Design and Implementation of a Chip Multiprocessor with an Efficient Multilevel Cache System

Amin El-Kustaban[(1,*)], Abdullah Qahtan[(1,*)]

Abstract

Computer designers utilize the recent huge advances in Very Large Scale Integration (VLSI) to get Chip Multiprocessor (CMP) by placing several processors on the same chip die. The CMP is the dominant architecture to improve the performance of the current computing systems. However, accessing a shared data by several processors is a primary challenge in CMP. The data consistency must be reached among all memory hierarchies to ensure correct behavior and higher performance. This paper, proposed a CMP with an efficient multilevel cache system, which enhances miss rate and latency (penalty) by designing and implementation of different write policies with two levels of cache. The proposed system is implemented and tested using Hardware Description Language (VHDL) on Altera's FPGA chip. The results show that a combination of write-through without buffer for the first level and write-back for the second level offers a clear improvement on the multilevel cache system performance.

Keywords: private cache system, write policies, miss penalty, VHDL, FPGA.

1. Introduction

The technology revolution in Very Large Scale Integration (VLSI) has enabled today's designers to design and implement Chip Multiprocessor (CMP), where two or more processors with a shared memory are integrated on a single chip [1]. In the next few generations, the number of processors that can be implemented on a single chip will significantly increase. Some famous research centers such as ParLab at Berkeley, UPCRC-Illinois, and the Pervasive Parallel Laboratory at Stanford assume that future microprocessors will have hundreds of cores [2].

[1] Department of Electronic Engineering, Faculty of Engineering, University of Science and Technology, Sana'a, Yemen
[*] Correspondence Authors: (a.alkustaban@ust.edu), (a.qahtan2@ust.edu)

CMP performance can be improved by using cache memories. Cache memories significantly reduce both the bus traffic and the average access time. Moreover, associate a cache memory with each processor in the multiprocessor environment, is one of the most effective solutions to the bus bandwidth problem [3, 4].

Since the processors in CMP have fast clock rate, the gap between processors speed and the relatively long time required to access the main memory becomes wider. To narrow this gab, microprocessors designers add additional levels of cache. These additional levels, such as a second level (L2), are accessed whenever a miss occurs in the first-level (L1) of cache. If the L2 cache contains the requested data, the miss penalty for the L1 cache will be the access time of the L2 cache, which will be much less than the access time of the main memory. If the data is not found in the L1 nor the L2, a main memory access is required, and a larger miss latency is incurred [5]. Also, with multilevel cache the miss rate will be decreased as long as size increasing of the overall cache memories [6].

The primary challenge in a system that runs multiple processors is how to control the access to a shared data in order to ensure correct behavior and data consistency, especially when multilevel caches are used. In bus-based shared memory multiprocessor, many protocols used to control accesses to that shared data in order to keep the view of memory provided to the processors consistent. However, the write policies used between cache levels is a critical point that effect the overall cache system behavior and performance.

In this paper, a CMP with four pipelining MIPS processors is built to validate the proposed cache system. Then a private multilevel cache with snoop-based MESI cache coherence protocol is integrated with each processor of the CMP system. In addition, in this proposed work, many scenarios of multilevel cache system with different write policies and a single level cache system have been designed and implemented. Then a comparison of these scenarios is shown.

This paper is organized as follows. Section 2, reviews related works. Section 3, focuses on designing the proposed multilevel cache system. Section 4, implements the CMP system with the proposed multilevel cache system using Hardware Description Language (VHDL) on Field Programmable Gates Arrays (FPGA). Testing the proposed system and comparing it with other cache system architectures are presented in section 5. Finally, conclusion and future work are summarized in section 6.

2. Related Work

Although there are several proposed designs for CMPs use the private L1 cache structure of traditional multiprocessors, the organization of on-chip L2 resources is not researched well [7].

[8] is a survey that classifies the organizations of multilevel cache into three types. First, a private hierarchy of caches, this paper follows this type. Second, a private first-level with shared multiport second-level. Finally, a private first-level with bus-based shared second-level. Some of the proposed CMP designs implement multilevel cache systems with private hierarchy [7]. Other proposed designs implement systems with private L1 caches and shared L2 caches [9-11]. Other designs implement cache systems with two private levels and a third shared level among all cores [6, 12].

