Exploring Cache Coherency Design for Chip Multiprocessor using Multi2Sim

Vinh Ngo Quang and Hao Do
IC Design Research and Education Center,
VNUHCM
Ho Chi Minh city, Vietnamese

Trang Hoang and Thanh Vu D.
University of Technology,
VNUHCM
Ho Chi Minh city, Vietnamese

Abstract—**Memory hierarchy design plays an important role in improving the performance of chip multiprocessor (CMP). The reason is that the performance of a CMP is strongly affected by the latency of fetching data from the memory system. Several organizations of the memory hierarchy have been explored to optimize this latency. In the memory hierarchy, data traversing is based on a cache coherence protocol which is the skeleton of the CMP's memory system. In this paper, we concentrate on exploring MOESI, a well-defined and popular cache coherence protocol in CMP. Our experiment is based on Splash-2 benchmark which is widely used in every publication regarding CMP design. The experiment results show that by rearrange the address range of the memory banks, L2 hit ratio could be improved up to 13,5 %.**

Keywords—Chip Multiprocessor; Memory Hierarchy; Coherence Protocol; MOESI; Memory Bank

I. INTRODUCTION

Nowadays, chip multiprocessor (CMP) is the main trend in designing the CPU for high performance devices. This originates from the fact that the single core chip reaches the limitation of execution speed because of the heat and power dissipation issues. Moreover, modern technologies support millions of transistors to be integrated in one chip which eases
the design of multicore on chip in terms of area. In fact, several CMPS have been commercialized in the market [1], [2], [3], [4].

In CMP chip, memory hierarchy design is a concern that takes a lot of effort of researchers. The memory organization, but not the CPU core, is the bottleneck in CMP design. Most of the memory systems have multiple levels of cache hierarchy. For instance, Intel’s commercial Ivy Bridge has 3 levels of cache. These cache levels and the main memory need a coherence protocol in order to keep the memory consistent. For example, a L2 cache must ensure data consistency among L1 caches. Moreover, a good coherence protocol also helps the CMP to improve the performance in terms of memory access latency. To the best of our knowledge, MOESI [6] is the most widely used cache coherence protocol in CMP. In this paper, we first evaluate the CMP performance by using MOESI protocol. Then, by observing that the instruction and data cache have different ways of access model, we concentrate on optimizing the last level cache memory to adapt the difference and get better performance in terms of L2 hit ratio. Our idea is to interleave the address range of the memory banks and we show that, by simulation, the performance could improve up to 13,5% in comparison with a baseline model that arranges the memory address linearly. The experiment is carried out with Multi2Sim [5], an open source simulator for heterogeneous multiprocessor design. Splash-2 benchmark [17] is used as the workload in the experiment. In the experiment result, we focus on analyzing the cache miss latency because this parameter not only can determine the efficiency of the MOESI but also strongly affects the CMP performance. Our contribution in this work is (1) the statistical result of the CMP performance using MOESI protocol, (2) demonstration of the performance improvement by rearranging the address range of the memory banks.

The paper is structured as follows. Section II surveys some of the latest work on CMP’s cache organizations and coherence protocols. Section III presents the experiment method. Section IV gives and explains the result. And section V finalizes the paper.

II. RELATED WORK

Several works have been carried out to improve the CMP’s performance by optimizing the on chip memory hierarchy. There are different aspects to look at in the memory hierarchy such as: the shared or private last level cache model, the cache coherence protocol, on-chip interconnection and so forth. While L1 cache is always private to the processor core, the L2 cache can be designed to be private or shared. Many research papers exploit the possibilities in designing the L2 cache [7]. Shared last level cache has an advantage in comparison with private cache is that it dynamically allocates the overall cache space for all cores on chip. Thus, the last level cache space is better utilized and its miss ratio is therefore reduced. Shared last level cache (LLC), can be physically centralized or distributed with respect to the processor cores. In the first designs of CMP, researchers proposed the shared LLC organization to have uniform cache access time (UCA). Even though the UCA is simple for designing, it was soon replaced by non-uniform cache access (NUCA) techniques. With NUCA, nearer cache banks will provide lower access latencies than further banks with respect to the requesting core. NUCA was first proposed in [9]. The complexity of the interconnection network is the price to pay as using NUCA. The latency for a bank access depends on its size and the network hops distance between the requesting core and the bank. Kim et al. [9] investigated a model with a

