Efficient Address Mapping of Shared Cache for On-Chip Many-Core Architecture

Fenglong Song, Dongrui Fan, Zhiyong Liu, Junchao Zhang, Lei Yu, and Weizhi Xu

Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{songfenglong,fandr,zyliu,jczhang,yulei,xuweizhi}@ict.ac.cn

Abstract. Performance of the on-chip cache is critical for processor. The multi-thread program model usually employed by on-chip many-core architectures may have effects on cache access patterns and eventually on cache conflict miss behaviors. However, the behavior of cache is still unclear, and little has been known of the effectiveness of XOR mapping scheme for many-core systems. In this paper we focus on these problems. We propose an XOR-based address mapping scheme for on-chip many core architecture to increase performance of cache system. Then we evaluate the proposed scheme for various applications, including an application for bioinformatics, matrix multiplication, LU decomposition, FFT from Splash2 benchmarks. Experimental results show that with the proposed scheme, it makes conflict misses of shared cache reduced by about 53% on average, and makes overall performance improved by about 6%. Experimental results also show that the XOR scheme is more cost effectively than victim cache scheme.

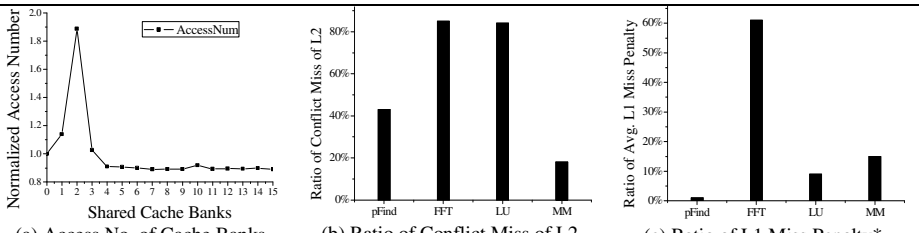
Keywords: Many-Core Architecture, Shared Cache, Address Mapping, XORM.

1 Introduction

On-chip multi-processor or many-core architecture is a promising trend of computer architecture [7,14,15,16]. Meanwhile, the *Memory Wall* hampers performance improvement further [14]. So a high-performance on-chip memory system for many-core architecture is necessary to reduce accessing latency and the number of accessing off-chip memory. It benefits more from shared last level cache than private design. However, it is a well known problem that conflict in shared table-based structure in computer architecture, such as TLB, cache et al. For the promising multi-thread program model [14], because there may be many threads accessing to shared memory simultaneously, it maybe make conflicts deteriorated. Many-core processors' performance may be depressed by tree saturation congestion [9] of on-chip network. Tree saturation occurs when packets routed to the shared cache are blocked in the way, they will block the tailing packets with another routing target. The assumption is validated by evaluation via Godson-T many-core architecture simulator [34]. As shown in Fig.1(a), a lot of memory access is concentrated to some certain cache banks. The worst evaluation results are shown as Fig.1 (b) and (c). It shows that conflict miss rate of the shared cache is not negligible, and it occupies about 80% of total

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 280–291, 2010.

shared cache misses for FFT and LU. As a result, the average ratio of level-one data cache miss penalty is worse, such as the worst case for FFT is about 60%. Here, level-one cache miss penalty is the practical executing cycles from occurrence of L1 cache miss to refill completion.



(a) Access No. of Cache Banks (b) Ratio of Conflict Miss of L2 (c) Ratio of L1 Miss Penalty*

*Ratio of Miss Penalty = Cycle of Processing L1 Cache Miss / Total Thread Executing Cycle

**L1 is private to processing core; L2 is shared by all on-chip processing cores

Fig. 1. Conflict on Many-Core Processor**

However, there are few researches focus on design and evaluation of conflict avoiding schemes with low hardware complexity in shared cache on many-core architecture. A simple but effectual solution is address mapping scheme. To the best of our knowledge, there is no a work about performance and properties of conflict-free XOR-based mapping scheme on many-core architecture with shared static NUCA cache [5]. All of these motivate us to focus on effect of XOR-based scheme in many-core processor and how to design a proper address mapping scheme with relative lower hardware implementation cost, complexity and lower computing latency, but with high performance. In this paper we propose an XOR address mapping scheme, called as XORM, for many core architecture. The evaluation is performed with Splash2 benchmarks and Bioinformatics workloads via a cycle accurate many-core processor simulator. Experimental results show the effectiveness of the proposed XORM mapping scheme.