[13] compares three different approaches for ensuring cache coherence in Multiprocessor System on Chip (MPSoC) architectures, which are snoopy-based, software-based, and OS-based approaches. The comparison results of different snoop-based cache coherency schemes, reveals that these schemes have a strong sensitivity to the cache write policies more than the coherency protocol. Write-back schemes appear to be more efficient by reducing the traffic on the shared bus. Nonetheless, these schemes increase the hardware complexity of the cache-coherency, perform out-of-order eviction of data to the lower memory hierarchies, and unacceptably losses for the data consistency [14].

[15] compares the memory write policies in shared memory multicore system. It compares write-through invalidate protocol with write-back MESI protocol. Its results shows that write-through protocol is a simple and possible solution to maintain coherency. Write-through performs very well in both execution time and generated traffic. However, other proposed papers in this field say that write-through protocol perform well to ensure data consistent with drawback of higher write traffic as compared to write-back, since write-through requires synchronous updates for every write [14, 16].

[9] explores a novel approach to mitigating multicore power consumption by using dynamic application memory behavior. The main notice is that using only write-through policy causes more contention on the bus. Moreover, using only write-back policy makes the data in L2 stale relative to the newer data in L1 caches. The proposed solution is to allow a fine-grained hybrid interaction between write-through and write-back policies to improve the overall performance. However, it is become more complex to generalize an inclusive state diagram to manage diverse coherence protocols possessed by each core when the number of the cores is increased [17].

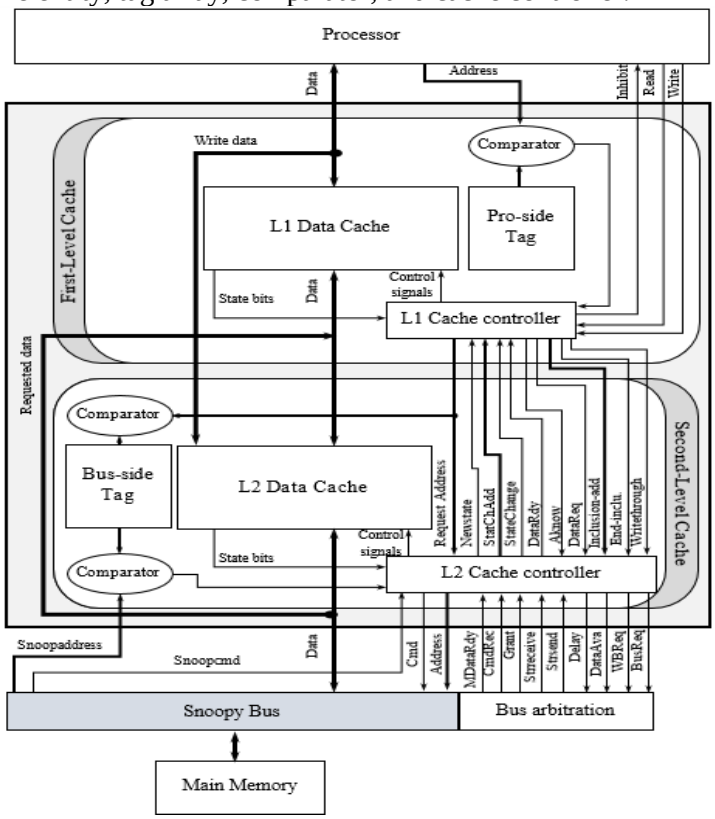
[14] develops and evaluates two consistent write-back caching policies, ordered and journaled, that are designed to perform increasingly better than write-through and traditional write-back policies. Paper's results show that ordered write-back performs better than write-through. Additionally, journaled write-back exceeds conventional write-back performance.

However, to my knowledge, no hardware implementations using HDL and FPGA are used to implement and test the effect of write policies on the behavior correctness of the multilevel cache systems. Moreover, the miss latency was not discussed in any previous works clearly.

3. The Proposed Multilevel Cache System Design

3.1 Multilevel Cache System Block Diagram

As shown in figure 1, the proposed multilevel cache system consists of two private levels of cache associated with each processor. Each level of cache consists of four main parts. Which are data cache entity, tag array, comparator, and cache controller.



