Efficient Way-based Cache Partitioning for Low-Associativity Cache

Byunghoon Lee and Eui-Young Chung
School of Electrical and Electronic Engineering, Yonsei University
50 Yonsei-ro Seodaemun-gu, Seoul, Korea
E-mail: bh2@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

Abstract: Cache Partitioning is well-known technique to reduce destructive interference among co-running applications in a shared last-level cache (SLLC). Way-based cache partitioning is a popular partitioning scheme due to its simplicity, but it can dramatically reduce associativity of each partition. Also, most SLLC have limited associativity because the higher associativity causes the higher cache access latency and power consumption. Therefore, we present Selective Cache Partitioning (SCP), a novel way-based cache partitioning technique for SLLC with the low associativity. SCP detects outstanding applications causing heavy cache pollution by on-line profiling and isolates them in private partitions. Then, it allocates non-outstanding applications to a shared partition, thereby partitioning SLLC selectively and providing more associativity to applications by sharing the partition. We provide experimental results to show the efficiency of SCP.

Keywords— **cache management, cache partitioning, last-level cache**

1. Introduction

Most chip multiprocessors (CMPs) have a shared last-level cache (SLLC), which is crucial resource on system performance. In CMP, multiple applications with various workloads execute concurrently, and contention of applications in SLLC can cause significant performance degradation due to imbalance of capacity distribution and destructive interaction among applications [1].

Cache partitioning is a viable solution to prevent the performance degradation caused by the cache contention. Way-based cache partitioning is a popular partitioning scheme, which divides SLLC by ways and permit each application to replace cache blocks only within its assigned subset of ways. Way-based cache partitioning has a great advantage of low hardware cost, but it can reduce the associativity of each partition because the associativity of each partition is proportional to its partition size. It means that SLLC must have a large number of ways for way-based cache partitioning to work well [2]. However, a large SLLC have a limited associativity since the higher associativity leads to the higher cache access latency and power consumption. SLLC with 32-way has up to 3.3x the energy per hit and is 32 % slower than SLLC with 4-way [3].

To tackle this limitation, we propose a way-based cache partitioning scheme for a SLLC with the limited associativity, Selective Cache Partitioning (SCP), which allocate private partitions and one or zero shared partition dynamically in contrast with the existing way-based cache partitioning methods allocating private partitions to every running application.