The rest of this paper is organized as follows. Section 2 describes related works. Section 3 proposes the XORM address mapping scheme, include theoretical formulation and XORM-based set index and interleave bits mapping scheme of shared static NUCA cache. Next section 4 describes the simulation infrastructure and workloads, then presents effectiveness of several address mapping schemes and victim cache. We conclude in section 5.

2 Related Work

There are lots of schemes to reduce conflict in cache, such as skewed-associative cache [2], victim cache [21], column-associative cache in [3] and prime-indexed cache [17,18,22]. The main shortcomings of these schemes are computing complexity and implement cost.

Two advantages of XOR-based mapping scheme are lower computing latency and lower implement cost. XOR-scheme has been researched and applied in computer

architecture widely, such as TLB [6], BTB [12], parallel memory[13], memory buffer [31] and cache [1,10,11,23,26,27,30], et al. Z. Liu et al presented a new XOR-based unsymmetrical address mapping functions in [29] and [30], called EE. González et al [1] analyzed the impact of XOR-based placement functions to cache performance. Vandierendonck et al [10] proposed an application-specific XOR-based hashing function in embedded systems, whose goal is to improve the trade-off of power/efficiency. Then they presented an algorithm to compute the optimal XOR-based functions to avoid maximum conflict miss in [11]. Then they analyzed conflict in cache by the null space and column space of mapping function in [12]. Sung-Jin Cho et al presented a definition of hashing function in [26], but their disadvantage is to use all address bits to compute set index, so it increases input ports to XOR gate and so the complexity and latency.

Although the traditional XOR-based mapping schemes are effective in single processor, when it comes to on-chip many-core architecture, especially homogenous many-core architecture, their performance has to be depressed, because they are symmetrical mapping scheme. But the disadvantage of EE unsymmetrical mapping scheme is that it has been evaluated on single-core processor only, and it should introduce all index bits to generate image index bits. But in the many-core architecture with static NUCA Cache, it cannot map similar enough address bits to different indexes if it does not introduce interleave bits to generate index bits. So we propose XORM unsymmetrical address mapping scheme, in which introduces different mapping matrix depending on even or odd bits. In addition, XORM is evaluated via cycle accurate many-core processor simulator, rather than analyze it theoretically only as previous works do.

3 XORM Address Mapping Scheme

3.1 Formulation of Address Mapping Scheme

For shared NUCA cache of many-core architecture, it is critical to avoid bank conflict. So we introduce XORM mapping scheme, which can be used on set index and interleave bits of cache bank. It is proven in [12] that two addresses that are conflict are always different in their least significant bits. So XORM includes two different matrices for even or odd mapping bits, and our goal is to introduce the most impactful bits to compute, but limit the input ports of XOR gate at the same time.

Assume mapping vector $\boldsymbol{M}$ to $\boldsymbol{N}$, where length of $\boldsymbol{M}$ and $\boldsymbol{N}$ is m. We can describe address mapping functions by an $m \times m$ binary matrix, where h_{ij}=1 if and only if the i^{th} bit is an input to the XOR-gate which computes the j^{th} bit of the set number.

Definition 1: The XORM mapping scheme is defined as:

$\boldsymbol{H} = (h_{ij})_{m \times m}$, where h_{ij} is

(1) When m is odd, *i.e.* $m = 2k + 1 (k \in N)$,

$$h_{ij} = \begin{cases} 1, & 0 \le i \le m-1, i+j = m-1; \\ 1, & 0 \le i \le k, \quad j \ge 0, \quad and \quad i+j \le k; \\ 0, & otherwise \end{cases}$$

(2) When m is even, *i.e.* $m = 2k$ $(k \in N)$,

$$h_{ij} = \begin{cases} 1, & 0 \le i \le m-1, i+j = m-1; \\ 1, & k \le i \le m-1, k \le j \le i; \\ 0, & otherwise \end{cases}$$

We introduce another binary vector $\boldsymbol{K}$ whose length is m, and the result vector is $\boldsymbol{N} = \boldsymbol{MH} \oplus \boldsymbol{K}$. □

