The Journal of Supercomputing (2019) 75:4519–4528
https://doi.org/10.1007/s11227-019-02763-3

Miss-aware LLC buffer management strategy based on heterogeneous multi-core

Juan Fang[1] · **Xibei Zhang**[1] · **Shijian Liu**[1] · **Zeqing Chang**[1]

Published online: 20 February 2019

Abstract

When multiple processor (CPU) cores and a GPU integrated together on the same chip share the last-level cache (LLC), the competition for LLC is more serious. CPU and GPU have different memory access characteristics, so that they have differences in the sensitivity of LLC capacity. For many CPU applications, a reduced share of the LLC could lead to significant performance degradation. On the contrary, GPU applications have high number of concurrent threads and they can tolerate access latency. Taking into account the GPU program memory latency tolerance characteristics, we propose an LLC buffer management strategy (buffer-for-GPU, BFG) for heterogeneous multi-core. A buffer is added on the side of LLC to filtrate streaming requests of GPU. Cache-insensitive GPU messages directly access to buffer instead of accessing to LLC, thereby filtering the GPU request and freeing up the LLC space for the CPU application. Then, for the different characteristics of CPU and GPU applications, an improved LRU replacement taking into account the recent access time and access frequency of the cache block is adopted. The cache misses-aware algorithm dynamically selects the improved LRU or LRU algorithm to fit the current operating state by comparing the miss rate of cache in buffer so that the performance of the system will be improved significantly.

Keywords Heterogeneous multi-core · LLC · Replacement strategy · Miss-aware

✉ Juan Fang
fangjuan@bjut.edu.cn

Xibei Zhang
s201507147@emails.bjut.edu.cn

Shijian Liu
s201507008@emails.bjut.edu.cn

Zeqing Chang
mermantsing@gmail.com

[1] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China

1 Introduction

Heterogeneous multi-core systems present new challenges by introducing integrated graphics processing units (GPUs) on the same die with CPU cores. In order to reduce the data transfer between memory and video memory and improve the efficiency of the system, researchers proposed the shared last-level cache (LLC) architecture, which is designed to store shared data between the CPU and GPU in the LLC to speed up program execution and reduce memory read and write times. However, CPU and GPU core heterogeneity have also caused competition for LLC space [1]. GPU applications can achieve more data access rates than CPU applications, so most of the available shared cache space will be occupied by GPU applications, leaving only a small amount of space for the CPU application. Compared to the CPU, the number of GPU threads is larger. When a long delay event is encountered, it is possible to switch from one thread to another, and then switch back to the previous thread to continue execution if there is sufficient data level parallelism, which can ignore the memory delay. Therefore, the size of the cache has less impact on the GPU application, and the CPU application is more sensitive than the GPU application on the cache size. From this point of view, CPU applications theoretically require more buffer space than GPU applications and actually get less cache space. Effective management of LLCs and improvement in LLC cache utilization have important implications for optimizing the overall performance of the system.

We propose a miss-aware LLC buffer management strategy based on heterogeneous multi-core. First, we create a buffer with the same structure and location adjacent to the LLC. By analyzing the cache sensitivity of the application, cache-insensitive GPU messages directly access the buffer without accessing the LLC, thereby filtering the GPU requests and freeing up the LLC space for CPU applications. In addition, we have improved LRU replacement strategy and proposed a LFRU (least frequently recently used) replacement strategy, adding a comparison of the frequency of data blocks accessed. According to the level of the buffer missing rate in the interval, we dynamically choose whether to use LRU or LFRU to improve system performance.

2 Related work

The traditional cache replacement strategies include random replacement strategy, FIFO, MRU, LFU, and LRU. The current cache management strategies are mainly divided into two categories, cache partitioning and cache replacement algorithms.

Recent research has employed page coloring mechanism to realize cache partitioning on real system. Zhang et al. [2] proposed a page coloring dynamic cache partitioning mechanism based on malloc allocator. Malloc allocator can be dynamically partitioned among different applications according to partitioning