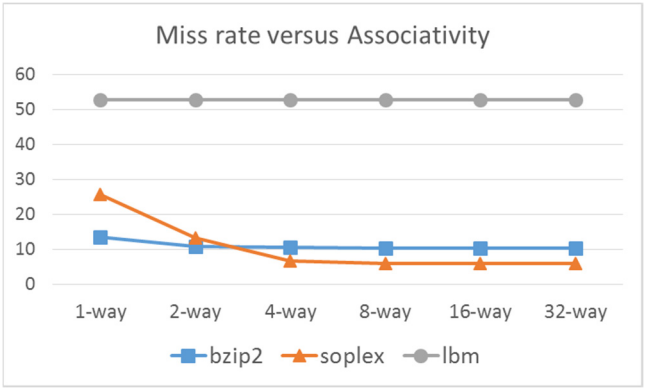

Fig. 1. Miss rate versus associativity of 3 benchmarks

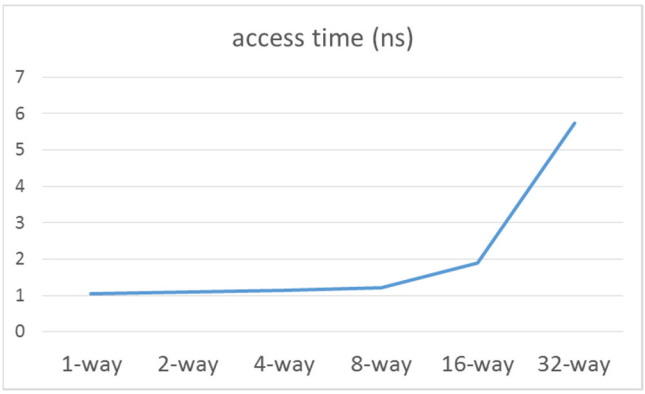

Fig. 2. access time versus associativity in 1MB SLLC

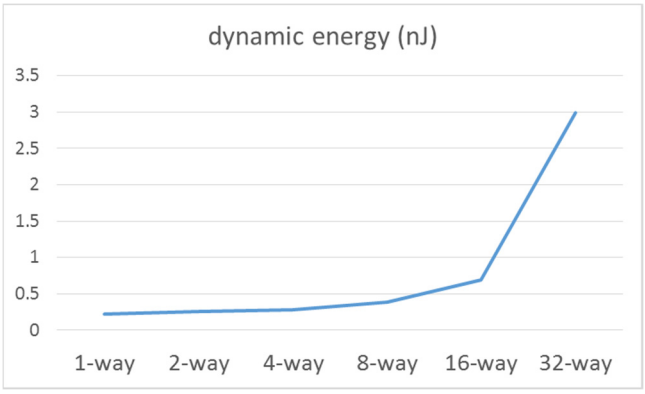

Fig. 3. dynamic energy versus associativity in 1MB SLLC

2. Motivation

Fig. 1 and Fig. 2 show the relationship between associativity and performance of SLLC, and Fig. 3 shows the relationship between associativity and energy consumption of SLLC. Fig. 1 represents the miss rate graph versus associativity with 1MB of SLLC, and three benchmarks of SPEC CPU2006 are used in this graph: *bzip2*, *soplex*, *lbm*. Each benchmark shows different shape of the graph. With *soplex*, associativity has a great impact on the miss rate. Miss rate of *soplex* in SLLC

with 4-way is 19.16 % less than that with 1-way, and the miss rate saturates with more than 8 ways. On the other hand, the SLLC miss rate of *bzip2* and *lbm* are rarely affected by the associativity. Fig. 2 shows access time versus associativity of SLLC whose capacity is 1MB, and Fig. 3 shows dynamic energy consumption. Fig. 2 and Fig. 3 are obtained from CACTI [4] which is cache timing and power model. Both access time and dynamic energy increase exponentially in the given size of SLLC as the associativity increases. Therefore, some benchmarks have lower bound of associativity to maximize performance. Also, the number of associativity of SLLC should be restricted due to exponentially increasing access time and energy consumption.

The conventional cache partitioning schemes focus on multi-programmed workloads and assigns separate partition to each program. It means that way-based cache partitioning assigns at least one way to each program and the associativity may be in short when a lot of applications require SLLC partitions. For example, 8 cores are running concurrently in the SLLC with 16 ways and all cores execute the same benchmark, *soplex*. In the case, each *soplex* needs at least 4 of associativity in SLLC to maximize SLLC performance, but the conventional cache partitioning schemes partition the SLLC as 8 partitions which have 2 ways since the conventional cache partitioning schemes partition the SLLC as the ratio of programs' cache utilizations. Therefore, allocating a separate partition to every program may be inefficient in the SLLC with the limited associativity.

3. Way-based Cache Partitioning Framework

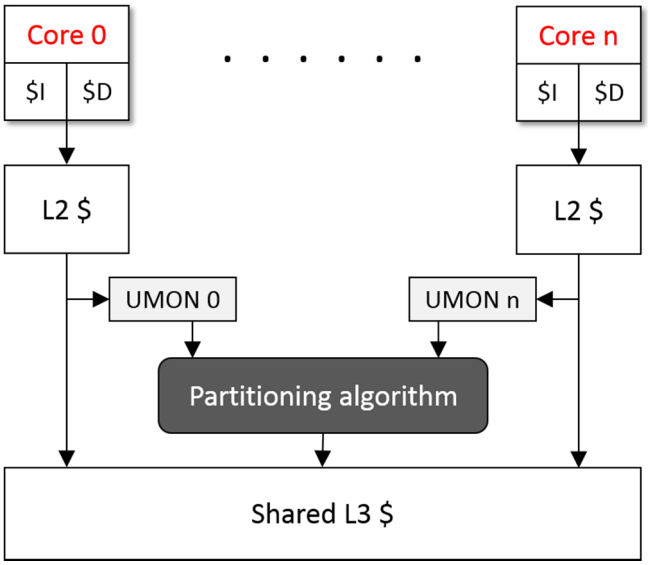

Fig. 4. Way-based cache partitioning framework

Cache partitioning scheme has two main component: partition allocation policy and cache partitioning framework to enforce the determined partition allocation. Also, dynamic cache partitioning changes the partition allocation periodically or when program phase is changed.

