Energy-Efficient Cache Design in Emerging Mobile Platforms: The Implications and Optimizations

Kaige Yan and Xin Fu
Department of Electrical and Computer Engineering
University of Houston, Houston, TX, 77004, USA
kyan@uh.edu and xfu8@central.uh.edu

***Abstract*—Mobile devices are quickly becoming the most widely used processors in consumer devices. Since their major power supply is battery, the energy-efficient computing is highly desired. In this paper, we focus on the energy-efficient cache design in emerging mobile platforms. We observe that more than 40% of L2 cache accesses are OS kernel accesses in interactive smartphone applications. Such frequent kernel accesses cause serious interferences between the user and kernel blocks in the L2 cache, leading to the unnecessary block replacements and high L2 cache miss rate. We propose to partition the L2 cache into two separate segments which can only be accessed by the user code and kernel code, respectively. Meanwhile, the overall size of the two segments is shrunk, which greatly reduces the energy consumption by 15% while still maintains the similar cache miss rate. We further find completely different access behaviors between the two separated kernel and user segments in our novel L2 cache design, and explore the multi-retention STT-RAM based user and kernel segments to maximize the cache energy savings. The experimental results show that our techniques significantly reduce the cache energy consumption (e.g. 75%) with only 2% performance loss in emerging smartphones.**

I. Introduction

In recent years, the use of mobile devices, especially smart phones, has experienced an explosive growth. According to Gartner [6], the shipment of mobile devices has reached astonishing 2.4 billion units compared to the 300 million units of traditional PC. Meanwhile, a rich variety of applications have been designed for mobile devices, such as web browser, game, video processing, and so on. There were more than 80 billion downloads of mobile applications in 2013 [6]. There is no doubt that mobile platforms have become the primary customer computing platforms, which brings urgent need for computer architecture and system researcher to design the best hardware optimized for mobile usage.

Mobile devices are generally powered by a finite-capacity energy source - a battery, whose size and capacity are quite limited due to the restricted physical size of the mobile devices. Therefore, energy consumption is undoubtedly the most important factor in modern mobile system design, and the energy-efficient computing (i.e., achieving substantial energy savings without degrading the performance) becomes one of the key challenges faced by the computer architects and system designers. There have been several studies observing that the device screen and the processor are the two major energy-hungry structures [12] [4]. The processors in emerging mobile platforms are growing sophisticated to satisfy various mobile application requirements, leading to the increasing energy consumption. Especially, the processor consumes more than 50% of the total energy capacity of the mobile device when the brightness of the screen drops to 25% [4]. It has been well known that the on-chip caches (especially the last level cache) consume a significant fraction of the total processor's energy budget. This has motivated us to explore the energy-efficient cache design, which is critical to achieve the optimal energy savings in modern mobile platforms.

Different from the SPEC benchmarks, smartphone applications are generally user interactive applications, which involve rich GUI display, internet access, file reading, and so on. These operations will generate a large amount of OS kernel accesses. Gutierrez et al. [7] have observed that the smartphone applications can spend up to 15% of their execution time in OS code while that fraction is $< 0.01\%$ in the SPEC benchmarks. We observe that such frequent kernel accesses cause the intense interferences between the kernel and user space accesses in the L2 cache, leading to the unnecessary block replacements and high L2 cache miss rate. In this study, we propose to partition the L2 cache into two separate segments which can only be accessed by the kernel and user code, respectively, to alleviate the interferences. By doing this, we are able to shrink the overall size of the two segments, which reduces both dynamic and leakage energy consumption by around 15% in the L2 cache while still maintain the cache miss rate.

With its strong capability in reducing the leakage energy consumption, the non-volatile memory (e.g., spin-transfer torque RAM (STT-RAM)) has been increasingly introduced into the mobile processors [14]. Blindly building the STT-RAM based L2 cache could cause considerable dynamic power overhead due to the high energy consumption during the STT-RAM write operations. Relaxing the data-retention time reduces the STT-RAM write energy but requires the periodical refreshing, thus more energy for the long lifetime blocks.