The main difference between EE and XORM scheme is that their mapping matrixes are different. EE scheme employs anti-diagonal unit matrix, while XORM scheme defines different mapping matrices when m is even or odd. For example, when $m = 3$ and $m = 4$, the matrices are H_1 and H_2 as follows.

$$H_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{, and} \quad H_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

From ***Definition 1***, we can map any length of vector to another vector via some logical XOR operations, and these operations can be implemented simply by logical gate circuit in computer architecture with negligible computing latency. For example, for binary vector $\boldsymbol{A} = (a_2, a_1, a_0)$ and $\boldsymbol{K} = (k_2, k_1, k_0)$, where $a_i = 0$ or 1, $k_i = 0$ or 1, (i = 0, 1, 2), so we can get the result binary vector $\boldsymbol{s}$ as follows.

$$\mathbf{s} = \mathbf{A}\mathbf{H}_1 \oplus \mathbf{K} = (a_2, a_1, a_0)\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \oplus (k_2, k_1, k_0) = (a_2 \oplus a_1 \oplus a_0, a_2 \oplus a_1, a_2) \oplus (k_2, k_1, k_0)$$

$$= (a_2 \oplus a_1 \oplus a_0 \oplus k_2, a_2 \oplus a_1 \oplus k_1, a_2 \oplus k_0)$$

Intuitionally, we can deduce that XORM mapping scheme can map two access patterns conflict-free, given that XOR computing result of combination of corresponding bits of $\boldsymbol{A}$ and $\boldsymbol{K}$ is different. Now we present some properties which are proved in [33], which indicate that XORM address mapping scheme can avoid conflict in shared table-based structure, *such as* cache, in on-chip many-core architecture.

Property 1: All matrices H defined by ***Definition 1*** are nonsingular.

Property 2: The Null Space of all matrices H is one-dimensional linear space.

Property 3: The mapping functions corresponding to matrices H can map many frequent access patterns conflict-free.

Property 4: XORM mapping scheme is a surjection.

Property 5: The probability of each set index occurs is equal to the probability of each address occurs.

An address mapping scheme should ensure that all set indices would be used with equal probability; otherwise it may incur pathological cache access behavior. The Property 4 and Property 5 illustrate them. And these properties ensure that conflict-free access to a memory access patterns with M addresses, including multi patterns such as rows, columns, (anti)diagonal, and rectangles et al, given the shared cache has M set indices when set associativity of the cache is 1.

Next, we will show how to design XORM address mapping scheme in static NUCA cache of on-chip many-core architecture.

3.2 Address Mapping Scheme of Shared Cache

In static NUCA cache [5], a logical set is cached in one bank, and one memory address would be routed to one certain cache bank only.

Let address is n bits, then address $A = \langle a_{n-1}, a_{n-2}, \ldots, a_0 \rangle$ can be divided into six parts, *that is*, $A = \langle A_5, A_4, A_3, A_2, A_1, A_0 \rangle$, where $A_0 = \langle a_{b-1}, \ldots, a_0 \rangle$, $A_1 = \langle a_{k+b-1}, \ldots,$

a_b>, A_2 = <$a_{m+k+b-1}$, …, a_{k+b}>, $A3$ = <$a_{2m+k+b-1}$, …, a_{m+k+b}>, A_4 = <$a_{2m+2k+b-1}$, …, a_{2m+k+b}>, A_5 = <a_{n-1}, …, $a_{2m+2k+b}$>, where A_0 is offset, A_1 is interleave bits of cache bank, A_2 is set index, A_3, A_4 and A_5 compose to the tag bits, as shown in Fig.2. In this paper, we assume that bits of tag are larger than sum of bits of index and interleave bits; otherwise, we can get A_3 and A_4 from other lease significant bits in the similar way.

