS, and the requestor's copy state becomes RS.

(5) Replacement operation of cache block

- The data block copy in ME is not written back to the main memory when the replacement operation is performed. The data block state finally is DI.
- The data block copy in LM needs to be written back to the main memory when the replacement operation is executed, and the data block state eventually becomes DI.

Figure 5 shows the state transition graph of when the number of shared copies in the system is two or more. The details are as follows:

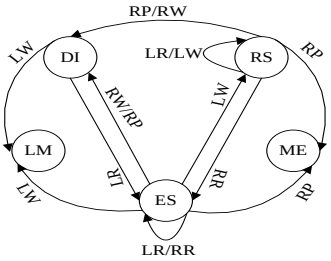

Figure 5. The state transition graph when the number of shared copy is two or more

(1) Local read operation

When the number of shared data blocks is two or more, the data block may be in RS or ES, and the local read operation does not change the current block state.

(2) Local write operation

- When the data block in RS is hit, the write update strategy is adopted because there are only two copies of data blocks in the system, so the two data blocks state is kept as RS and does not change.
- When the data block in ES is hit, the state transition depends on the number of data block copies in the system. When the number is two, the data block state is transferred to RS because the system takes the write update strategy. When the number is more than two, the data block state is transferred to LM because the system takes the write invalid strategy, and the rest caches containing the data block copy eventually become DI.

(3) Remote read operation

- If the remote read operation hits the data block copy in RS, the data block is first written back to the main memory. Then this data block information is transferred from the source processor node to the destination processor node through a shared bus by the D-Cache locating the state of the original cache containing the data block. The requestor's cache is changed to ES.
- If the data block copy in ES is hit, the number of data block copies determines the transition state. When the number is two, the original data block state remains unchanged since the read operation does not change the data, and the requestor’s cache state is also changed to ES. When the number is more than two, the data information also does not change, and the statuses of the original cache and requestor's cache are ES.

(4) Remote write operation

- When the remote write operation hits the data block copy in RS, it exceeds the write operation range between the two copies, so we get the write obsolete, and the cache state containing the data block copy finally becomes DI while the data requestor’s cache state becomes LM.
- When the remote write hits the data block copy in ES, the system takes the write invalid strategy regardless of whether the number of data block copies in the system is two or more. That is, the cache containing the data block copy becomes DI, and the requestor’s cache state becomes LM.

(5) Replacement operation of cache block

- When the data block copy in RS does a replacement operation, the data in the main memory is obsolete because the cache in RS contains the latest data. So, the first the copy is written to the main memory, and then the cache state becomes DI while another copy becomes ME.
- When the data block copy in ES does a replacement operation, the number of data blocks in the system is two, and the replacement of the block is not written back to the main memory. The other cache state is ME. When the number of data block is more than two, the local cache state becomes DI, and the other caches containing this data block keep ES unchanged.

4. Experimental verification

In order to test the performance of CCPBVB, the CCPSVB protocol is compared to the MESI protocol by selecting the GEMS system multi-core simulator platform [12]. The structure of the GEMS simulator is shown in Figure 6. The GEMS simulator consists of five parts: Random Tester module, Microbenchmarks module, Simics module, Opal module and Ruby module [16].

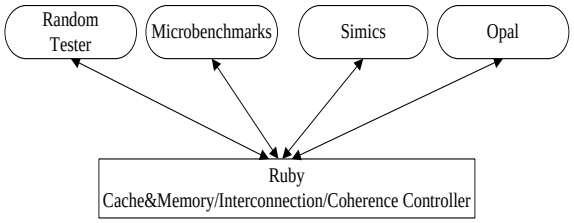

Figure 6. GEMS simulator structure

The parameters selected in the experiment are shown in Table 3.

Table 3. Experimentalenvironmental parameters

Simulation parameters	Parameter value
number of processor core	4
L1 instruction cache capacity	32KB
L1 instruction cache access delay	1 Cycle
L1 data cache capacity	32KB
L1 data cache access delay	1 Cycle
size of the Cache block	32 byte
shared bus bandwidth	32 byte
main memory capacity	4G

In the selection of test procedures, the paper uses the SPLASH-2 centralized test program LU, Ocean, Radix, FFT and Water-SP to test the performance of CCPBVB and MESI, as shown in Table 4.

Table 4. Test procedures

Name of test procedure	Characteristic parameters
LU	512*512 matrix
Ocean	258*258 ocean
Radix	1M keys,1024 radix
FFT	256K points
Water	512 molecules

As shown in Figure 7, the test procedure is run in the MESI and CCPBVB environments respectively. Based on the running time of the five test procedures in the MESI protocol environment, we compared them to the time in the CCPBVB protocol environment, and the unit of running time is CPU cycle. It can be concluded that the average running time of the test procedures in the CCPBVB environment is 3.84% less than that in MESI, and the test results in the CCPBVB environment are better than those in MESI. As a result, the CCPBVB protocol improves the efficient utilization of the bus and system performance to a certain extent.

Then, the failure rates of the two protocols of L1 cache with the same test procedure are compared. As shown in Figure 8, the MESI protocol L1 Cache failure rate is compared with that in CCPBVB. The average failure rate of CCPBVB is 2.76% lower than that of MESI.

In the comparison of the test procedures running time, based on the test procedure running time in MESI environment, the time in CCPBVB is shorter, indicating that the CCPBVB protocol improves the system efficiency to a certain extent and the effectively utilizes the rate of bus. In the L1 Cache failure rate, it can be seen from Figure 8 that the CCPBVB protocol shows a better performance, indicating that CCPBVB solves the access delay caused by the ping-pong phenomenon to a certain extent, achieving the goal of the protocol design.

