IP-aware cache partition and replacement scheme for mobile computing devices

Bobae Kim and Hyukjun Lee

Abstract Mobile computing devices employ multiple task-specific IP cores to improve performance and energy. With the main memory shared by all CPU cores and IPs, it becomes a critical bottleneck as increasing multimedia IPs consume significant memory bandwidth whereas the improvement of memory bandwidth lags behind. We propose an IP-aware cache management scheme for the last-level cache to reduce the load on the main-memory, while satisfying QoS requirements of frame-based multimedia applications. Evaluation with CPU applications and 4K/60fps video streaming applications shows that our proposed scheme reduces DRAM bandwidth by 28.62% on average over LRU and completely avoids frame-drops.

Keywords: cache partition, cache replacement, sub-framing

Classification: Integrated circuits

1. Introduction

Meeting various user demands becomes very challenging with the growing real-time multimedia applications in energy-constrained mobile computing devices. Mobile computing devices employ multiple task-specific IP cores to improve performance and energy. However, these IPs require high memory bandwidth. For instance, a streaming video application supporting 4K resolution and 60 frames-per-second requires roughly 3 GBps memory bandwidth, which is 10 times larger than the average memory bandwidth of SPEC 2006 CPU benchmarks. Typically, mobile multimedia applications process sequential frames of data whose size ranges from a half Kbytes (Audio) to 24 Mbytes (Video). Reading and writing a video frame thrash cache and consume significant memory bandwidth.

In [1, 2, 3], sub-framing, partitioning a frame into smaller ones, is proposed to reduce memory traffic. In that work, a memory controller bypasses the memory traffic from IPs to IPs directly without accessing the DRAM memory. However, it requires additional area for flow buffers in the memory controller to capture sub-frames or only entirely re-designed IPs capable of processing fine-grain sub-frames can benefit from the approach. In [4, 5, 6], memory scheduling schemes are proposed for hardware accelerators (IPs). They provide higher priority to memory accesses from accelerators, which are likely to miss given deadlines. These schemes, however, do not reduce the memory traffic and our proposed scheme is complementary to these schemes.

Considering the cache size for the mobile computing device increases with scaling technology (e.g. 8 Mbytes) [7], utilizing last-level cache (LLC) is more flexible and promising. Prior LLC management schemes partition cache to isolate memory traffic with different characteristics and optimize resource allocation [8, 9, 10], or propose the optimal cache line replacement policy by accurately estimating reuse distance [11, 12, 13, 14, 15, 16, 17, 18, 19]. Some propose both partition and replacement scheme [20, 21]. However, these schemes do not consider the characteristics of frame-based IP memory traffic and result in inefficient use of cache. In this study, we propose an IP-aware cache partition and replacement scheme for both full-frame and sub-framed applications in mobile computing environments.

2. IP-aware cache management scheme for IP-rich mobile computing devices

2.1 IP-aware cache partition

As shown in Fig. 1, frame data flow through IPs in streaming video applications and two neighboring IPs have a master(write)-slave(read) relationship. In the proposed scheme, LLC is shared by CPU cores and IPs and should be partitioned to avoid CPU data being thrashed by streaming data. For the cache partition, we divide LLC into the CPU-only and CPU-IP shared region. The CPU only region maps CPU data with high temporal locality and the CPU-IP shared region maps CPU data with low tem-
poral locality or IP data. To guarantee good performance for both CPU and IP memory traffic, we determine the optimal partition sizes for the CPU-only and CPU-IP shared region using Algorithm 1.

Algorithm 1 Cache partition algorithm

for each $i \in \text{leader.set}$ do
  $\minCount = \min(\minCount, \text{missCounter}[i])$
  end for

$MC.\_\text{Threshold} = \minCount \times \alpha$
for each $i \in \text{leader.set}$ do
  if $\text{missCounter}[i] < MC.\_\text{Threshold}$ then
    $\text{SharedPartSize} = \text{CPU\_IP\_Shared\_Size}[i]$
  end if
end for
return $\text{SharedPartSize}$

For partitioning, we use a dynamic set sampling scheme proposed in [22, 23]. In this scheme, several sets in the cache are randomly sampled and become a leader set. Each set in the leader set maintains a miss count measuring cache misses for a different cache partition size. In the 16-way set-associative cache, the leader set maintains the cache miss counters for 2:14, 4:12, 6:10, 8:8, 10:6, 12:4 and 14:2 partitions where $N:M$ represents the number of ways allocated for the CPU-only and CPU-IP shared region. We use a CPU bit to differentiate two regions (CPU-only region:1, CPU-IP shared region:0). We allocate as many ways as possible to the CPU-IP shared partition while still providing good CPU performance by choosing a slightly larger number than $\minCount$ for $MC.\_\text{Threshold}$ ($\alpha$ is set to 1.05 in experiments).