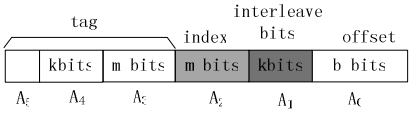


Fig. 2. Division of Memory Address

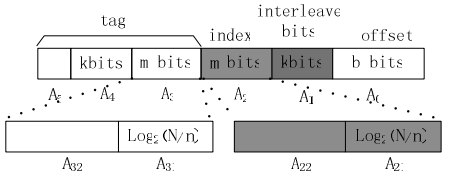


Fig. 3. Simplified Address Division

Every part of memory address can be taken as a binary vector in GF(2). According to Definition 1, address mapping scheme can be described as a vector-matrix multiplication in GF(2), in which addition is actually a logical XOR and multiplication is a logical AND. We can apply XORM mapping scheme to set indices and interleave bits as presented by following vector-matrix multiplication:

$$index = A_3 H_{mxm} \oplus A_2;\ interleave = A_4 H_{kxk} \oplus A_1 \tag{1}$$

As Definition 1, for an m bits mapping scheme, the input ports of XOR-gate are different, and the maximum is $\lceil m/2 \rceil + 1$. It is well known that the computing latency and area of an XOR gate are increasing with the number of input ports. In the on-chip many-core architecture with shared static NUCA cache, we assume that there are N processing cores on chip, and an n-way set associativity shared cache. We assume the execution model is non-preempt multi-thread model further. So what we should do is to map data blocks of these N threads to N/n different set indexes in a cache bank for the same data structure in a program. It is proven in [12] that two conflict addresses are always different in their least significant bits of tag. We can introduce $Log_2(N/n)$ bits (A_{31}) to XOR with the least $Log_2(N/n)$ significant bits of index bits (A_{21}), as shown in Fig.3.

Now the mapping function of set index can be presented by the $Log_2(N/n) \times Log_2(N/n)$ matrix H', rather than $Log_2(N) \times Log_2(N)$ matrix. The mapping scheme is changed from Equation 1 to Equation 2 as follows.

$$index = A_{31} H_{log2(N/n)xlog2(N/n)} \oplus A_{21};\ interleave = A_4 H_{kxk} \oplus A_1 \tag{2}$$

4 Methodology and Simulation

4.1 Many-Core Simulator Architecture

To combine simplicity of dance-hall [7,16] and scalability of tiled architecture [35], we present a many-core architecture, called as Godson-T [34]. Godson-T is targeted

to applications that are highly parallelizable and require enormous amount of computation. It employs on-chip network and distributed shared memory, as shown in Fig.4. It is a homogenous many-core processor which is integrated with in-order dual-issue processing-cores, and each processing core is based on the MIPS II architecture and clocked at 1GHz. Each core has independent private level-one data and instruction cache, and configurable programmer-controlled Scratch-Pad Memory. It has shared level-two static NUCA cache, synchronization manager and memory controller on chip. It introduces a 2-D mesh as on-chip network to communicate between processing-cores, Scratch-Pad Memory and level-two cache. Applications run on Godson-T via the management of a lightweight runtime system, and the runtime system is responsible for thread creation, dispatch, and joins.

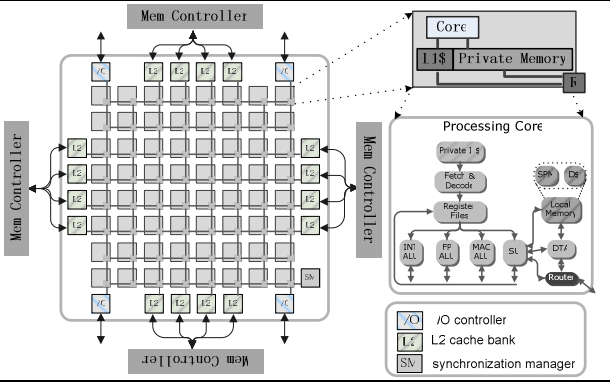


Fig. 4. Godson-T Simulator Architecture

Table 1. Configuration Parameters

Module	Description
processing core	64 processing cores, 8 stage pipeline, in-order 2-issue, 2 ALU&FPU, 1 DIV/MUL, non- preemptive thread execution model
L1 DCache	private, 32KB, 32B/block, 4-way set associative, 1cycle/hit; write through policy, write non-allocate policy
SPM	configurable from L1 data cache; 1 cycle latency
L1 ICache	private, 16KB, 32B/block, 2-way set associative, 1cycle/hit
L2 Cache	shared, total 2MB, 16 banks (128KB/bank), single read & write port per bank, 64B/block, 8-way set associative per bank; out-standing miss support, 4 outstanding cache misses, hit latency 4 cycles
NoC	8x8 2-D mesh, static X-Y routing, 2cycles/hop
Router	2 stage pipeline, 2 virtual channel, 128-bit data bandwidth
Memory Controller	4 controllers, interleaved address mapping, 512-bit data bandwidth, read latency 52 cycles, write latency 32 cycles