policy. Only coloring the dynamically allocated pages can remit memory pressure and reduce page copying overhead led by re-coloring compared to all-page coloring. Data prefetching is a well-known technique to hide the memory latency in LLC. Mahmood and Hamid [3] separate the pattern length from the prefetching degree and design an aggressive prefetcher that can generate more addresses with a given pattern length. This adaptive method is suitable for CMP processors where the prefetcher resides in the shared LCC. Li et al. [4] and others used the GPU's memory delay tolerance feature to propose a buffer filter strategy for sharing LLC. This strategy adds a buffer filter to the memory. When the GPU application caches the data or the instruction is missing, it sends the GPU request message to LLC and buffer filter. If there is a hit in LLC, the message is directly returned; if hit in the buffer filter, the data block needs to be sent to LLC; Hao et al. [5] and others considered the memory delay tolerance of GPU programs and proposed a new bypass mechanism. The bypass technology forwards a part of the application's access requests to the memory, leaving cache space for other applications, which can alleviate the application contention for LLC. This method changes the cache coherence protocol MESI_Two_Level so that the GPU application can directly access the memory and separate the CPU and GPU applications from the shared cache access. Results show that when the CPU application is cache sensitive and the GPU application is cache insensitive, the overall performance of the system is significantly improved.

3 Buffer-for-GPU

3.1 Memory access behavior

In general, the number of GPU cores is far more than the number of CPU cores, so the number of GPU application accesses will be much larger than the number of CPU application accesses [6, 7]. The average number of LLC accesses per 1000 clock cycles of benchmarks was statistically calculated. The number of

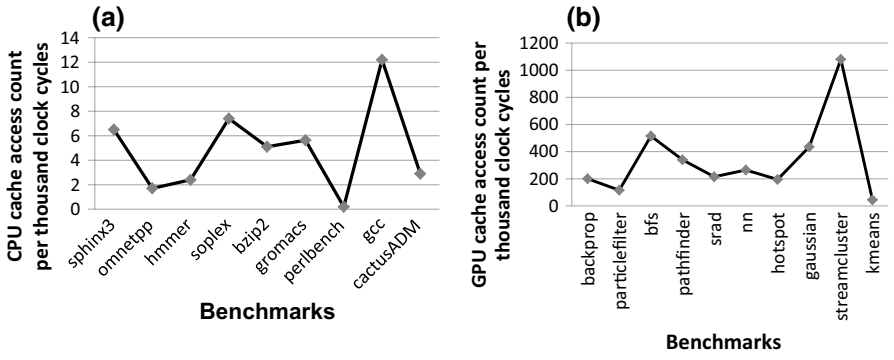

Fig. 1 CPU and GPU cache access count per 1000 clock cycles

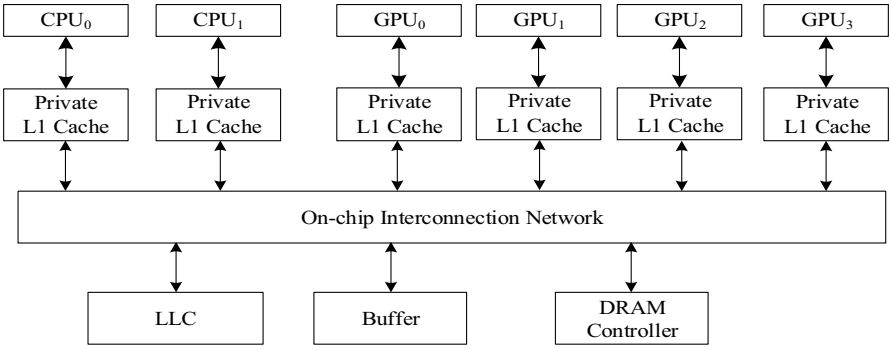

Fig. 2 Heterogeneous architecture with LLC buffer

LLC memory accesses per 1000 clock cycles of the CPU is shown in Fig. 1a and GPU is shown in Fig. 1b.

From the results of Fig. 1a, b, the average number of LLC memory accesses per 1000 clock cycles of the CPU is 5, and the maximum number is 12. The average number of accesses to the GPU for each 1000 clock cycles is 341, and the maximum number is 1100.

It is obvious that the number of GPU accesses is much higher than the number of CPU accesses. According to the rules of the LRU, when CPU and GPU applications share the LLC, LRU implicitly allocates most of the LLC space to applications that have a large number of accesses, whereas CPU applications that have fewer accesses only can share a small part of the shared LLC resources, leading to a decline in overall performance.

3.2 Cache sensitivity

Cache sensitivity indicates how much application performance can benefit from increased cache capacity. For GPU applications, multiple threads form a warp, the unit of scheduling [8, 9]. When a memory access block occurs, there will be a quick context switch without affecting the normal operation of the program. Therefore, the number of concurrent threads of the GPU application can be used as a criterion for judging the cache sensitivity.