single core with a large L2 cache divided into multiple banks. They argued that a highly-banked cache structure with distributed cache controller is desirable in reducing the cache access latency. Two main techniques for NUCA were proposed in their paper are static UCA (S-NUCA) and dynamic UCA (D-NUCA). In S-NUCA, data is statically mapped into banks with the least significant bits determining the bank. S-NUCA is proven to have more advantages than UCA in [9] because of two reasons. Firstly, the banks have non-uniform access times and thus accesses to the nearer bank to the requesting core incur lower latency. Secondly, different banks can be accessed
simultaneously which helps to reduce the contention. The D-NUCA further improves the performance of S-NUCA by dynamically mapping data into different LLC banks. In other words, frequently accessed data are placed in closer banks while less used data are cached in farther banks. The D-NUCA leads to the data management policies issues. The key issues in data management for D-NUCA are (1) how the data are mapped to banks and which banks a specific data can be reside, (2) how to search a cache line as quick as possible in a large cache with multiple banks and (3) how and when to migrate the data between banks of the cache. D-NUCA is widely exploited in several researches [10], [11], [12]. To the best of our knowledge, there is no research paper that refers the way in which the entire memory address range is divided into different memory banks. In this paper we show that the L2 hit ratio can be improved up to 13,5% by interleaving the memory address range between different memory banks. To ensure a consistent view of memory between all processor cores, a cache coherence protocol needs to be implemented in CMP. Two main parts of a coherence mechanism are: (1) a storage that holds the data sharing information and (2) a set of protocols to keep the consistency of the data using the information in (1). One essential information of the data sharing is their status. The status of the cached copies of any data block is usually kept by attaching the state to each cache data block. The minimum states that a coherence protocol must have are: (1) the invalid (I) state which indicates that the cache block is not holding the valid data; (2) the shared (S) means that the cache block is shared by one or more other processor caches in the system, it also means that this block can only be read from (not written to) and it is holding the same data value with the memory; (3) the modified (M) to signify that the block is uniquely held. If a cache block is in M state, it must be written back to the memory before being evicted. This simple coherence protocol is therefore named as three-state MSI protocol.

More sophisticated protocols employed more cache block states to reduce the coherence traffic and the latency of fetching a data block. Some popular protocols are MESI, MOSI, MOESI [13]. MOESI is considered to be the most complex protocol which encompasses all the possible states commonly used in other protocols. The O is added to describe a dirty and shared block. This state helps to reduce the coherence traffic because a block in M state doesn't need to write back the data when it receives a read request and just changes to O state instead. On the other hand, state E is implemented to signify a clean and exclusive block. A block can change to M state without the need of notifying the lower evel cache/memory. Besides, when a block in state E is evicted, it doesn't need to write back the data because the data is clean. Based on research papers, it seems that MOESI is better in terms of performance comparing with MESI, MOSI. Anyway, there is no evidence that prove it.

Depends on the interconnection, there are two major cache coherence protocols: bus-based and directory-based. Directory-based is desirable for big number of cores on chip using scalable interconnection such as mesh. The directory-based protocol breaks the broadcast coherence message as in bus-based into point-to-point messages that only involve appropriate nodes in the interconnection network. In more detail, the sharing information is logically centralized into a directory. The directory is usually co-located with the data block in the memory and each of its entry corresponds to one memory address. The entry keeps essential information to track the memory block's current sharers and their read/write privileges. The directory-based mechanism dramatically reduces the coherence traffic in comparison with bus-based. Moreover, it allows coherence messages to traverse over the dedicated and fast channels on the interconnection network rather than a single shared bus as in the bus-based mechanism. In this paper, we choose the MOESI and directory-based coherence protocol for the sake of the performance and the scalability [8].

The simulator used in this research is Multi2Sim. It models an event-driven memory hierarchy which uses MOESI as the coherence protocol between caches from different processor cores. It also supports multi-level cache organization as well as directories for caches and main memory. The simulator is written purely in C language which is more simple to read and modify the code in comparison with GEM5 [16]. Other well known simulators for this research are CACTI [14] and Simics [15]. But CACTI solely builds the cache model which costs users more effort to run the simulator with benchmarks while Simics is mainly for commercial usage.

III. METHODOLOGY

We use Multi2sim to run and simulate the operation of CMP, focusing on memory hierarchy. Multi2sim is a simulation framework written in C for heterogeneous computing. It provides an easy way to design, configure, launch the CMP for research purpose on heterogeneous system. This framework allows creating the CMP by using INI files. We can intercede to the main memory, cache, internal network from these files. In the first experiment, we use INI files such as memory configuration and network configuration to generate a CMP given in Fig. 1.