Level-two and level-one cache are non-inclusive, which means that the data cached in L1 and L2 can be replaced respectively and will not incur invalidation to each other. The level-two cache is designed as non-blocking cache. Each level-two cache bank can support four outstanding cache misses. Therefore, all sixteen level-two

cache banks can support sixty-four outstanding off-chip load operations. The interleaved banks are located along four sides of the chip die and are shared by all processing cores. All on-chip processing cores access level-two cache through mesh network. The latency of an access from a processing core to different level-two cache banks is different. Every cycle, each memory controller can pump one double word into the chip from the off-chip memory. Since the Godson-T processor is clocked at 1GHz, the maximum off-chip memory bandwidth is 32GB per second.

There are three advantages with the proposed many-core architecture, *i.e.* scalability, lower access latency and larger on-chip capacity [34]. The following evaluation bases on the Godson-T cycle accurate C simulator and the simulation configuration parameters are as Table 1 shows.

4.2 Evaluated Scheme

The traditional XOR-based hash functions [1,11,12,13,26] are symmetric mapping schemes. So their performance improvement on homogeneous many-core architecture is negligible, and we do not present their results. Because the computing complexity and hardware implementation cost of prime-based scheme, we do not compare performance of XORM scheme to prime-based schemes.

At the knowledge of us, XORM is the first to introduce mapping scheme both in interleave bits of cache bank and set index, so we need to evaluate the performance benefit. We implement EE mapping scheme both in interleave bits and set index, called as EE_INT, to compare with XORM scheme. We implement victim cache with different capacity for each cache bank through configuring write back queue of every cache bank. The victim cache with smaller capacity is called VMC(1), and the one with larger capacity is called VMC(4), where number in parentheses means the different configurable grain of write back queue entries. All evaluated schemes are shown as Table 2, where ORIGINAL scheme means the traditional shared cache.

Table 2. Evaluated Scheme

Name	Implementation Scheme	
	Set Index Bits	**Interleave Bits**
ORIGINAL	Modulo-based	Modulo-based
EE	EE-based	Modulo-based
EE_INT	EE-based	EE-based
XORM	XORM-based	XORM-based
VMC(1)	Modulo-based	Modulo-based
VMC(4)	Modulo-based	Modulo-based

4.3 Workloads

We select three kernels of scientific computation, *that is*, matrix multiplication (MM), FFT and LU from Splash2 [24], and pFind workload [28] from bioinformatics. PFind is a bioinformatics algorithm which is for automated peptide and protein identification from tandem mass spectra. PFind is a memory access intensive program, and it can evaluate memory systems abundantly. In addition, matrix multiplication (MM), FFT

and LU decomposition are important algorithm kernels which are used in high performance computing frequently.

The program size of workloads as Table 3 shows, and we statistic performance data during these workloads execution without shared cache pre-warm.

Table 3. Program Size

Workload	Program Size	Optimization	Instructions(10^6)
pFind	32,768 peptides	-O3	547.49
FFT	1,048,576 Complex Doubles		101.8
LU	512x512 Matix 16x16 Element Blocks		618.2
MM	256x256 Doubles Matrix		10.9

4.4 Results

Because XORM mapping scheme is XOR-based, it can be implemented by a series of logical XOR gate circuit, so its computing latency is negligible; on the other hand, it is applied on the last level shared cache on chip, so its negative effect on cache performance is also negligible. In this section, we do not evaluate computing latency increased by XOR-gate directly. We evaluate and compare the overall performance of related schemes from the following aspects.

4.4.1 Effect on Level-Two Cache Miss Rate

We do not introduce prefetching, so the proposed scheme does not influence the number of compulsory cache misses. The effect on shared cache miss rate is shown as Fig.5. The miss rate is normalized to traditional shared cache. It shows that XORM scheme can improve shared cache miss rate for most of the evaluated workloads. For pFind, the Fig.1 shows that the private data cache miss penalty is smaller relatively, so the improvement for total miss rate of shared cache is not obvious. But on average, it can improve the value by about 4%.