The experiment adopts Fermi [10] architecture. The maximum number of warps in each GPU core is 48. We selected a threshold as a criterion for judging whether the GPU application is cache sensitive. By a large number of experiments, set the threshold to 40. Obtain the average number of warps in the current interval while the GPU application is running, and compare it to the threshold. If it is greater than the threshold, it is determined that the GPU application is cache insensitive, otherwise the GPU application is determined to be cache sensitive.

3.3 Implementation of BFG

The CPU and GPU have different access behaviors. The LLC is dominated by thousands of memory accesses from GPU application threads, and the GPU has high memory latency tolerance, resulting in existing cache sharing strategy that is beneficial to the GPU core. CPU cannot reasonably use resources to cause performance degradation.

In response to the GPU application's occupation of LLC, we limit the GPU access to the LLC by creating a GPU-exclusive buffer (buffer) to the memory system. As shown in Fig. 2, we modify the configuration file to create a GPU-exclusive buffer with the same structure as the LLC. It is stipulated that the IDs of LLC and buffer are different, and the size of the buffer can be dynamically modified (the size is initially set to 6% of the LLC size). The memory access request is tracked, and the 0/1 flag is used to distinguish between different core memory access requests. The number of concurrent threads of the GPU application is obtained at the upper-level cache, and the cache sensitivity of the GPU application is determined in comparison with the threshold. When the access request message in the upper-level cache reaches the last-level cache, the message type is first judged in the port, and different address space mapping is performed on the request from the CPU core or the GPU core. Request from the CPU application directly accesses the LLC. For GPU access requests, it is necessary to determine the cache sensitivity of the application. If it is cache insensitive, then access the buffer and do not access the LLC; if it is cache sensitive, directly access the LLC. By this method, the purpose of filtering the GPU stream request is achieved, and the LLC application space is freed for CPU application.

4 Miss-aware LLC buffer management strategy

LRU replacement strategy only considers the recent access information of the data block and does not consider the frequency of access to the data block. When the cache capacity is less than the working set of the program, cache will exhibit jitter phenomenon, which will lead to a decrease in the performance of the computer [11]. Based on BFG, we propose least frequently recently used (LFRU) replacement strategy for different features of CPU applications and GPU applications. LFRU adds a comparison of the frequency of data blocks to be accessed when selecting a sacrificial block. If the access frequency is high, the data block has a high priority; otherwise, the data block has a low priority. If the access frequency of adjacent data blocks is the same, then whether or not the access block is a relatively recent access block is considered. Among the access blocks with the same frequency, the priority value of the most recently accessed data block is higher. When there is no data block to be accessed in the queue, the block with the lowest priority needs to be selected as a victim block and swapped out. Combined with LRU and LFRU, we propose missing-aware LLC buffer management strategy (miss-aware buffer-for-GPU, MBFG). MBFG monitors its performance indicators during program execution based on the state of GPU application execution and dynamically switches the replacement strategy that favor the current operating

state. The performance judging index is measured based on the GPU cache miss rate. At run time, the missing rate threshold and time interval are set to periodically monitor the running status of the application. If the current use of the replacement strategy results in a higher buffer miss rate, it is switched to another alternative strategy.

The L2 cache receives the cache request from the L1, uses the LRU replacement strategy in the initial state, and then monitors the GPU access information at each time interval. The buffer missing rate evaluation index is calculated according to the number of missing GPU accesses and is compared with the previous stage. The buffer missing rate is compared to dynamically switch the replacement strategy that is beneficial to the current operating state, thereby reducing the buffer missing rate.

Missing perception-based dynamic replacement strategy needs to calculate the buffer miss rate in the interval and set the interval to 100 k clock cycles. According to the GPU accesses miss information and GPU access hit information collected during the interval, the GPU miss rate is calculated. The formula is shown in Eq. 1. Among them, cur_miss_rate represents the buffer missing rate, cur_misses represents the GPU access missing number, and cur_hits represents the GPU access hits.

$$\text{cur\_miss\_rate} = \text{cur\_misses} \div \left(\text{cur}_{\text{misses}} + \text{cur\_hits}\right). \tag{1}$$

In order to dynamically compare the missing rate of buffer, the missing rate of the previous stage needs to be calculated. The missing rate in the previous phase is defined as the total missing rate of buffer in consecutive time intervals that used different replacement strategies.