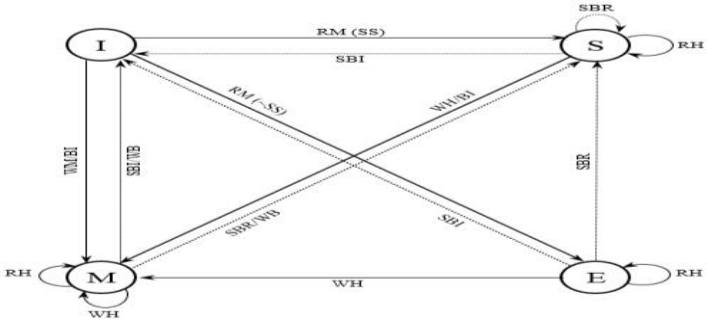
Figure(1): The proposed multilevel cache system block diagram

The data cache entity is direct-mapped architecture to save the frequent used data in its blocks. Each block has additional bits represent the block state and indicate if the block data is valid or not. The tag array contains an address information, which are required to identify whether the

data in the cache blocks corresponds the requested data. The comparator part is used to examine the address tag field with the associated tag bits in the cache. The cache controller is used to check the comparator results and the block state bits value, generate the hit/miss signal, and serve the request from the cache blocks or from the lower memory hierarchies.

3.2 Cache Coherence Protocol

The selected cache coherence protocol to be applied on the proposed cache system is the MESI [4], which is illustrated in figure 2 and the explanation of the abbreviation used in MESI state diagram are listed in table 1. Since there is no communicate between the L1 caches via the shared bus, the coherence protocol must be applied on the L2 caches which are in touch with the shared bus. L1 cache takes the required data the block state from L2 cache [5].



Figure(2):MESI state transition diagram

Tabel 1:The explanation of the abbreviation used in MESI state diagram

Abbreviation	Description
RH	Read Hit in local cache
RM	Read Miss in local cache
WH	Write Hit in local cache
WM	Write Miss in local cache
SBR	Snoop Bus Read from remote cache
SBI	Snoop Bus Invalidate from remote cache
SS	Shared signal on the bus
WB	Write back modified (dirty) data
BI	Send invalidate signal to all remote caches

3.3 Write Policies and Hit-Miss Actions

3.3.1 First-Level Cache

- **Write Policy*:*** The write policy used in L1 cache is write-through, where writes always update both the cache and the lower level of the hierarchy (either another cache level or main memory) at the same time. Using write-through policy insures that the data is always consistent between cache levels.
- **Hit State**: L1 cache controller monitors every request from the processor. When there is a request (Read/Write), the comparator examines the address, the cache controller checks the comparator results to generate hit or miss signal and serve the request. According to the MESI protocol, hit occurs in one of the following states:
- For read request: when the tags are equal and the block state is S, E, or M. Then the requested data is placed on the data bus.
- For write request: when the tags are equal and the block state is E or M. Then the data on the data bus is written to the corresponding cache block, the block state is updated to M if it was E, and the Writethrough signal is asserted to write the same data to L2 cache at the same clock.
- **Miss State:** When there is no hit, miss occurs and the Inhibit signal in figure 1 is asserted to freeze the processor datapath during the miss manipulation. If the L1 block has a dirty data belongs to other address, the cache controller asserts the EndInclusion signal and sends the dirty block address to L2 to inform it that this block will not be exist in L1. Then the cache controller asserts the DataReq signal and sends the requested data address to L2 cache. When the requested data is ready, L1 cache puts it with its state in the corresponding block and asserts the Aknow signal.

3.3.2 Second-Level Cache

- Write Policy
- The write policy used in L2 cache is write-back, which handles writes by updating values only to the block in the cache, the modified block is writing to the lower level of the hierarchy (the main memory) in two states. First, when it is replaced. Second, when it is invalidated by another processor. Using write-back policy is used to reduce the bus traffic and thus allows more processors on a single bus. During the write process, a write-back buffer is used to allow the processor to continue as soon as possible on a cache miss that causes a write-back. It is better to delay the write-back and instead first service the miss that caused it [4]. Moreover, a cache to cache transfer is supported in the proposed architecture in order to minimize the miss latency.

