Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 9, Number 1, pages 19–22, January 2020

Hardware Design and Evaluation of Cell Allocation Cache

Tomoaki IKARI
Department of Information Science and Technology, Aichi Prefectural University
Aichi prefecture, Japan
is161095@cis.aichi-pu.ac.jp

Takahiro SASAKI
Department of Information Science and Technology, Aichi Prefectural University
Aichi prefecture, Japan
sasaki@ist.aichi-pu.ac.jp

***Abstract*—Multi-core processors are widely used to improve performance of computer systems. To achieve both high performance and low power consumption, many researches from device level to processor architecture or application software level have been done. This paper focuses to cache system which is one of the significant module consuming large power. To achieve both high performance and low power consumption, Cell Allocation Cache is proposed which is similar to dynamic cache partitioning technique, but it allocates cache spaces called "Cell" which is smaller than a way. Cell Allocation Cache can achieve better performance than normal cache and cache partitioning. The previous research evaluates the performance of the Cell Allocation Cache. However, circuit scale has not been clarified because detailed design is not performed. This paper evaluates circuit scale of Cell Allocation Cache and shows the hardware overhead.**

***Index Terms*—Cache partitioning, shared cache, cell allocation cache, high performance and low power, VLSI design.**

I. Introduction

In recent years, the processors used in from smart phone to high-end computer need high performance and low energy consumption. There are many studies to achieve both high performance and low energy consumption. Multi-core processor is one of the ways to achieve high performance. However, in multi-core processor architecture, memory accesses are increased and each processor uses same entries frequently which cause frequent data confliction than that of single-core processor. Then the problem which cache miss rates increase occurs. Memory access is one of the bottle necks of the processor performance and it is important for high performance to decrease memory accesses or decrease cache miss rates. In order to solve the problem, we propose Cell Allocation Cache [1] which allocates smaller cache spaces called "Cell". The Cell is smaller than a way and dynamically assigns to the processor core. However, Cell Allocation Cache required parameters to optimize cache performance. Reference [2] shows automatic parameter tuning technique and performance for Cell Allocation Cache. However, circuit scale has not been clarified because detailed design is not performed. This paper evaluates circuit scale of Cell Allocation Cache and shows the hardware overhead.

II. Cell Allocation Cache

A. Abstract of Cell Allocation Cache

Cell Allocation Cache is based on the way applied cache partitioning [3] for multi-core processor, but it can control cache with more smaller units. In general, each core competes for shared cache memory which causes performance degradation. Cache partitioning can reduce conflicts by partitioning ways and assign them to each core dynamically. However, there are temporal and spatial localities in the program, accessed memory area is nonuniform. To explain this problem simply, we assume four cores system. If single light-weight process and three multi-threaded heavy process runs simultaneously, normal cache is not aware of working set. Therefore, single light-weight process is robbed cache capacity by heavy process and it degrade performance. Cache partitioning assign one or more ways to each cores, so light-weight process can keep its own cache space. However, as well known, memory access has locality of reference. Therefore, some part of assigned cache space is not used well. Cache partitioning cannot use cache memory effectively if the core does not need much cache memory size. To solve the problem, our Cell Allocation Cache assign smaller cache spaces called 'Cell', into that a way is subdivided. Accoding to [2], Cell Allocation Cache can improvement 2.3 and 2.9 better percent point compared to cache hit ratio of normal and cache partitioning technique.

Fig. 1 shows the outline of Cell Allocation Cache. The Cell is a group that holds some sequential cache lines, and the "Block" is a group of Cells located in the same row. Cell is assigned to one core dynamically. If a core running a program whose working set is large, more Cells are assigned to the core. However, in multi-thread program, each thread may share data. To handle shared data effectively, we introduce two Cell types called "Private Cell" and "Shared Cell". Private Cell is assigned to one specific core. On the other hand, Shared Cell behaves as normal cache which can be replaced from any core. From now, we explain the operations of line replacement. line is a unit used to replace data between the cache and main memory. If cache miss occurred, the cache read the data from main memory or lower hierarchical cache. If both Shared Cell and assigned Private Cell do not exist in the block, the replace

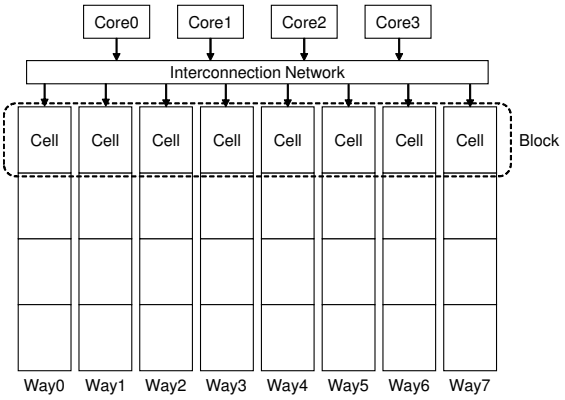

Fig. 1. Cell Allocation Cache.

target is decided using LRU(Least Recently Used). If at least one assigned Private Cell or Shared Cell exists, Cell Allocation Cache decides a replace target from them. If there are some candidates, the target line is decided using LRU from assigned Private Cells and Shared Cells.