4.4.2 Reduction of Shared Cache Conflict Misses

The proposed XORM scheme is dedicated to avoid conflict misses in shared cache. It is shown as Fig.6, and the result is normalized to traditional shared cache. Although it shows in Fig.5 that the proposed scheme has small improvement, it shows that it can reduce conflict misses for all of these evaluated benchmarks, especially for matrix multiplication and pFind. The reason is that we evaluate a matrix with smaller work set, which has less conflict misses. On average, the proposed scheme makes the ratio of conflict miss rate in miss rate of shared cache reduced by about 53%.

4.4.3 Average Load Latency of Shared Cache

The average access latency of level-two cache is shown as Fig.7. The result is normalized to load latency of traditional shared cache. Here, load latency means that the processing cycles elapse from shared cache accepting a load request to the cycle when shared cache sends refilling data to private cache. During the elapsed cycles, there may be a shared cache miss caused by the load request to off-chip memory. On average, it reduces the metric by about 6 cycles, and it is improved by about 5%.

4.4.4 Average Load Latency of Level-One Data Cache

The average load latency includes hit latency and miss penalty of private data cache. The accessing latency from a certain processing core to shared cache banks is different, which depends on number of hops in on-chip network. When employing mapping schemes on interleave bits, it distributes access requests to level-two cache banks evenly. It avoids congestion of on-chip network caused by tree saturation, so the average load latency of level-one cache is decreased. The normalized results normalized to traditional shared cache are shown in Fig.8. On average, XORM scheme makes average load latency of private cache decreases by about 2 cycles, and about 7%.

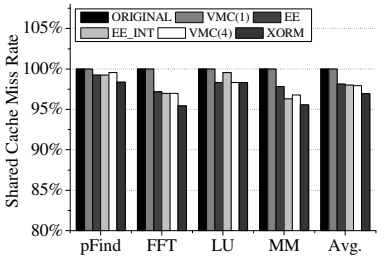


Fig. 5. Effect on L2 Miss Rate

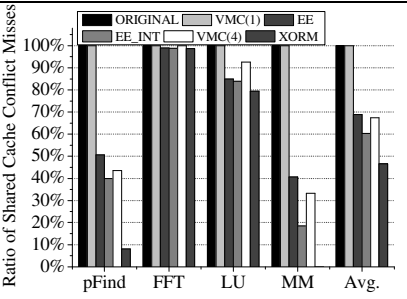


Fig. 6. Shared Cache Conflict Misses

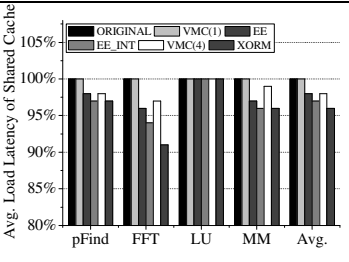


Fig. 7. Avg. Read Latency of L2 Cache

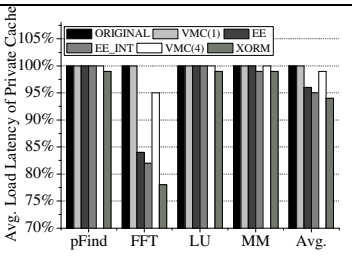


Fig. 8. Average Load Latency of DCache

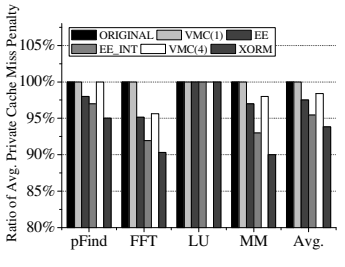


Fig. 9. Avg. L1 Data Cache Miss Penalty

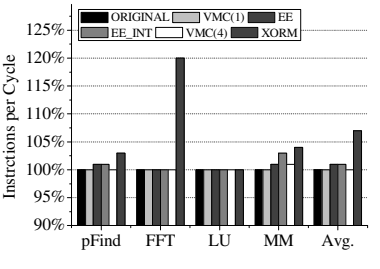


Fig. 10. IPC

4.4.5 Average Miss Penalty of Level-One Cache

The miss penalty of private cache is processing cycles recorded start from a private cache miss occurs to completion of refilling from next-level shared cache. The normalized result is shown in Fig.9. For LU, there are lots of barriers in the non-optimized program, which hamper performance of XORM. But on average, XORM scheme can improve the ratio of average miss penalty of private cache by about 7%.