2.2 IP-aware cache replacement

In addition to the partitioning scheme, we propose new insertion, promotion, and victim selection policy. Fig. 2 shows the flowchart for the insertion/promotion/victim selection policy of CPU and IP memory accesses.

Insertion and promotion: Upon a cache miss, a new IP block\(^1\) is inserted into the CPU-IP shared region. Unlike memory accesses from CPU, IP memory accesses have long reuse distance, causing them to be evicted before being accessed again. To avoid this problem, we use a protection bit for each block. When an IP block is inserted, its protection bit is set. In applications using master-slave IPs, an IP block is referenced only once after it is allocated. Thus, once an IP block is hit, its protection bit is reset so that it can be evicted in the future. On the other hand, a new CPU block upon a cache miss is initially inserted into the CPU-IP shared region assuming the block has low temporal locality. When the CPU block allocated in the CPU-IP shared region is accessed again (a hit), it is promoted to the CPU-only region. This promotion set the CPU bit of the block to 1 as it was set to 0 upon an insertion and set RRPV\(^2\) to 2.

Victim selection: To choose a victim block for replacement, we first identify a set from which we choose a victim and compare the predicted and current CPU-IP shared region size for the set\(^3\). For a CPU memory reference, if the predicted size is bigger than the current size of the shared region, we choose a victim from the CPU-only region and set its CPU bit to 0, which reduces the size of the CPU-only region. Otherwise, we choose a victim block from the CPU-IP shared region as long as some blocks in the CPU-IP shared region are not protected. If all blocks are protected, a victim block is chosen from the CPU-only region. For an IP memory reference, if the predicted size is bigger than the current size of the shared region, a victim is chosen from the CPU-only region. Otherwise, a victim is chosen from unprotected shared-region blocks. If all blocks are protected, the missed IP block is not allocated in the cache. This is called bypass.

3. Evaluation

3.1 Evaluation methodology

Table I shows the system architecture parameters for the simulation. We implement realistic IP models [24, 25, 26] in GEMS [27]. Details of IP models are described in Table I. We use ARM processor cores for the processor model [28]. Table II shows the workloads used in the

\(^1\)We call a block observing a master-slave relationship between two IPs an IP block.

\(^2\)RRPV is a two-bit vector used in RRIP [11] where the value 0 and 3 represent a way in MRU and LRU respectively.

\(^3\)Predicted size refers to the size of a CPU-IP shared region that is computed as $\text{SharedPartSize}$ in Algorithm 1.
simulations. Each workload selects three benchmarks from SPEC CPU 2006 benchmark suites [30]. They are sorted based on Miss Per Kilo Instructions (MPKI) in a descending order and are run with a 4 K/60 fps streaming video application. We measure performance, memory bandwidth and energy, and IP memory access latency for LRU, RRIP [11], and the proposed scheme.

3.2 Results

Fig. 3(a) and (b) show the bandwidth reduction and energy reduction respectively. Results are normalized to LRU, which is a baseline cache management scheme. Dynamic partition without sub-framing (FullFrame) and with sub-framing (SubFrame) reduce the memory bandwidth by 4.43% and 28.62% respectively over LRU on average. With sub-framing, the 24 MB frame is partitioned into sub-frames of 0.78 MB, which are efficiently mapped on the cache resulting in large memory bandwidth reduction. Similarly, energy is reduced by 0.79% and 16.55% on average in dynamic partition with FullFrame and SubFrame respectively. RRIP requires 45.69% more bandwidth and 28.54% more energy than Dynamic partition with SubFrame as RRIP captures IP traffic for longer than necessary and thrashes LLC.

With the sub-frame size of 0.78 MB, the proposed scheme does not have any frame drops whereas LRU has 21.33% of frames dropped as shown in Fig. 4(b). For RRIP, all frames do not meet the timing requirements and we omit the result from Fig. 4(b).