Excitingly, we further observe completely different access behaviors between the two separated kernel and user segments in our L2 cache design. For instance, there are much fewer write operations and cache misses in the kernel segment, and cache blocks there typically exhibit much longer lifetime as compared to the user segment. All these make the multi-retention STT-RAM [13] as a perfect candidate to further maximize the energy savings in the L2 cache. We thus propose

978-3-9815370-4-8/DATE15/©2015 EDAA

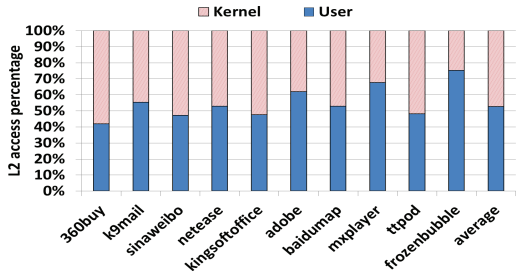


Fig. 1: The percentage of the user and the kernel accesses in the L2 cache.

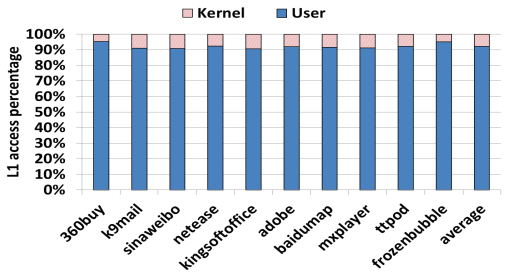


Fig. 2: The percentage of the user and the kernel accesses in the L1 cache.

to adopt the long-retention (short-retention) STT-RAM into the kernel (user) segment. The short-retention STT-RAM effectively mitigates the write energy in the user segment with negligible refresh overhead, while the long-retention STT-RAM successfully preserves the data for the kernel segment with quite few write operations. Our technique is able to further reduce the energy by 60%, thus leading to overall 75% energy savings in L2 compared to the default design, with only 2% performance loss.

To our best knowledge, this is the first work to investigate the interferences between the OS kernel and user accesses in the L2 cache of the emerging mobile platforms, and more importantly, partition the user and kernel space into two segments and intelligently match the multi-retention STT-RAM with the different kernel and user access patterns for the optimal energy savings.

II. The Partition of the User and the Kernel Space in the L2 Cache

A. Observation: The Strong Interferences Between The User and The Kernel Accesses in The L2 Cache

As described in Section I, mobile applications usually spend much more time on system code as compared to the SPEC benchmarks. We conduct a deeper investigation on this interesting characteristic. In Fig. 1, we profile the percentages of the user and kernel accesses (including both cache hit and miss) in the L2 cache when running a set of mobile benchmarks. Detailed experimental setup will be discussed in Section IV. In the Android system, accesses to the shared library are considered as the user accesses. Therefore, each L2 cache access is the access either to the user or to the kernel space. Surprisingly, there are tremendous amount of kernel accesses, which account for almost 50% of the overall L2 accesses on average across the investigated benchmarks.

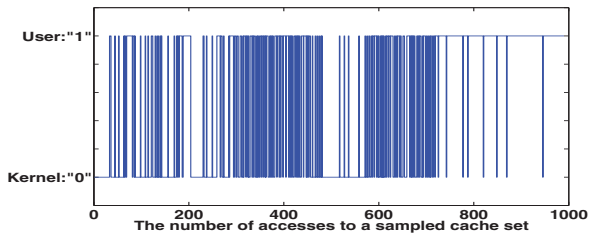
We also profile the kernel and user access percentages in the L1 cache. As shown in Fig. 2, the kernel accesses only contribute to $<$10% of the L1 cache accesses, which do match with the observation made in [7] that mobile applications generally spend 10% of the execution time in system codes. The completely different kernel access percentages in the L1 and L2 caches are caused by the high L1 instruction miss rate and low L1 data miss rate [7] [8]. Mobile applications often heavily use shared libraries and system codes, leading to a large instruction working set and high instruction L1 cache miss rate [7] [8] [3]. On the other hand, the L1 data miss rate keeps relatively low because the data access patterns in mobile applications exhibit good locality in the L1 cache [7] [8]. Therefore, the instruction references account for a high percentage in the L2 cache accesses. Since all the user data is in the user space, the user accesses contains relatively less instruction references than the kernel. So the kernel access percentage increases significantly in the L2 cache as compared to the L1 cache.