4.4.6 IPC

The compare result of IPC is shown in Fig.10. The result is normalized to traditional shared cache. It shows that the proposed scheme can improve most of the evaluated benchmarks excluding LU decomposition. The reason is also lots of barriers in LU. On average, the proposed scheme makes IPC improved by about 6%.

In summary, for traditional shared cache with a smaller victim cache VMC(1) for each bank of shared cache, its performance has little improvement. When employing EE mapping scheme on set index, the performance improvement on many-core architecture is not obvious as it in single core processor; but if also employing it on interleave bits, it can improve performance further, because it reduces bank conflict. The result also shows that XORM scheme can avoid most of conflict misses in shared cache of homogeneous many-core processor simulator, and can adapt to load balance when number of threads increases. In addition, although we evaluate XORM scheme with a certain many-core simulator only, the conflict-free properties ensure that its performance in other architectures.

4.5 Implementation Complexity and Cost

The Godson-T many-core processor has been fabricated using the TSMC 130nm ASIC technology and physical compiler by Synopsys Design Compiler. The RAM blocks used in cache (including data bank and tag bank) are generated by Artisan 130nm Memory Compiler. From the elementary area and latency data, RAM modules occupy 82.6% of total area of level-two cache, and occupy 23.3% of total area of whole Godson-T many-core processor.

The XOR computation is logic circuit, and the additional area cost and computing latency are negligible. However, victim cache VMC(1) and VMC(4) occupy 2.5% and 11% of total area of level-two cache respectively. Although larger victim cache has comparable performance improvement with XORM scheme, it seems not valuable to implement a larger victim cache for many-core architecture.

5 Conclusions and Future Work

Since all processing cores share the on-chip cache in many-core processor, as the number of threads executing concurrently increases, the conflict in shared cache increases obviously. How to design a conflict avoiding scheme in many-core architecture with low hardware complexity and implement cost is a valuable studies. In this paper, we contribute in three aspects. Firstly, we investigate cache performance on many-core system and the effectiveness of XOR mapped static NUCA shared cache. Secondly, we propose an XOR-based address mapping scheme, called as XORM, for

many core architecture. Finally, we implement some conflict avoiding schemes on many-core processor simulator, such as victim cache, and evaluate these schemes for various applications. Experimental results show the effectiveness of XORM mapping scheme when compared with other conflict avoiding schemes.

In static NUCA cache, accessing time of a processing core to different level-two cache banks is different, and the more conscious of this difference the mapping scheme would be, the better the performance. But we leave the case as future work. And we will characterize conflict misses in shared memory in detail, and present corresponding schemes of combining the address mapping scheme with cache partitioning scheme in many-core architectures.

Acknowledgments. This paper is supported by the National Natural Science Foundation of China under Grant No.60736012, the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321600, the National High-Tech Research and Development Plan of China under Grant No.2009AA01Z103, the National Science Foundation for Distinguished Young Scholars of China under Grant No.60925009, and the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant No. 60921002.

References

1. González, A., Valero, M., et al.: Eliminating cache conflict misses through XOR-based placement functions. In: ICS'11, pp. 76–83 (1997)
2. Seznec, A.: A Case for Two-Way Skewed Associative Caches. In: ISCA 1993, pp. 169–178 (1993)
3. Agarwal, A., Pudar, S.: Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches. In: ISCA 1993, pp. 179–190 (1993)
4. Beckmann, B.M., Wood, D.A.: Managing Wire Delay in Large Chip-Multiprocessor Caches. In: Micro'37 (2004)
5. Kim, C., Burger, D., Keckler, S.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In: ASPLOS 2002 (2002)
6. Fotland, D.A., et al.: Hardware Design of the First HP Precision Architecture Computers. Hewlet-Packard Journal 38(3), 4–17 (1987)
7. Almasi, G., et al.: Dissecting Cyclops: A Detailed Analysis of a Multithreaded Architecture. ACM SIGARCH Computer Architecture News 31(1), 26–38 (2003)
8. Suh, G.E., Rudolph, L., Devadas, S.: Dynamic Partitioning of Shared Cache Memory. The Journal of Supercomputing 28(1), 7–26 (2004)
9. Pfister, G.F., Norton, V.A.: 'Hot-spot' contention and combining in multistage interconnection networks. IEEE Trans. Comput. C-34, 943–948 (1985)
10. Vandierendonck, H., Manet, P., Legat, J.: Application-specific reconfigurable XOR-indexing to eliminate cache conflict misses. In: DATE 2006, pp. 357–362 (2006)
11. Vandierendonck, H., Bosschere, K.: Constructing Optimal XOR-Functions to Minimize Cache Conflict Misses. In: ARCS 2008, pp. 261–271 (2008)
12. Vandierendonck, H., Bosschere, K.: XOR-based Hash Functions. IEEE Transactions on Computer 54(7) (July 2005)
13. Frailong, J.M., Jalby, W., Lenfant, J.: XOR Schemes: A Flexible Data Organization in Parallel Memories. In: ICPP 1985, pp. 276–283 (1985)