In addition, instructions-per-cycle (IPC) of the proposed scheme (Dynamic partition with sub-framing) is larger than LRU and RRIP by 28.72% and 39.25%, respectively. Both LRU and RRIP suffer from interference from IP traffic. Unlike our proposed scheme, both LRU and RRIP cannot isolate IP traffic efficiently, capture IP frames in cache for longer than necessary, and thrash LLC, leading to lower performance.

### Table 1. System architecture parameters

| Component     | Specifications                                      |
|---------------|-----------------------------------------------------|
| Processor     | ARM, 4-core Out-of-Order processor @ 2 GHz          |
| L1 Cache      | private, 4-way, 32 KB, 2-cycle data access          |
| L2 Cache      | shared, 16-way, 4 MB, 20-cycle data access          |
| Memory        | 2 GB LPDDR3-1600 [29], 1 channel, 8 banks, FR-FIFS |
| Video Decoder | @ 400 MHz, Video Frame: 4 K (3840 x 2160), 32 KB Input/Output Buffer, Coding ratio 1:32, compute time: 12 cycles per cache line (64 B) |
| Display Controller | @ 400 MHz, Frame Rate: 60 fps, 32 KB Input/Output Buffer, displaying time: 1 cycles per cache line (64 B) |
| Network Module| @ 400 MHz, Downloading: 64 Bytes/cycle 32 KB Input/Output Buffer |

### Table II. Mixed workloads (three from SPEC CPU 2006 + one frame-based streaming application)

| Mixed Workload | Description                                      |
|----------------|--------------------------------------------------|
| Workload1      | milc+libquantum+jjeng+4 K Video streaming        |
| Workload2      | lbm+astar+perlbench+4 K Video streaming          |
| Workload3      | wrf+perlbench+hmmer+4 K Video streaming           |
| Workload4      | astar+jjeng+bwaves+4 K Video streaming            |
| Workload5      | calculix+bzip2+povray+4 K Video streaming         |

![Fig. 3](image)

(a) DRAM bandwidth reduction (b) Energy reduction

![Fig. 4](image)

(a) Distribution for sorted frame completion time (b) Percentage of frames meeting the deadline
3.3 Hardware cost
Implementing LRU for 16 ways requires four bits for each block to record the access order. In the proposed scheme, we use two bits per block for RRIP in addition to two bits for the CPU and protection bit. Thus hardware cost is same as LRU in terms of additional bits per block.

4. Conclusion
We propose a cache partition and replacement scheme for IP-rich mobile computing platform targeting for frame-based multimedia applications, which successfully reuses sub-frame data while guaranteeing CPU performance and results in bandwidth and energy reduction by 28.62% and 16.55% over LRU respectively.