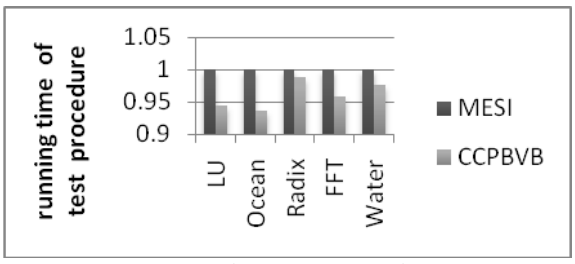

Figure 7. Comparison of the running time of test procedures

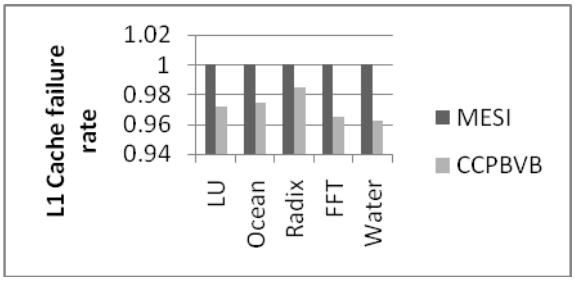

Figure 8. The comparison of Cache failure rate

5. Conclusions

The cache consistency problem has become one of the hot issues in the field of multi-core processor research. This paper summarizes the current problems of cache consistency protocols, the effective use of shared bus resources that the bus listening consistency protocol lacks, and its broadcast consistent transaction mechanism that leads to inadequate use of resources. The directory-based consistency protocol has a long access delay. Aimed to address the shortcomings of these two consistency protocols, the D-Cache virtual bus architecture model of the paper effectively solves these problems of bus effective utilization, and the design of the CCPBVB protocol is a better way to alleviate the access delay caused by ping-pong, improving the system's performance. But, limited by the author's ability and time factors, this paper has some shortcomings, which will be further studied and resolved in following scientific research work.

Acknowledgements

This work is supported by Research on Compiling Technology Based on FPGA Reconfigurable Hybrid System (No.61003036). The authors would like to thank all of the co-authors of this work.

References

1. B. Cuesta, A. Ros, M. E. Gómez and et al, "Increasing the Effectiveness of Directory Caches by Avoiding the Tracking of Noncoherent Memory Blocks," IEEE Transactions on Computers, vol. 62, no. 3, pp. 482-495, March 2013
2. S. K. Chen, "Study and Implementation of Cache Consistency Protocol in Multi-core Processor," National Defense University of Science and Technology, pp. 25-40, 2005
3. M. Dalui and B. K. Sikdar, "An efficient test design for CMPs cache coherence realizing MESI protocol," International Conference on Devices, Circuits and Systems, IEEE, pp. 718-722, 2012
4. S. L. Guo, H.X. Wang, Y. B. Xue and et al, "Hierarchical Cache Directory for CMP," Journal of Computer Science and Technology, vol. 25, no. 2, pp. 246-256, May 2010
5. A. Hsia, C. W. Chen and T. J. Liu, "Energy-efficient synonym data detection and consistency for virtual cache," Microprocessors & Microsystems, vol. 40, no. C, pp. 27-44, February 2016
6. G. M. Li, "Research on Cache Consistency Model in On-chip Multiprocessor Architecture," University of Science and Technology of China, pp. 57-65, 2013
7. F. J. J. Maturana, C. P. Gómez, E. H. Gómez and et al, "Teaching the cache memory coherence with the MESI protocol simulator," I.e.s.emilio Canalejo Olmeda Universidad De Córdoba, vol. 98, no. 541, pp. 131-132, January 2006
8. L. G. Menezo, V. Puente and J. A. Gregorio, "Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability," IEEE International Symposium on High Performance Computer Architecture, pp. 198-209, February 2015 (DOI 10.1109/HPCA.2015.7056033)
9. A. Ros and M. E. Acacio, "DASC-DIR: a low-overhead coherence directory for many-core processors," Journal of Supercomputing, vol. 71, no. 3, pp. 781-807, March 2015
10. A. Ros and A. Jimborean, "A Dual-Consistency Cache Coherence Protocol," Parallel and Distributed Processing Symposium, IEEE, pp. 1119-1128, July 2015 (DOI 10.1109/IPDPS.2015.43)
11. A. Ros and A. Jimborean, "A Hybrid Static-Dynamic Classification for Dual-Consistency Cache Coherence," IEEE Transactions on Parallel & Distributed Systems, vol. 27, no. 11, pp. 3101-3115, February 2016
12. J. W. Shu, Y. Y. Lu, J. C. Zhang and et al, "Research progress on non-volatile memory based storage system," Science &Technology Review, vol. 34, no. 14, pp. 86-94, June 2016
13. L. S. Selvin and Y. Palanichamy, "Push-pull cache consistency mechanism for cooper caching in mobile ad hoc environments," vol. 24, no. 5, pp. 3459-3470, January 2016
14. J. J. Valls, A. Ros, M. E. Gómez and et al, "The Tag Filter Architecture: An energy-efficient cache and directory design," Journal of Parallel & Distributed Computing, vol. 100, pp. 193-202, February 2017
15. J. J. Valls, A. Ros, J. Sahuquillo and et al, "PS-Dir: a scalable two-level directory cache," International Conference on Parallel Architectures and Compilation Techniques, pp. 451-452, September 2012 (DOI 10.1145/2370816.2370891)
16. Z. Zhang, "Research and implementation Multi-core cache consistency protocol," Xi'dian University, pp. 8-11, 2013