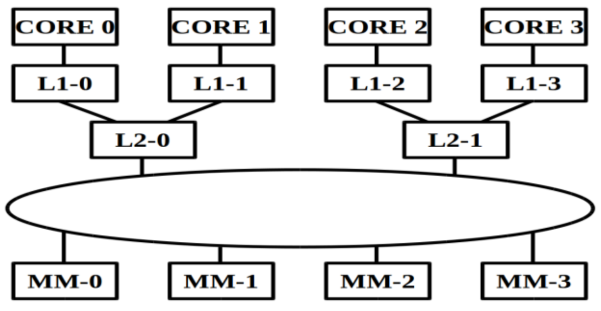

Fig. 1. The CMP architecture used for experiment.

The CMP composes 4 processor cores, 4 L1 cache modules, 2 L2 cache modules and a main memory module. Each L2 cache is shared for 2 L1 cache modules, but is private from the main memory viewpoint. We also create 4 threads for our four-core CMP. The detail configuration of this system is shown in TABLE I.

Element	Properties	Parameter
CMP	Number of cores	4
	Architecture	X86
	Cache	L1, L2
Cache L1 (Data)	Number	4
	Set	128
	Associative	2
	Blocksize	256
Cache L1 (Instruction)	Number	4
	Set	128
	Associative	2
	Blocksize	256
Cache L2	Number	2
	Set	512
	Associative	4
	Blocksize	256
Main memory	Total size	4 GB
	Number of banks	4
	Internal network	Ring network
	Address range	Linear

TABLE I. Configuration of CMP in detail

We use a ring network to connect all modules of main memory. This helps L2 private caches can access directly to every memory banks in the ring. The MOESI protocol is used in this CMP design to keep data consistent between L1 caches of processor cores.

The experiment is run with Splash-2 benchmark. This benchmark contains 11 applications that solve 11 computing problems such as: N-Body, Cholesky factorization, FFT, etc. These are the most classical problems in parallel computing theory. As we know, the main application field of CMP is solving the complexity problem by parallel computing. That is the reason we choose this benchmark to evaluate MOESI protocol in CMP performance. We run 11 applications and record the L1 and L2 hit ratio. The result is reported in Fig. 2. As we can see, the average hit ratio of L1 is very high, over 98% for L1-Data and 99% for L1-Instruction. When it comes to L2, this number is lower, about 77%. These are relevant results because L1 cache contains the most likely to used data and instructions. L2 hit ratio is less than L1 but also acceptable because L2 is one level lower and processor core can proceed with other processing tasks that do not need to wait for the data from L2. Besides, in this configuration, L2 effective size is limited by its private characteristic. With this result, we can agree that Multi2sim simulator and the MOESI protocol work correctly. More importantly, we divide the main memory into 4 equal banks. Each bank preserves a continuously range of memory address. This method of separating bank is considered as the baseline for the next experiment which shows the improvement of L2 hit ratio by interleaving the bank's address range.

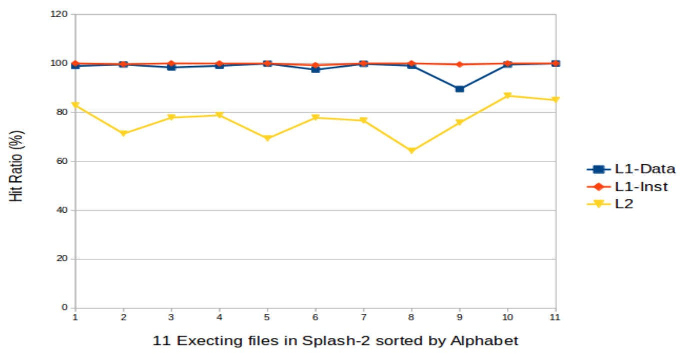

Fig. 2. The average of hit ratio in L1 and L2 cache.

In the second experiment, the main memory in this model is separated into 4 banks by interleaving the memory address range. There are 32 bits to index memory bytes, but we do not use all of them. When using interleaving to divide the main memory, a memory bank is a set of equal smaller range, if the interleaving ranges are too small, it's not efficient for memory access pattern because of its locality of reference. Thus, we choose the smallest range is 1 KB, this means that we have 22 bits to index a range. On the other hand, we have 4 memory banks, and also, we need to employ a pair of bits to locate the memory bank which contains the block. In theory, we can use any pair for this task, but we use 2 consecutive bits within the 22 index bits for our experiment. This approach, which is called interleaved address memory bank, helps us to spread the data and instructions into every memory bank. Fig. 3 illustrates the cases of dividing memory address into banks. There are 21 pairs of consecutive bits in 22 index bits. So that we can divide the main memory by 21 difference ways based on these pairs of bits. Our purpose, finally, is identifying which pair is the best choice, and how much it improves the hit ratio in L2. Firstly, We run the model in Fig. 1 for 21 cases using a benchmark from Splash-2 and get the L2 hit ratio of each case. We randomly choose the Sparse Cholesky Factorization problem. After getting the result, we determine the best pair for interleaving. Next, we run all remaining benchmarks in Splash-2 with that pair and compare the result