References
[1] P. Yedlapalli, et al.: “Short-circuiting memory traffic in handheld platforms,” Proc. of the IEEE/ACM Symposium on Microarchitecture (2014) 166 (DOI: 10.1109/MICRO.2014.60).
[2] N. Nachiappan, et al.: “Domain knowledge based energy management in handhelds,” Proc. of IEEE International Symposium on High Performance Computer Architecture (2015) (DOI: 10.1109/HPCA.2015.7056029).
[3] N. Nachiappan, et al.: “VIP: Virtualizing IP chains on handheld platforms,” ACM/IEEE 42nd Annual International Symposium on Computer Architecture (2015) 655 (DOI: 10.1145/2749469.2750382).
[4] H. Usui, et al.: “DASH: Deadline-aware high-performance memory scheduler for heterogeneous systems with hardware accelerators,” ACM Trans. Archit. Code Optim. 12 (2016) 65 (DOI: 10.1145/2847255).
[5] Y.-J. Lin, et al.: “Hierarchical memory scheduling for multimedia MPPSoCs,” Proc. of the International Conference on Computer-Aided Design (2010) 190 (DOI: 10.1109/ICCAD.2010.5654145).
[6] J.-K. Kim and H.-J. Lee: “IP deadline-aware bank partition scheme for memory scheduler in mobile platform,” Proc. of International Conference on Consumer Electronics-Asia (2016) (DOI: 10.1109/ICCE-Asia.2016.7804731).
[7] A. Frumusanu: “The iPhone XS & XS max review: Unveiling the silicon secrets,” https://www.anandtech.com/show/13392/the-iphone-xs-xs-max-review-unveiling-the-silicon-secrets/2 (2018).
[8] K. Huang, et al.: “Curve fitting based shared cache partitioning scheme for energy saving,” IEICE Electron. Express 15 (2018) 20180886 (DOI: 10.1587/exle.15.20180886).
[9] X. Zhang, et al.: “Improving cache partitioning algorithms for pseudo-LRU policies,” IEICE Trans. Inf. & Syst. E96-D (2013) 2514 (DOI: 10.1587/transinf.E96.D.2514).
[10] S. Khan, et al.: “Decoupled dynamic cache segmentation,” Proc. of IEEE International Symposium on High Performance Computer Architecture (2012) (DOI: 10.1109/HPCA.2012.6169030).
[11] A. Jaleel, et al.: “High performance cache replacement using reference interval prediction,” Proc. of the International Symposium on Computer Architecture (2010) 60 (DOI: 10.1145/1815961.1815971).
[12] H. Lim, et al.: “A cache replacement policy to reduce cache miss rate for multiprocessor architecture,” IEICE Electron. Express 7 (2010) 850 (DOI: 10.1587/exle.7.850).
[13] D.-J. Bang, et al.: “An adaptive cache replacement policy based on fine-grain reusability monitor,” IEICE Electron. Express 15 (2018) 20171099 (DOI: 10.1587/exle.14.20171099).
[14] J. Gaut, et al.: “Bypass and insertion algorithms for exclusive last-level caches,” Proc. of the 38th Annual International Symposium on Computer Architecture 39 (2011) 81 (DOI: 10.1145/2000064.2000075).
[15] M. Qureshi, et al.: “A case for MLP-aware cache replacement,” Proc. of the International Symposium on Low Power Electronics and Design (2006) 167 (DOI: 10.1109/ISLPED.2006.5).
[16] J. T. Robinson and M. V. Devarakonda: “Data cache management using frequency-based replacement,” Proc. of the ACM Conference on Measurement and Modeling Computer Systems 18 (1990) 134 (DOI: 10.1145/98457.98523).
[17] V. Seshadri, et al.: “The evicted-address filter: A unified mechanism to address both cache pollution and thrashing,” Proc. of International Conference on Parallel Architectures and Compilation Techniques (2012) 355.
[18] W. A. Wong and J.-L. Baer: “Modified LRU policies for improving second-level cache behavior,” Proc. of High-Performance Computer Architecture (2000) 49 (DOI: 10.1109/HPCA.2000.824338).
[19] S. Das, et al.: “Reuse distance-based probabilistic cache replacement,” ACM Trans. Archit. Code Optim. 12 (2016) 33 (DOI: 10.1145/2818374).
[20] Y. Xie, et al.: “Promotion/insertion pseudo-partitioning of multicore shared caches,” SIGARCH Comput. Archit. News 37 (2009) 174 (DOI: 10.1145/1555815.1555778).
[21] N. Duong, et al.: “Improving cache management policies using dynamic reuse distances,” Proc. of the IEEE/ACM International Symposium on Microarchitecture (2012) 389 (DOI: 10.1109/MICRO.2012.43).
[22] M. K. Qureshi, et al.: “Adaptive insertion policies for high performance caching,” Proc. of International Symposium on Computer Architecture (2007) 381 (DOI: 10.1145/1273440.1250709).
[23] A. Jaleel, et al.: “Adaptive insertion policies for managing shared caches,” Proc. of the International Conference on Parallel Architectures and Compiler Techniques (2008).
[24] M. Roitzsch: “Slice-balancing H.264 video encoding for improved scalability of multicore decoding,” Proc. of the ACM & IEEE International Conference on Embedded Software (2007) 269 (DOI: 10.1145/1289927.1289969).
[25] B. Juurlink, et al.: “Scalable Parallel Programming Applied to H.264/AVC Decoding” (Springer Science & Business Media, 2012).
[26] J. Kim, et al.: “An H.264/AVC decoder with reduced external memory access for motion compensation,” IEICE Trans. Inf. & Syst. E94-D (2011) 798 (DOI: 10.1587/transinf.E94.D.798).
[27] N. Binkert, et al.: “The gem5 simulator,” SIGARCH Comput. Archit. News 39 (2011) 1 (DOI: 10.1145/2024716.2024718).
[28] ARM, Research Starter Kit (2017) http://gem5.org/wiki/images/c/cf/Summit2017_starterkit.pdf.
[29] S. Dumas: “Mobile memory forum: LPDDR3 and WideIO,” JEDEC mobile forum 201 (2011).
[30] J. Henning: “Spec cpu2006 benchmark descriptions,” SIGARCH Comput. Archit. News 34 (2006) 1 (DOI: 10.1145/1186736.1186737).