B. Management of Cell Types

Initially, cells are assigned to cores evenly. But, at every predefined interval, Cell Allocation Cache re-assigns Private Cell to other core, and/or changes types of Cells from Private to Shared, and vice versa. Thus, the number of Private Cells for the hit ratio lowest core is increased. SharedRatio is shown in Eq(1), and is calculated for every Cells. Cell types are managed based on SharedRatio.

$$\text{SharedRatio} = \frac{\text{Hit count of other core}}{\text{Total hit count}} \quad (1)$$

To calculate SharedRatio for each Cell, we introduce three counters shown in Fig. 2, "Access Counter" which counts total access including cache miss, "Hit Counter" which counts the number of hit and "Hit Other Counter" which counts the number of access from not assigned core and causes cache hit, into each Cells. This approach works well if target programs are multiple single-thread programs or uniform multi-thread program. However, to handle multi-thread program which shared data and single-thread program simultaneously, memory access pattern is unbalanced. To handle shared data effectively, UpperThreshold and LowerThreshold parameters are very important. Details of Cell Allocation Cache parameters are shown in [2]. In the next section, we research the impact of these counters on circuit scale to evaluate Cell Allocation Cache.

III. Impact on circuit scale

A. Additional elements

As described above, the Cell Allocation Cache has three counters but there are other elements we need to implement. Fig. 3 shows the data path of Cell Allocation Cache. Cell Allocation Cache requires adders to increment each counter

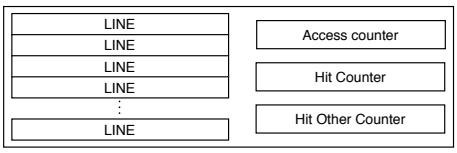

Fig. 2. Counters in each Cell.

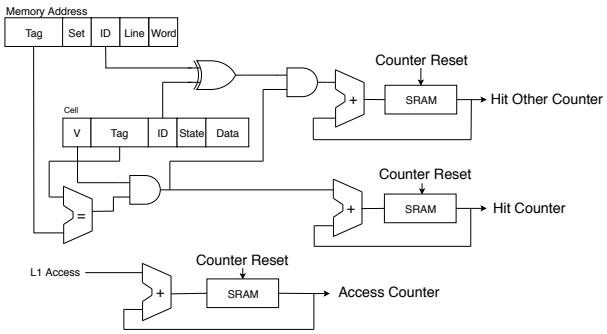

Fig. 3. Data path when cache hit occurs of Cell Allocation Cache.

and bit representing the state of the Cell. "Hit Counter" is increment when cache hit and cache block matches the core specified by the address. ID in Fig. 3 is for identifying the core. "Hit Other Counter" counts all cache hits and "Access Counter" counts all cache accesses. Because [2] evaluates performance using a software simulator, hardware cost is not considered. In this paper, to estimate hardware cost, we design the cache control logic in detail. According to [2], the interval of parameter tuning is 300,000cycles. So we use 19bit counters for each three counter. Fig. 4 shows the mechanism to calculate SharedRatio. When Access Counter value reaches interval parameter, each counter is reset. Table I shows what is added elements per Cell. In addition to the above, phase detection circuit shown in Fig. 5 also impact circuit scale.

TABLE I
Bits added per Cell.

Parts	Amounts
Access Counter	19bit × 1
Hit Counter	19bit × 1
Hit Other Counter	19bit × 1
Adder	3
AND Gate	2
XOR Gate	1
Comparator	2
Divider	1

As another concern, Cell Allocation Cache has possibility of enlargement of circuit scale because it adds three counters in each Cell. Therefore, we estimate how chip area increases using CACTI [5]. According to our estimation results, compared to general cache, the area increasing rate is less than 0.1% when Cell Allocation Cache is implemented on the same condition described in [2]. Thus we can neglect increase of chip area by adding the counters. Whereas, the divider tends

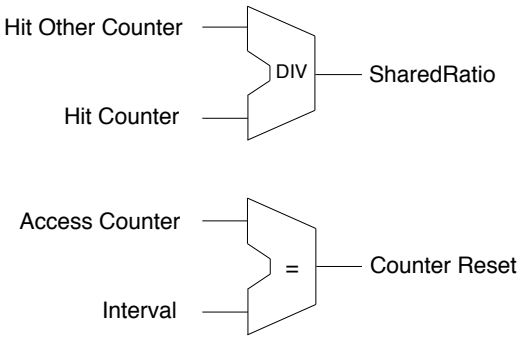

Fig. 4. Mechanism to calculate SharedRatio.