According to the GPU accesses miss information and GPU access hit information collected in the previous stage, the miss rate of the GPU in the previous stage is calculated, and the initial value is set to 80%. The formula is shown in Eq. 2. last_miss_rate represents the missing rate of the previous stage, last_misses represents the GPU missing number, and last_hits represents the GPU access hits of the previous stage.

$$\text{last\_miss\_rate} = \text{last\_misses} \div (\textit{last\_misses} + \text{last\_hits}). \tag{2}$$

The missing rate of the GPU can reflect the utilization efficiency of the GPU for the buffer space in the interval. If it is detected that the missing rate of the GPU is in a relatively high state, it means that the currently used replacement strategy may not be suitable for the current running state. The key of MBFG is to switch replacement algorithms by comparing the missing rate of the buffer. In order to prevent the previous stage from becoming more and more in the process of replacing the policy switching, when the missing rate in the previous stage is greater than the threshold, it is manually modified to 80%. Based on a large number of experimental, we set a missing rate threshold of 90%.

5 Experimental results

We use the gem5-gpu simulator [12] to simulate a heterogeneous processor with 2 CPU cores, 4 GPU cores, each with its own L1 cache, and a shared L2 cache. Each workload consists of a SPEC CPU2006 benchmark and a GPU application selected from Rodinia. This paper uses the instruction per cycle (IPC) executed in the unit

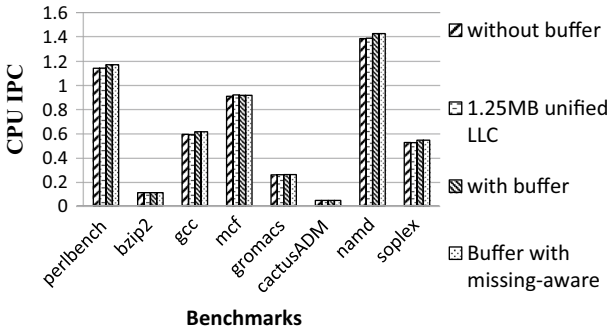

Fig. 3 CPU IPC comparison of LLC buffer management policy based on missing-aware replacement algorithm

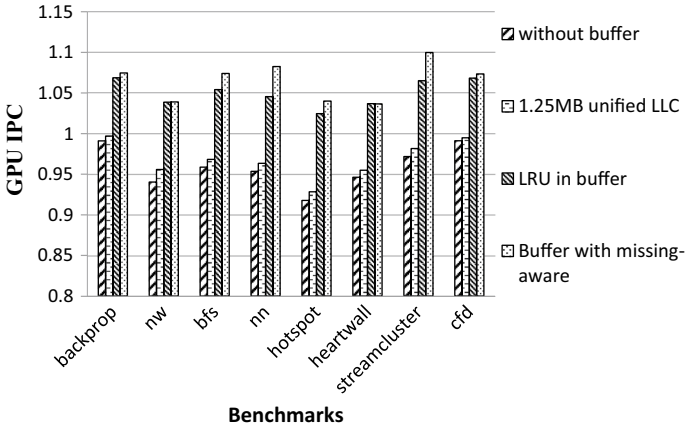

Fig. 4 GPU IPC comparison of BFG

period as the main performance evaluation indicator. The IPC formula is shown in Eq. 3.

$$\mathrm{IPC} = \sum_{i=0}^{n-1} \mathrm{Instructions}_i / \mathrm{Cycles}. \tag{3}$$

The experiment compares the performance measured based on MBFG with the traditional architecture, the 0.25-fold increase in the LLC capacity architecture, and the BFG architecture. CPU IPC comparison of LLC buffer management policy based on missing-aware replacement algorithm is shown in Fig. 3 and GPU IPC in Fig. 4. CPU LLC hit rate is shown in Fig. 5.

Analyzing the experimental data, the average IPC of the BFG architecture compared to the traditional heterogeneous multi-core architecture has increased by 2.48% up to 3.80%. CPU application hit rate increased by an average of 70.45%.