In this work, we focus on the partition allocation algorithm to allocate the proper SLLC capacity to each program and implant the proposed partition allocation algorithm to the way-based cache partitioning framework proposed in UCP [5].

In the cache partitioning framework, the partitioning algorithm monitors cache utility of each program from utility monitor (UMON) during the fixed length of interval, then the partitioning algorithm makes new partition allocation at the end of interval.

Fig. 4 shows the cache partitioning framework for SLLC in the three-level of cache hierarchy. In the framework, SLLC accesses from private L2 caches are monitored by UMONs, and each program monitored by a UMON. The partition allocation algorithm decides partition size of each partition.

To enforce the decided partition allocation to SLLC, replacement policy is modified as victim block for the specific program is selected among the blocks belong to the partition for the program.

4. Selective Cache Partitioning

SCP allocates private partition or shared partition to running applications dynamically at starting of each time interval with cache access information profiled during the previous time interval. For this on-line profiling, utility monitors [5] are used in SCP architecture.

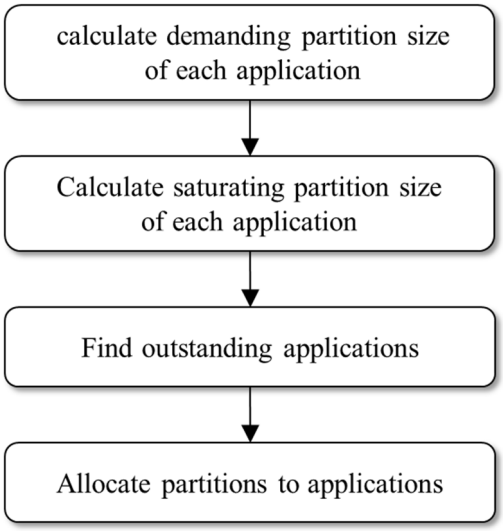

Fig. 5. SCP partition decision procedure

The partition decision of SCP has 4-step procedure shown in Fig. 5. The main idea of the procedure is to isolate applications with poor locality, called outstanding applications in SCP, and to allocate applications with low or medium locality to a shared partition. This is because the main source of cache pollution is the application with poor locality and destructive interaction among applications with the similar level of locality is not significant. Therefore, the shared partition can provide sufficient associativity to applications assigned to it without considerable destructive interference.

The first step of partition decision is to calculate demanding partition size of each application. Demanding partition size means the portion of SLLC demanded by an

application when all applications accesses SLLC concurrently without cache partitioning. In the second step, saturating partition size of each application is calculated using profile information from the utility monitor. The saturating partition size means that the utility of an application is maximized at the size. Then, outstanding applications allocated to private partition are decided with demanding partition size and saturating partition size. If demanding partition size is larger than saturating partition size for an application, the application is selected as outstanding application and its partition size is equal to the saturating partition size. Therefore, SCP can isolate the outstanding applications which demand large cache size unnecessarily. Finally, non-outstanding applications are allocated to the shared partition whose size is calculated by subtracting the sum of all outstanding partition sizes from total associativity of SLLC. Because the number of partitions is less or equal to the number of applications, SCP manager keep the mapping information between application ID and partition ID.

5. Experiments

We use an in-house trace-driven SLLC (L3) simulator. L1 I-cache (32KB, 4-way), D-cache (32KB, 4-way), and L2 cache (512KB, 8-way) are private to the processor core. SLLC (8MB, 8/16/32-way) is shared among 8 cores. Cache line size of all caches is 64B and replacement policy is LRU. We use benchmarks from SPEC CPU2006 suite and combine 8 benchmarks randomly to generate multi-programmed workloads on an 8-core system.

The performance metric used in experiments is IPC throughput [5] defined in Equation 1.

$$IPC\ throughput = \sum_{i=1}^{N} IPC_i \qquad (1)$$

where IPC_i denotes instruction per cycle of the ith application and N denotes the total number of applications.

We compare *SCP* with *noCP* (baseline, SLLC without cache partitioning) and *UCP* (utility-based cache partitioning [5]).

Fig. 6, Fig. 7, and Fig. 8 show IPC throughput of SLLC with 8-way, 16-way, and 32-way, respectively.