14. Asanovic, K., et al.: The Landscape of Parallel Computing Research: A View from Berkeley, Technical Report No.UCB/EECS-2006-183, December 18 (2006)
15. Olukotun, K., et al.: The Case for a Single-Chip Multiprocessor. In: Proceedings Seventh ASPLOLS VII, Cambridge, MA (October 1996)
16. Seiler, L., Carmean, D., et al.: Larrabee: a Many-Core X86 Architecture for Visual Computing. In: SIGGRAPH 2008, Los Angeles, California (2008)
17. Kharbutli, M., Irwin, K., Solihin, Y., Lee, J.: Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses. In: HPCA 2004 (2004)
18. Kharbutli, M., Solihin, Y., Lee, J.: Eliminating Conflict Misses Using Prime Number-Based Cache Indexing. IEEE Transactions on Computers 54(5) (May 2005)
19. Kandemir, M., Li, F., et al.: A Novel Migratrion-Based NUCA Design for Chip Multiprocessors. In: SC 2008, Austin, Texas (November 2008)
20. Qureshi, M.K., Patt, Y.N.: Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In: MICRO 2006 (2006)
21. Jouppi, N.P.: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. In: ISCA 1990, pp. 364–373 (1990)
22. Gao, Q.S.: The Chinese Remainder Theorem and the Prime Memory System. In: ISCA 1993, pp. 337–340 (1993)
23. Yang, Q.: Introducing a Conflict-free Cache Design. In: Tao, L., Li, T. (eds.) Practical Aspects of Parallel Computing, pp. 19–33. International Academic Publishers (1995)
24. Woo, S.C., Ohara, M., et al.: The SPLASH-2 Programs: Characterization and Methodological Considerations. In: ISCA 1995, pp. 24–36 (1995)
25. Kim, S., Chandra, D., Solihin, Y.: Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In: PACT 2004 (2004)
26. Cho, S., Choi, U., et al.: Design of new XOR-based hash functions for cache memories. Computers and Mathematics with Applications 55, 2005–2011 (2008)
27. Ding, X., Nikolopoulos, D.S., et al.: MESA: reducing cache conflicts by integrating static and run-time methods. In: ISPASS 2006, pp. 189–198 (2006)
28. Fu, Y., Yang, Q., et al.: Exploiting the kernel trick to correlate fragment ions for peptide identification via tandem mass spectrometry. Bioinformatics 20, 1948–1954 (2004)
29. Liu, Z., Li, X.: XOR Storage Schemes for Frequently Used Data Patterns. Journal of Parallel and Distributed Computing 25, 162–173 (1995)
30. Liu, Z., Li, E., Qiao, X.: Technique and mechanism of Address Mapping in Cache (Chinese Patent NO.:ZL97120245.1), November 6 (1997)
31. Zhang, Z., Zhu, Z., Zhang, X.: A Permutation-based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality. In: Micro 2000, pp. 32–41 (2000)
32. Chishti, Z., Powell, M.D., Vijaykumar, T.N.: Optimizing replication, communication, and capacity allocation in CMPs. In: ISCA 2005, pp. 357–368 (2005)
33. Song, F., Liu, Z., et al.: Design of New Hash Mapping Functions. In: IEEE CIT 2009, pp. 45–50 (2009)
34. Fan, D., et al.: Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions. Journal of Computer Science and Technology 24(6), 1061–1073 (2009)
35. Taylor, M.B., et al.: Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In: ISCA-31 (June 2004)