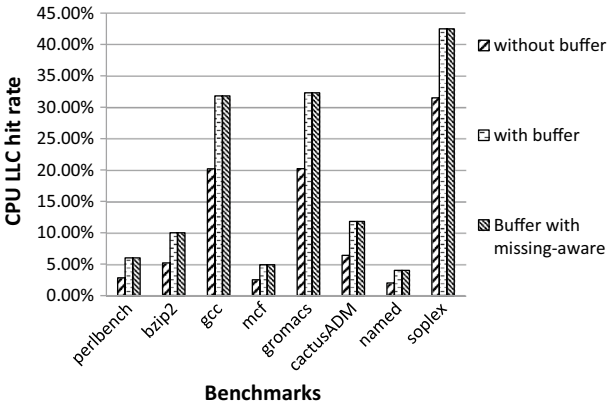

Fig. 5 CPU LLC hit rate comparison of MBFG

Compared with the heterogeneous LLC buffer management mechanism architecture based on multicore, CPU IPC and CPU LLC hit rates remain unchanged. Therefore, using a missing-aware-based dynamic replacement strategy in buffer does not affect the performance of the CPU application. Compared with traditional heterogeneous multi-core architectures, MBFG improves GPU performance by an average of 11.10%. Among them, the GPU performance improvement from the benchmark *nn* in the data mining field is the best and the improvement is as high as 13.52%. Compared with the increase in the 0.25-fold LLC capacity architecture, the GPU performance is improved by 12.36%. The GPU performance of *streamcluster*, which also comes from the data mining field, increased by 13.18%. The GPU performance of the *hotspot* in the field of physical simulation increased by 13.32%, which was higher than the average of the performance improvement. The BFG was used in the buffer architecture. After the strategy, the performance of the test procedures in the above areas has improved.

6 Conclusions

This paper proposes a heterogeneous cache-based LLC buffer management mechanism. This strategy attempts to free up the LLC space for CPU applications by creating GPU-exclusive buffers and achieve separation of memory access by CPU and GPU applications. In addition, this paper applies the dynamic replacement strategy based on the missing perception to the buffer. According to the missing rate of the buffer in the comparison time interval, it can dynamically switch the replacement strategy that is beneficial to the current operating state and further improve the system performance.

The experimental results show that the heterogeneous cache management mechanism based on heterogeneous LLC can effectively relieve CPU and GPU applications' contention for shared cache, ensure the efficiency of CPU utilization for LLC, and improve the performance of CPU applications; buffer performs effective

management, uses a dynamic replacement strategy based on missing awareness, optimizes the buffer management scheme, improves the performance of GPU applications, and improves the overall performance of the system.

Benchmark used in the experiment covers different fields. According to the results, the proposed cache management strategy has different effects on different applications, and the performance of applications in the fields of *data mining* and *physical simulation* is greatly improved. Efficient cache management technology can help data center to perform better. Our proposed cache management strategy can be applied to high-performance data center instead of traditional methods.

Acknowledgements This work is supported by the National Natural Science Foundation of China (Grant No. 61202076), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

References

1. Le TT, Ning R, Zhao D, Wu H, Bayoumi M (2017) Optimizing the heterogeneous network on-chip design in manycore architectures. In: 2017 30th IEEE International System-on-Chip Conference (SOCC), pp 184–189
2. Zhang L, Liu Y, Wang R, Qian D (2014) Lightweight dynamic partitioning for last-level cache of multicore processor on real system. J Supercomput 69(2):547–560
3. Mahmood NT, Hamid SA (2014) Adaptive prefetching using global history buffer in multicore processors. J Supercomput 68(3):1302–1320
4. Li S, Meng J, Yu L, Ma J, Chen T, Wu M (2015) Buffer filter: a last-level cache management policy for CPU-GPGPU heterogeneous system. In: IEEE International Conference on High Performance Computing and Communications IEEE, pp 266–271
5. Fang J, Hao X, Fan Q, Chang Z, Song S (2017) Improving the performance of heterogeneous multicore processors by modifying the cache coherence protocol. In: International Conference on Materials Science AIP Publishing LLC, pp 1–29
6. Ausavarungnirun R, Chang K, Subramanian L, Loh GH, Mutlu O (2012) Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. In: International Symposium on Computer Architecture ACM, pp 416–427
7. Coşkun Ç, Cüneyt FB (2013) Energy and buffer aware application mapping for networks-on-chip with self similar traffic. J Syst Archit 59(10):1364–1374
8. Ausavarungnirun R, Ghose S, Kayiran O, Loh GH, Das CR, Kandemir MT, Multu O (2015) Exploiting inter-warp heterogeneity to improve GPGPU performance. In: International Conference on Parallel Architecture and Compilation IEEE, pp 25–38
9. Yu L, Chen T, Wu M, Liu L (2014) Buffer on last level cache for CPU and GPGPU data sharing. In: 2014 IEEE International Conference on High Performance Computing and Communications, pp 417–420
10. Heinecke A, Klemm M, Bungartz HJ (2012) From GPGPU to many-core: Nvidia fermi and intel many integrated core architecture. Comput Sci Eng 14(2):78–83
11. Lee J, Kim H (2012) TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture. In: IEEE International Symposium on High-Performance Comp Architecture, pp 1–12
12. Power J, Hestness J, Orr MS, Hill MD, Wood DA (2015) gem5-gpu: A Heterogeneous CPU-GPU Simulator. IEEE Comput Archit Lett 14(1):34–36