A New Cache Contention Management Scheme for Multicore Systems

Hoyoung Hwang¹, Hyo-Joong Suh² *

¹ Dept. of Multimedia Engineering, Hansung University, Korea
hyhwang@hansung.ac.kr
² Dept. of Computer Science and Engineering, the Catholic University, Korea
hjsuh@catholic.ac.kr

Abstract. In multicore systems including PCs and smartphones, processes compete for the shared cache memory with each other. We propose a new cache contention management scheme to prevent cache pollution problems in multicore systems. This scheme minimizes unnecessary pollution of private L1 cache by extending the status information of shared L2 cache and grouping the cache access patterns. Simulation results show that the proposed scheme can reduce the cache misses and the side effects invoked by the inclusion property of hierarchical cache memory system significantly.

Keywords: cache contention, multicore, multi-level cache, inclusion property

1 Introduction

In computer systems, cache memory performs a critical role both for the performance and the energy efficiency. Recently, multiprocessor systems with multi-core processors are widely used for mobile smartphone systems as well as the high performance computer systems. Generally, multi-core processors have higher level private cache for each core, and lower level shared cache used by several cores. Fig. 1 shows typical memory hierarchy of a quad-core processor.

Fig. 1. Cache memory hierarchy for multi-processor systems

* Corresponding author
In multicore systems like smartphones, it is common situation to perform the primary processes for communications and other various user applications simultaneously. Each process may be performed on a different core with hierarchical L1 and L2 cache structure as shown in Fig. 1. In that case, a working group of higher priority primary process with less frequent cache requests can be polluted by a lower priority user application processes with frequent cache requests. The processor with frequent cache requests will increase the possession of shared L2 cache and this will affect the cache performance of the primary process since it will increase the invalidation of L1 cache blocks of the primary process.

This kind of cache pollution can be worse to keep the cache coherence [1] and inclusion property [2]. The cache coherency protocol guarantees that the validity of the cache block with latest updated content, and the inclusion property enforces the lower level cache to maintain a superset of its higher level caches [3]. The multi-level cache inclusion property has its benefits but it wastes valuable cache blocks duplicated between L1 and L2 caches. Fig. 2 shows the cache contention and invalidation by the inclusion property which this study addressed to resolve. The L2 cache requests of P1 (2000) may replace the cache blocks used by P2 (3000), and by inclusion property, the duplicated cache block in L1 cache of P2 (3000) will be invalidated. Furthermore, if unbalanced workloads are dispatched on the each core with shared L2 cache, lightweight core is disturbed by invalidation requests to maintain the inclusion property between caches.

![Fig. 2. L1 cache invalidation by the L2 cache contention and inclusion property](image)

In this paper, we proposed a new cache contention management scheme to remove the cache invalidation of the primary process by the inclusion property. This scheme classifies the memory blocks into two groups; a shared data group and a private data group, and allows the inclusion property be relieved for the private data group since the cache coherence can be guaranteed for this group even without the inclusion property. To do this, the first step is finding shared cache blocks which can be writeable among cores. Writeable shared data are keys of the coherency transactions because the other data do not incur major coherency problems. Thus the inclusion property can be limited to the writeable shared data only. For the inclusion control, 5 data types are defined; executable codes, read only data, writable private data, writable shared data, run-time allocated data. The inclusion property can be relieved for the executable codes, read only data, writable private data. The type distinction
can be processed by the compiler except some run-time allocated data. For simplified processing, the run-time data can be handled the same as the writeable shared data.

2 Updated Cache Contention Management

To minimize of the inclusion property overhead, it is needed to check each memory access whether it should keep the inclusion property or not. For this, the inclusion bit was added which has the information related to the address range of memory. The inclusion bits can be attached to the page table/TLB, and it is controlled by the OS. Every virtual address access from the core has to be translated to the physical address with the inclusion attribute. If a translated address does not need to maintain the inclusion property, it can be placed in the shared L2 cache with additional status bit that indicates the data block can be removable without invalidating L1 cache.

Every memory access from the core may invoke a block replacement in the shared L2 cache according to the LRU-like algorithm. If the most aged block is removable, it can be removed without invalidating of the higher level L1 cache. Fig. 3 exhibits this scenario which does not keep the inclusion property for the L1 data block (3000) thus P2 request results in hit.

3 Simulation Results

For the evaluation of the proposed method, the DierroIV [4] cache simulator was used to test the impact of the L1 and shared L2 caches. The DierroIV is a memory trace driven simulator, but does not support the hierarchical shared cache structure of multi-core processor. Thus the simulator was modified through trace control program which handles two memory traces, two L1 caches, and shared L2 cache.

For the memory access traces, CEXP was selected as a small workload. Other 8 programs and 507 writes (writeable shared data) from CEXP are mixed into another
trace which as a memory consuming workload. Table 1 shows the property of detailed workloads from the SPEC’92 benchmark [5].

Table 1. Simulated trace data.

<table>
<thead>
<tr>
<th>Program</th>
<th>Instruction</th>
<th>Data Read</th>
<th>Data Write</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEXP</td>
<td>18041</td>
<td>1452</td>
<td>507</td>
<td>20000</td>
</tr>
<tr>
<td>Mixed trace</td>
<td>42240</td>
<td>4722</td>
<td>6962</td>
<td>53924</td>
</tr>
</tbody>
</table>

Generally, size of the shared L2 cache must be bigger than the sum of its higher level L1 caches for efficiency, thus the simulated sizes of the shared L2 caches are tuned to 4 times and 8 times of the sum of L1 caches sizes. Fig. 4 shows simulation results of conflict miss ratios originated from the non-inclusive types of data that according to the proposed method and the traditional method.

Fig. 4. Conflict miss rate of CEXP executed private L1 cache

The simulation results show the inclusion property incurs most of the conflict misses in the L1 cache, and the proposed method removes these unnecessary cache misses significantly.

4 Conclusion

Multicore processors with multi-level cache architecture are widely used in mobile smart devices. In those devices, the inclusion property may incur the cache contention in the shared cache, and generates unintended invalidation to the higher level cache. If unbalanced workloads dispatched into the cores with shared cache, the cache contention problem may be worse. The proposed method solved this issue by limiting the inclusion property to the writeable shared data, and by adding the inclusion bit in
the page table/TLB and removable bit in the shared L2 cache. The simulation results show that the proposed method reduced the cache miss ratio significantly.

Acknowledgment. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2057967)

References