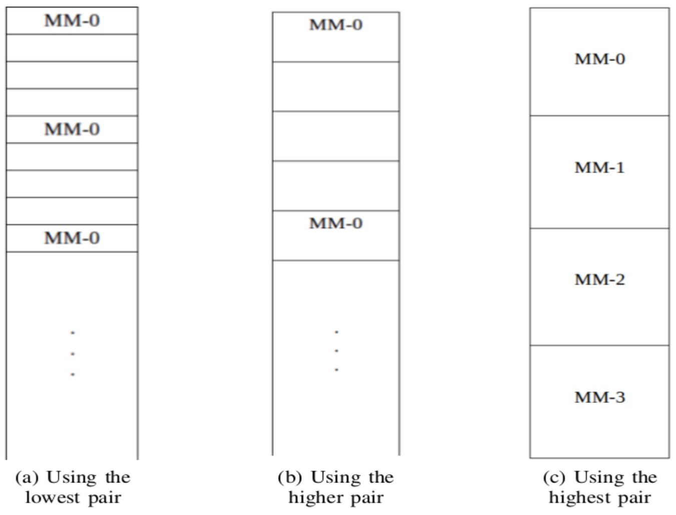
MM-0
MM-0
MM-0
MM-0
MM-0
MM-0
MM-1
MM-2
MM-3
(a) Using the lowest pair
(b) Using the higher pair
(c) Using the highest pair

Fig. 3. Memory partitions using difference pair of bits

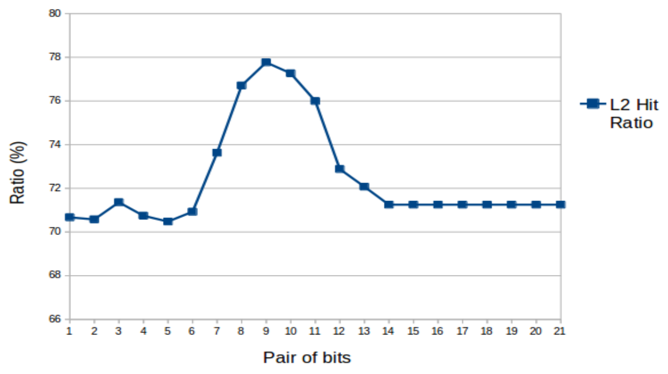

Fig. 4. The hit ratios in L2 based on using difference pairs of bits to partition main memory when launching Cholesky factorization

with the based line. After that, we compute the improvement when using interleaved banks versus linear range banks model. This design is an inclusive cache design so that the directories in 4 main memory banks limit the number of cache blocks in all cache modules level 1 and level 2. If the directories are full, cache blocks in the directories are evicted and therefore the corresponding blocks in L1 and L2 caches are also evicted. Thus, we predict that dividing the main memory banks into 4 linear address range banks should give the lower performance than that when interleaving the address range into banks.

IV. EXPERIMENT RESULTS

Fig. 4 shows 2 important information after running Cholesky factorization. We see that the hit ratio interestingly reaches the highest value when we use the ninth pair of bits for partitioning. Besides, when the interleaving size is too large, hit ratio is a constant.

First, we present why the hit ratio in L2 is a constant when the size of interleave is large. Because the size of the application is limited, when the pair of bits is high, or the range is large enough, the application code fits within one bank. Actually, the hit ratio is not a constant due to some objective reason such as hardware, other programs ..., but it fluctuates around a number with an amplitude. This amplitude
is very small, so we can not recognize its appearance.

Secondly, the hit ratio reaches the maximum value when we use the ninth pair of bits corresponding with the range of each interleaved bank is 256 KB. The reason for this it that L2 is a unified cache for both instruction and data. Besides, CMP is running 4 threads in parallel. By interleaving the memory address range, the instruction and data of applications are distributed equally into 4 directories of main memory banks. This mechanism of memory banking should help the directories efficiently store the cache block with regarding its limited capacity. Moreover, when a cache block in the directory is evicted, it is high potential that the block will not be accessed again in upper level caches.