Hit State :L2 cache controller monitors every request from the L1 cache and from the bus side. When there is a request from L1 cache (Writethrough/EndInclusion), the request is served immediately since such requests cannot be happened if the data block does not exist

in L2 cache. When L1 cache request is DataReq, L2 cache first comparator examines the incoming address tag field with the associated tag bits in the cache. The cache controller checks the comparator results and the state bits value to generate the hit/miss signal and serve the request. Hit occurs in one of the following states:

- The request is done to perform a read process in L1: when the tags are equal and the block state is S, E, or M. Then the requested data is placed on the intermediate data bus.
- The request is done to perform a write process in L1: when the tags are equal and the block state is E or M. Then the requested data is placed on the intermediate data bus.

Miss State

When miss occurs, five steps are performed. First, asserting *BusReq* signal. Second, waiting for bus G*rant*. Third, putting address and command on the bus. Fourth, waiting for one of the acknowledgment signals (*CmdReceive, MDataRdy, and StrRec).* Finally, transferring the data between the shared bus, L2, and L1 at the same clock to reduce the miss penalty. Table 2 summarizes the issued commands and the block states when there is L2 cache request according to MESI cache protocol.

Tabel 2: The command and the block states values according to L2 cache request and MESI protocol

L2 cache Request	Cache Command (Cmd)	Bus Acknowledgment Signal	Block State
Invalidate other cache copies	011	CmdReceive	M
Read for exclusive	010	CmdReceive & MDataRdy	M
		CmdReceive & StrRec	M
Read for shared	001	CmdReceive & MDataRdy	E
		CmdReceive & StrRec	S
Write back	111	-	-
Release the bus	000	-	-

• Snooping Actions

Since the cache coherence protocol is applied in L2 cache because of its contact with the shared snoopy bus, L2 cache controller monitors (snoop) every transaction on the shared bus. The second comparator checks if the bus transaction belongs to one of its contents. Then the cache controller takes the comparator results and checks the block state bits to decide which action must be applied. Table 3 summarize the snooping actions according to the bus transaction and the block state.

Tabel 3: Snooping actions according to the bus transaction and the block state

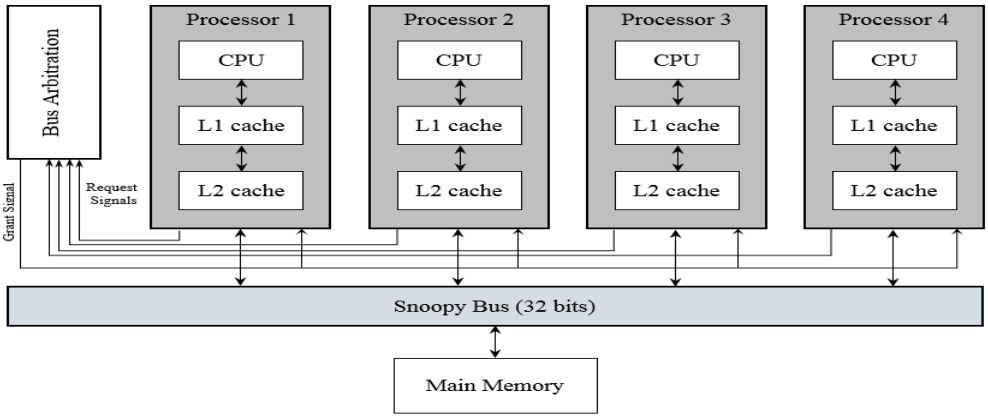
Bus transaction		Current block state	Snooper action	New block state
Request	Snooped Cmd			
Read for exclusive	010	S or E	• Assert *DataAvailable* signal to supply the request with the required data.	I
		M	• Assert *WBRequest* signal to supply the request with the required data and write the dirty data to the main memory, too.	I
Read for shared	001	M		S
		E	• Assert *DataAvailable* signal to supply the request with the required data.	S
		S		S
Invalidate other cache copies	011	S	-	I

The DataAvailable/WBRequest signals have the benefit of reducing the miss latency time. It informs the bus arbitration that the cache has an up-to-date (S or E / M) data, and it can supply the request with it. When there are many cache on different processors assert this signal, bus arbitration will choose one of them to supply the data by asserting StrSend signal to the chosen processor and StrRec with CmdReceive signals to the requester processor, this action is better than bring the requested data from the main memory , which may take more time. Also, WBRequest informs the bus arbitration to write the data to the main memory through the write-back buffer.