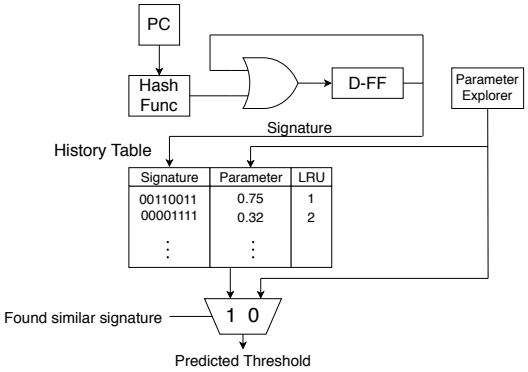

Fig. 5. Adaptive Cell Allocation Cache.

to have a relatively large circuit scale, resulting in an area and power bottleneck. Therefore, we try to implement the control logic without dividers.

B. To reduce overhead

To calculate the SharedRatio, [2] uses a divider. Division is complex calculation, but it can be easily calculated when a divisor is a power of 2. In this case, the result is obtained by shifting the dividend. This paper proposes two methods to use a shift logic instead of a divider. Fig. 6 shows example of simple shift operation. If the divisor is a power of 2, for example, 8, the result is obtained by shifting 3 bits to the right. The number to shift is given by Eq(2).

$$\text{Shift amount} \quad = \quad \log_2(\text{Divisor}) \tag{2}$$

Now, we propose two methods on how to implement approximate divider with shift operation. Two method dividers use the 45nm CMOS process to evaluate the area. As a first method, we propose "approximation method" which approximates the divisor to a power of 2. This method can calculate the SharedRatio at the exact interval. However, this method increases calculation complexity and occurs calculation error occurs in SharedRatio. The circuit scale of this divider is equivalent to 933 NAND gates.

As a second one, we propose "pickup method" which calculates SharedRatio only when the divisor is a power of 2.

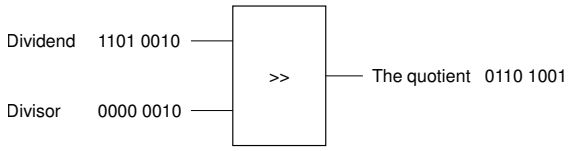

Fig. 6. Example of shift operation.

The calculation complexity and the circuit scale of this method are small. However, the timing to calculate the SharedRatio is inaccurate. The cache controller waits to execute the predefined numbers of load/store instructions. After meeting the condition, the controller waits for the divisor to be a power of 2 to calculate SharedRatio with shift operation. Therefore, timing to calculate the SharedRatio depends on the divisor and memory access frequency. The divider of this method has only 431 gates in terms of the number of equivalent NAND gates.

As shown in section III-A, the effect of counters on the circuit scale is negligible. Similarly, we examine the area increase rate of the circuit with the proposed the divider added. Table II shows the circuit scale and area increase rate of the two methods. The area increase rate was calculated by comparing a normal cache with a Cell Allocation Cache equipped with a counter and an approximate divider. In this survey, the Cell Allocation Cache is assumed to be 8MB, 8way, 128 Cells. As a result, the circuit overhead implemented in the Cell Allocation Cache was negligible. This result can be attributed to the fact that most of the circuit area of the cache is memory cells. However, since peripheral circuits are driven more frequently than memory cells, it is meaningful to reduce the circuit scale. Survey the impact on the performance of the two methods is a challenge for the future.

TABLE II
THE CIRCUIT SCALE OF THE PROPOSED APPROXIMATE DIVIDERS.

Method	Gates*1	Increase rate
approximation	933	1.000132
pickup	431	1.000109

*1 in term of the number of equivalent NAND gates.

IV. CONCLUSION AND FUTURE WORKS

In this paper, we estimate circuit scale of Cell Allocation Cache and we showed units required to implement the circuit. It was found that impact of three counters and approximate divider on circuit scale is slight. Therefore, Cell Allocation Cache overhead is phase detection circuit to calculate SharedRatio.

The following are the future research plans:

- evaluate area / performance tradeoffs with various combinations of parameters,
- design of Cell Allocation Cache by SystemVerilog, and
- layout design with place and route tools.

We also plan to implement our Cell Allocation Cache into FabHetero [4] heterogeneous multi-core processors design automation project.

V. Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15K00074, and VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Synopsys, Inc. Cadence Design Systems, Inc., Rohm Corporation and Toppan Printing Corporation.

References

[1] M. Tone, et al., "Improvement of Cache Performance by Dynamic Control of Partition Allocation," Proceedings of the International Symposium for Sustainability by Engineering at Mie University, pp.13–14, 2016.

[2] T. Sasaki, et al., "Adaptive Cell Allocation Cache using Phase Detection Technique," Proc. of the International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), pp. 1–4, 2019.

[3] G. E. Sue, L. Rudolph, and S. Devadas, "Dynamic Partitioning of Shared Cache Memory," Journal of Supercomputing, vol.28, no.1, pp.7–26, 2004.

[4] T. Okamoto, T. Nakabayashi, T. Sasaki, and T. Kondo, "Fabcache: Cachedesign automation for heterogeneous multi-core processors," in 2013 FirstInternational Symposium on Computing and Networking. IEEE, pp.602-606, 2013.

[5] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp.3–14, 2007.