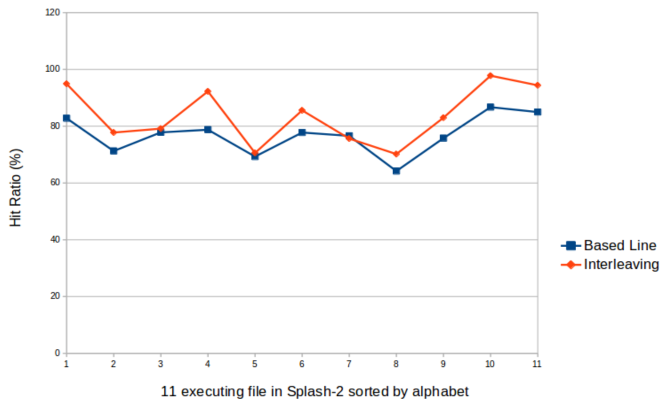

Fig. 5. The hit ratios in L2 when using interleaving 256 KB with base line in comparison

In Fig 5, the above line present the hit ratio when using interleave 256 KB, the other is base line. The average improvement is 7% and up to 13.5% in the best case. So, we can make the conclusion for these experiments that using interleaving method is the good choice.

V. CONCLUSION

The paper explores the MOESI protocol used in CMP. Based on the different characteristic of instruction and data cache block, we reorganize the memory bank by interleaving the address range between the banks. And the experiment result shows that by interleaving 256 KB between 4 memory banks, the L2 hit ration of the CMP improves up to 13,5% compared with the baseline model in which the memory address is divided into 4 continuous address ranges. This result should be applicable for other different workloads. Depends on the size of the workload, the interleaving coefficient might change
accordingly to achieve the best L2 hit ratio.

ACKNOWLEDGMENT

This research was funded by Vietnam National University - Hochiminh city under grant number C2015-40-01.

REFERENCES

[1] Chen, Thomas, et al., “Cell broadband engine architecture and its first implementation - a performance view,” IBM Journal of Research and Development 51.5 (2007): 559-572.

[2] George, Varghese, T. Piazza, and H. Jiang., “Technology Insight: Intel Next Generation Microarchitecture Codename Ivy Bridge,” (2011).

[3] Conway, Pat, et al., “Cache hierarchy and memory subsystem of the AMD Opteron processor,” IEEE micro 30.2 (2010): 16-29.

[4] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: A 32-way multi- threaded spark processor,” IEEE Micro, vol. 25, pp. 21-29, March 2005.

[5] Ubal, Rafael, et al. ”Multi2Sim: a simulation framework for CPU-GPU computing.” Proceedings of the 21st international conference on Parallel architectures and compilation techniques. ACM, 2012.

[6] Milo M. K. Martin. Token Coherence, Ph.D. Dissertation. Dec. 2003.
[7] Balasubramonian, Rajeev, Norman P. Jouppi, and Naveen Muralimanohar. “Multi-core cache hierarchies,” Synthesis Lectures on Computer Architecture 6.3 (2011): 1-153.
[8] Martin, Milo MK, Mark D. Hill, and Daniel J. Sorin. “Why on-chip cache coherence is here to stay.” Communications of the ACM 55.7 (2012): 78-89.
[9] Kim, Changkyu, Doug Burger, and Stephen W. Keckler,“An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches,”Acm Sigplan Notices. Vol. 37. No. 10. ACM, 2002.
[10] Chang, Jichuan, and Gurindar S. Sohi, “Cooperative caching for chip multiprocessors”, Vol. 34. No. 2. IEEE Computer Society, 2006.
[11] Zhang, Michael, and Krste Asanovic, “Victim replication: Maximizingcapacity while hiding wire delay in tiled chip multiprocessors,” ACMSIGARCH Computer Architecture News. Vol. 33. No. 2. IEEE Computer Society, 2005.
[12] Beckmann, Bradford M., and David A. Wood, “Managing wire delay inlarge chip-multiprocessor caches,” Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on. IEEE, 2004.
[13] Sorin, Daniel J., Mark D. Hill, and David A. Wood, “A primer on memory consistency and cache coherence,” Synthesis Lectures on Computer Architecture 6.3 (2011): 1-212.
[14] Wilton, Steven JE, and Norman P. Jouppi, “CACTI: An enhanced cache access and cycle time model,” Solid-State Circuits, IEEE Journal of 31.5 (1996): 677-688.
[15] Magnusson, Peter S., et al., “Simics: A full system simulation platform”,Computer 35.2 (2002): 50-58.
[16] Binkert, Nathan, et al., “The gem5 simulator,” ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.
[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, June 1995.