As another interesting observation, we find that the frequent kernel accesses generally lead to the frequent switches between the kernel and user accesses in each L2 cache set. Fig. 3 shows 1000 continuous accesses in a sampled L2 cache set for both 360buy and sjeng, which are the representative benchmarks from the mobile benchmark suites (detailed in Section IV) and the SPEC, respectively. The X-axis describes the number of cache accesses to the set, and the Y-axis shows the corresponding kernel or user space that the access belongs to. In Fig. 3, the Y value of a kernel (user) access will be low (high). We further use a line to go through all the Y values from the first to the last access for a better understanding of the kernel and user access switches. It is obvious that 360buy includes much more switches as compared to sjeng.

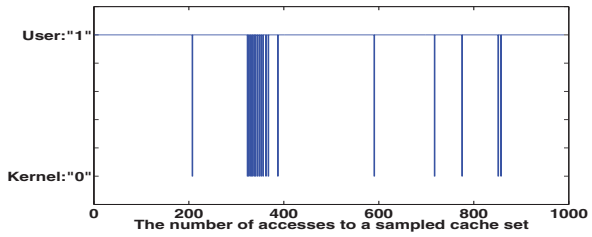
Unfortunately, such frequent switches inevitably cause the strong interferences between the user and kernel blocks in the L2 cache. As Fig. 3(a) shows, there are usually quite few accesses (e.g. 4) between two switches in a cache set. It is highly possible that a useful kernel block that will be accessed in the near future gets evicted by a user block when switching from the kernel to the user space, and vice versa. All these motivate us to separate the user and kernel blocks in the L2 cache to preserve the useful blocks.

B. The Idea and Implementation

In this study, we explore a way based partition to divide the L2 cache into two separate cache segments, which are used to keep the kernel and user blocks, respectively. Note that the number of sets in both segments keeps the same as that in the default L2 cache design. By doing this, the kernel and user access interferences are effectively alleviated, which greatly helps to bring up the L2 cache hits. Thus,



(a) 360buy



(b) sjeng

Fig. 3: The access trace comparison of 360buy and sjeng.

a large L2 cache size is unnecessary anymore to maintain the same cache miss rate. Although this partition technique could help to boost the performance, maximizing the energy savings without performance loss is more attractive in modern mobile platforms given the quite limited power resources. We then propose to shrink the size of the two segments by decreasing their ways without hurting the performance, and meanwhile, obtain both leakage and dynamic energy savings: (1) it is obvious that the leakage energy is proportional to the structure size, and a smaller L2 leads to lower leakage energy consumption; (2) each L2 cache access will direct to one of the two much smaller segments instead of a large and unified cache, leading to a smaller amount of dynamic energy consumption. We perform the sensitivity analysis and find that a 16-way L2 cache (the default L2 configuration used in most mobile processors) can be partitioned into two 7-way segments for the user and kernel, respectively, to achieve the optimal energy-efficiency.

Generally, L2 cache is physically indexed physically tagged, and directly using the physical address is unable to justify an L2 cache access as the kernel or the user access. For the write-through L1 cache, every L2 cache access can trace back to a TLB access and the "User/Supervisor" (U/S) bit in the corresponding TLB entry can be read to decide whether this reference should be directed to the user or kernel segment of the L2 when it gets an L1 miss. In the case that the "U/S" bit is not available in TLB, the virtual address will be compared with the boundary between the user and the kernel spaces to make the decision.

For the write-back L1 cache, the case becomes a little complicated when there is a block written from the L1 to the L2 because the L1 can only provide the physical address of that block and its virtual address may not be present in the TLB. Thus, a "U" bit is added to each cache block in the L1.

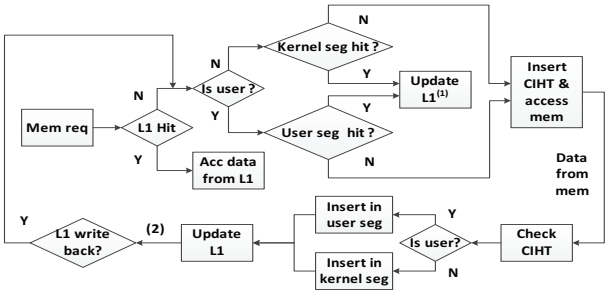


Fig. 4: Illustration of the cache and memory access with the user and the kernel space partitioned in the L2 cache. (1): The L1 is only updated if necessary (for write-no-allocation case, the L1 may not be updated). If blocks get evicted in the L1, the flow goes to (2). (2): The flow continues only with the write-back L1 cache.