If the requested data is not exist in other caches blocks, the request will be supplied from the main memory. The bus arbitration will read the requested data and asserts MDataRdy with CmdReceive signals to the requester processor when the memory data is ready on the bus.

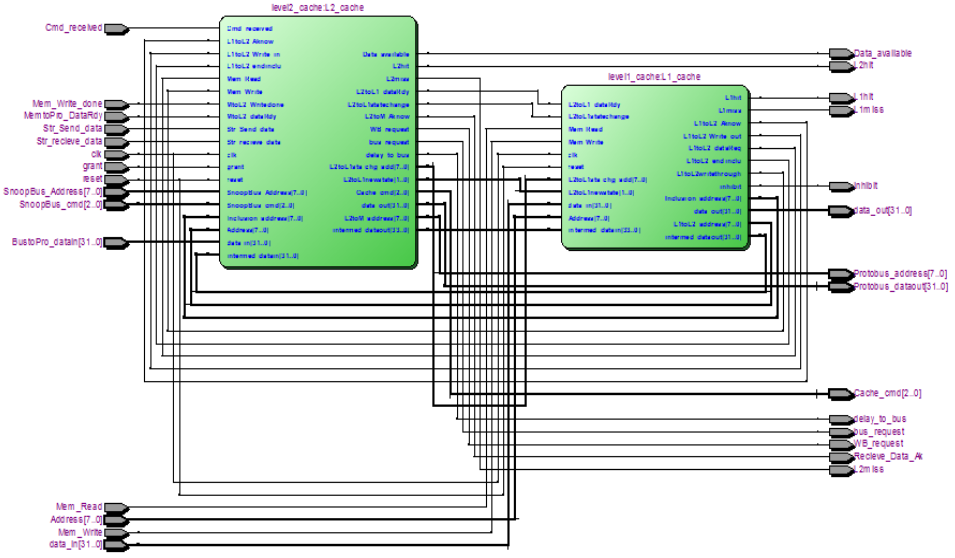
4. The Proposed CMP With Multilevel Cache System Implementation

To validate the proposed multilevel cache system, a CMP with four pipelining MIPS processors is built. Each processor has two private levels of data cache as shown in figure 3. A shared bus connects the whole system components with each other and with the off-chip main memory. The management of the shared bus is the responsibility of a central bus arbitration unit.



Figure(3): CMP system's block diagram

The CMP with the proposed multilevel cache system is implemented using VHDL language on Altera CycloneII (EP2C35F672C6) FPGA. Moreover, many scenarios of multilevel cache system with different write policies and a single level cache system have been designed and implemented. Figure 4 shows the top level Register Transfer Level (RTL) synthesis for the proposed cache system.



Figure(4): Proposed cache system top-level RTL synthesis

5. Result and discussion

This section compares between the scenarios that are implemented to show the appropriate architecture for the multilevel cache system. These scenarios are:

- Cache system with write-back L1 cache and write-back L2 cache.
- Cache system with write-through (via buffer) L1 cache and write-back L2 cache.
- The proposed cache system with write-through (without buffer) L1 cache and write-back L2 cache.
- Cache system with only L1 cache.

The results are obtained from a VHDL simulation tool from Mentor Graphics Company, which is ModelSim. ModelSim has an ability to illustrate the simulation results as waveform which is an easy way to recognize the required results.

The comparison factors are the miss rate, the miss latency (penalty) and the correct behavior of the system. Many tests, such as read/write sequence of data from the main memory and other processors' cache system, read and modify two blocks that are mapped to the same cache block from the main memory, have been applied on the four scenarios to verify them. Table 4, summarizes the implemented cache system test's result. The simulation results involve the required time transferring data through the bus. From the results, the proposed cache system improves the latency by 10% compare to the first and the second scenario. The fourth scenario is better than the other scenarios. This result is logically correct since the fourth scenario has only one level of cache. However, the fourth scenario suffers from high miss rate, which is solved by the second level of cache on the other scenarios. The required time to complete the write process to both levels is improved in the proposed system by 50% compare to the first scenario and by 66% compare to the second scenario.