In Fig. 6, IPC throughput of SCP is increased by 2.89 % and 10.7 % on average compared with noCP and UCP, respectively. Also, IPC throughput of SCP is increased by up to 3.79 % and 13.26 % compared with noCP and UCP, respectively. In UCP, each application has only 1-way associativity in 8-way SLLC shared by 8 cores because UCP should allocate at least 1 way to each application. Therefore, performance is degraded due to poor associativity despite reduction of inter-application conflict misses. On the other hand, SCP provide more associativity than UCP to each application and also reduce inter-application conflict misses by isolating the applications causing cache pollution.

In Fig. 7, IPC throughput of SCP is increased by 5.0 % and 0.22 % on average compared with noCP and UCP, respectively. In Fig 8, IPC throughput of SCP is increased by 5.8 % and -0.86 % on average compared with noCP and UCP, respectively. This results mean that 16-way is sufficient for way-based cache partitioning.

The experimental results show that SCP is much more effective than UCP in the case of very low associativity cache and the performance of SCP is similar to that of UCP with more than 16 ways.

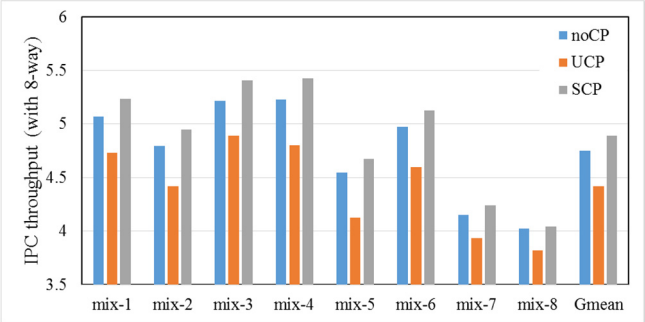

Fig. 6. IPC throughput of SLLC with 8-way

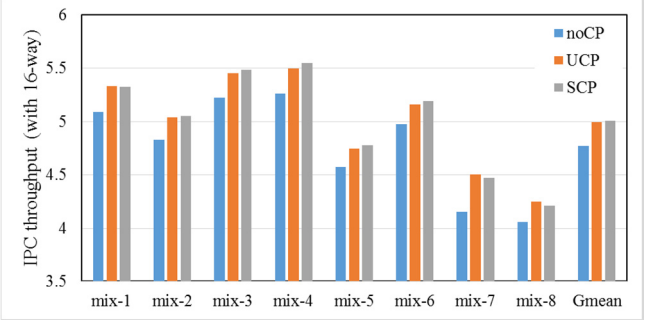

Fig. 7. IPC throughput of SLLC with 16-way

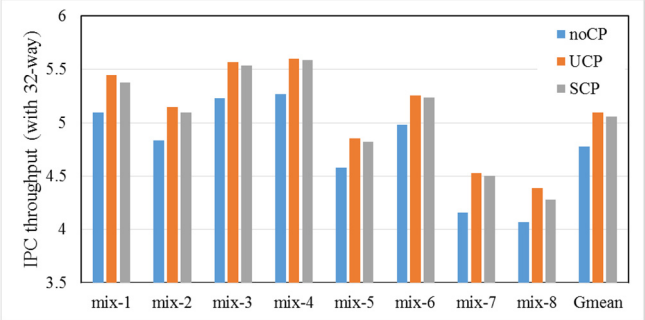

Fig. 8. IPC throughput of SLLC with 32-way

6. Conclusion

In this paper, we present SCP, a novel cache partitioning for SLLC with low associativity. To overcome associativity problem of way-partitioning, SCP allocates private and shared partition to applications. Therefore, applications causing cache pollution can be isolated in the private partitions, and applications allocated to the shared partition can utilize the large cache size as well as the high associativity. The experimental results strongly support the effectiveness of SCP.

Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP, [2016 (R7177-16-0233), Development of Application Program Optimization Tools for High Performance Computing Systems]

References

[1] Li Zhao and et al., “CacheScouts: Fine-Grained Monitoring of Shared Caches in CMP Platforms”, PACT 2007

[2] D. Sanchez and C. Kozyrakis, “Vantage: Scalable and Efficient Fine-Grain Cache Partitioning”, ISCA 2011

[3] D. Sanchez and C. Kozyrakis, “The ZCache: Decoupling Ways and Associativity”, MICRO 2010

[4] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “CACTI 6.0: A Tool to Model Large Caches”, HP Laboratories 2009

[5] M. Qureshi and Y. Patt, “Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches”, MICRO 2006