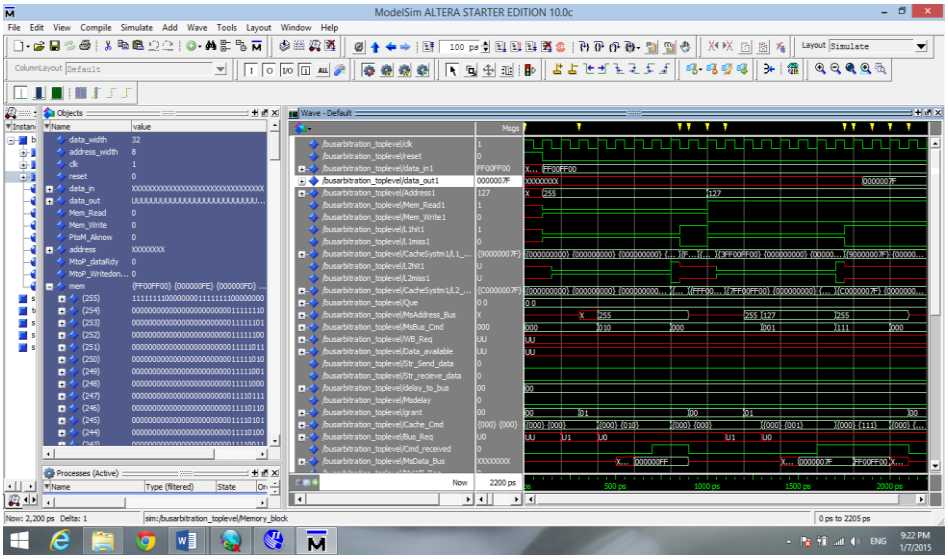
Tabel 4:Cache system test's result

Comparison Factor \ Cache System Architecture		Multilevel cache			Single level cache
		First Scenario	Second Scenario	The proposed Scenario	
Miss Rate		Low	Low	Low	High
Miss Latency (in clock cycle)	Main Memory	10	10	9	6
	Other cache system	9	9	8	5
	Lower Cache Hierarchy (L2)	3	3	3	-
Complete a write process from L1 to L2		2 on replacement	3	1	-
Correct Behaviour		x	x	√	√

Test results show that when write-back policy is used between L1 and L2 cache, the data consistency cannot be insured between all processors' cache systems. For example, suppose that a requested data brought to the cache system from the main memory. This data placed in L2 and L1 caches with (E) states. Then if the processor needs to modify that data, the new data will be modified only in L1 cache and the corresponding block state will be modified to (M) state in L1 cache, too. When other processor requests the same address, L2 controller in the owned processor will snoop that request. Then it will change its block state to (S) state and supply the request with a false data since the up-to-date data exists in L1 cache.

In addition, using write-through policy with buffer may cause a conflict between write-through address and the data request address in some cases, such as the following worst case example. Suppose that the memory blocks with addresses 255, 127 which are mapped to the same L1 cache block. In the beginning, both blocks are not exist in the processor's cache system. When the processor request address 255 to perform a write instruction. The request is done and the data is placed in L1 and L2 with (E) or (M) states according to MESI protocol. Then the new data write process is performed to L1 cache and the write-through buffer. During write-through process, written data address is sent to L2 cache to perform the write process. If the processor request address 127, that address will be sent to L2 cache because of the missing station in L1 cache. As long as write-through is not complete, the write-through address conflicts with the data request address. This confliction causes an ambiguity in L2 cache which may provide L1 cache with a false data.

Solving these problems, by using more checking circuits or splitting the address bus, can reduce the overall performance through increasing the miss penalty, the design complexity, and the overall chip area. Other scenarios' problems are solved in the proposed scenario by modifying the write techniques as described in section 3. The simulation results show that the same cases are performed on the proposed system without any confusion. The proposed cache system write the new data to the both levels in the same clock cycle. Moreover, writing the new data to L2 cache is done before writing it to L1 cache by half of clock cycle. That action makes the new data available to others cache system immediately. Figure 5 depicts the simulation waveform for the proposed cache system with the worst case example.



Figure(5): Simulation waveform for the proposed cache system

6. Conclusion and Future Work

In this paper, a CMP with multilevel cache system is designed and implemented using VHDL and FPGA. Using multilevel cache improves the overall performance of CMP. The two-Level direct-mapped data cache with a snoopy MESI cache coherency protocol is designed and implemented for each processor. The simulation and implementation results show that when the write-through policy without buffer in L1 cache is combined with the write-back policy in L2 cache, is the best combination to get the benefits of multilevel cache. Actually, this combination improves the miss latency by at least 10% compared to other architecture.

This paper can be extended by examining the effects of using more flexible placement techniques such as the set-associative to reduce the cache miss.

7. References

[1]K. Olukotun, L. Hammond, and J. Laudon, "Chip multiprocessor architecture: techniques to improve throughput and latency," Synthesis Lectures on Computer Architecture, vol. 2, no. 1, pp. 1-145, 2007.

[2]B. Catanzaro, A. Fox, K. Keutzer, D. Patterson, B.-Y. Su, M. Snir, et al., "Ubiquitous parallel computing from Berkeley, Illinois, and Stanford," IEEE micro, vol. 30, no. 2, pp. 41-55, 2010.

[3]V. S. Bhure and D. Padole, "Design of Cache Controller for Multi-core Systems using Multilevel Scheduling Method," in Fifth International Conference on Emerging Trends in Engineering and Technology (ICETET), Himeji, pp. 167-173, 2012.

[4]D. E. Culler, J. P. Singh, and A. Gupta, Parallel computer architecture: a hardware/software approach, Gulf Professional Publishing, 1999.
[5]D. A. Patterson and J. L. Hennessy, Computer organization and design: the hardware/software interface, Newnes, 2013.
[6]L. G. Menezo, V. Puente, and J. A. Gregorio, "The case for a scalable coherence protocol for complex on-chip cache hierarchies in many core systems," in 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT), Edinburgh, pp. 279-288, 2013.
[7]R. E. Ahmed, "Cooperative caching in power-aware chip-multiprocessors," in Canadian Conference on Electrical and Computer Engineering (CCECE'09), St. John's, NL, pp. 195-198, 2009.
[8]M. Tomasevic and V. Milutinovic, "A survey of hardware solutions for maintenance of cache coherence in shared memory multiprocessors," in The Twenty-Sixth Hawaii International Conference on System Sciences, pp. 863-872, 1993.
[9]G. Bournoutian and A. Orailoglu, "Dynamic, multi-core cache coherence architecture for power-sensitive mobile processors," in The 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), Taipei, pp. 89-97, 2011.
[10]J. Huh, "On the Partial Inclusion Protocol for Multi Level Cache Hierarchy in Disk Controller," in Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT) Luoyang, Henan, China, pp. 309-314, 2007.
[11]J.-M. Li, P. Yang, N. Ding, H. Guan, J. Zhang, C. Men, et al., "A New Kind of Hybrid Cache Coherence Protocol for Multiprocessor with D-Cache," in International Conference on Future Computer Science and Education (ICFCSE), Xi'an, pp. 641-645, 2011.
[12]Z. Pang, S. Wang, D. Wu, J. Zhang, and P. Lu, "Building efficient transactional memory support based on snoopy coherence," in 3rd International Conference on Computer Research and Development (ICCRD) Shanghai, pp. 46-50, 2011.
[13]M. Loghi, M. Poncino, and L. Benini, "Cache coherence tradeoffs in shared-memory MPSoCs," ACM Transactions on Embedded Computing Systems (TECS), vol. 5, no. 2, pp. 383-407, 2006.
[14]R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala, and M. Zhao, "Write policies for host-side flash caches," in 11th USENIX Conference on File and Storage Technologies (FAST'13), pp. 45-58, 2013.
[15]P. G. de Massas and F. Pétrot, "Comparison of memory write policies for NoC based multicore cache coherent systems," in Design, Automation and Test in Europe (DATE'08), Munich, pp. 997-1002, 2008.
[16]A. K. AG, J. DA, and R. M, "Power and performance efficient secondary cache using tag bloom architecture," in International Conference on Electronics and Communication Systems (ICES), Coimbatore, pp. 1-5, 2014.
[17]M. Lou and J. Xiao, "Dynamic, Tagless Cache Coherence Architecture in Chip Multiprocessor," in Foundations of Intelligent Systems: Springer, pp. 201-209, 2014.