It is set as 1 if the block belongs to the user space; otherwise it is set as 0. The "U" bit will help to write the block into the appropriate segment in the L2. Conversely, an L1 block's "U" bit is determined by whether it is from the user or kernel segment of the L2 when it is filled into the L1 cache.

On an L2 cache miss (i.e., the block is not in the corresponding segment), the block information (i.e. belonging to the user or kernel space) will lose when it is fetched from the memory. This is the case for both write-back and write-through L1 cache. We add a small table, called cache insertion handling table (CIHT), to hold the information of the missing block in the L2. The CIHT contains multiple CIHT entries which can be organized fully-associatively. Each CIHT entry has two fields: the block address and "U" bit to record the block information. Before fetch requests are issued to the memory, entries are created in the CIHT. When the blocks are fetched from the memory, they can be directed to the appropriate segment in the L2 by looking up the CIHT. Then corresponding entries in the CIHT can be deleted. Fig. 4 shows the flow-chart of the cache access when our kernel and user segments enabled.

Our kernel and user partition technique can be easily extended to support the cache coherence protocol. The "U" bit is attached to each block in all other caches which are not partitioned and it will go along with its block. When other cache initiates a transaction to the partitioned cache, the "U" bit will be used to determine which segment should be accessed. On the other hand, when the partitioned cache initiates a transaction to other caches, the "U" bit will be generated and send out depending on which segment (user or kernel) starts the transaction.

C. Overhead Analysis

Our partition technique introduces a quite small CIHT table and the "User" bit to L1 cache. The number of entries included in the CIHT table is determined by the number of outstanding L2 cache misses which is usually less than 32 in modern mobile processors. The "User" bit only adds one bit to each L1 block. Therefore, the area and energy

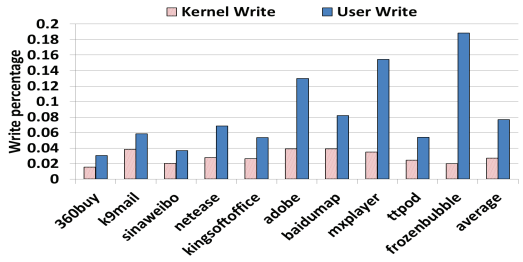


Fig. 5: The percentage of the overall, the kernel and the user write in all the L2 cache accesses.

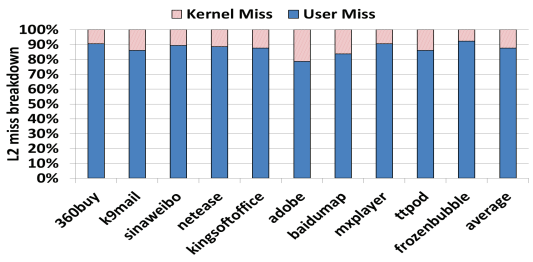


Fig. 6: The breakdown of the L2 misses.

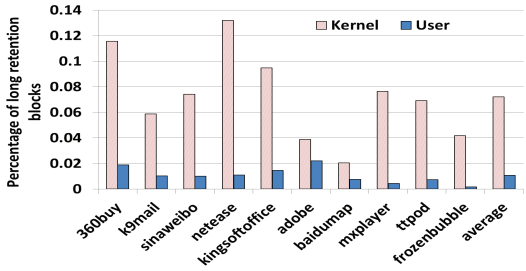


Fig. 7: The percentage of long retention blocks for the kernel and the user segment.

overhead is negligible (<1%) comparing with the large L2 cache size. Not mention that we shrink the overall segments' size whose area and energy savings are far beyond such trivial overhead. Moreover, the access latency to the CIHT table is also negligible comparing with the long latency on an L2 miss. Thus, there is almost no performance overhead in our technique.

III. Multi-retention STT-RAM for the L2 Cache in Mobile Platform

A. Background

Nowadays, the non-volatile memory, such as STT-RAM, has drawn a lot of attention because of its high density and low leakage power. However, the write operation of the STT-RAM, takes longer time and consumes more energy. Thus, directly using it for the L2 cache design will cause significant dynamic energy overhead. Previous research [13] has shown that the write latency and energy of STT-RAM can be reduced by relaxing the data retention time. The STT-RAM could be configured to different retention levels to satisfy various design requirements and achieve the best energy-efficiency. However, aggressively reducing the retention time in exchange for smaller write penalty may not be desirable because the data stored in the STT-RAM needs to be frequently refreshed.

B. Key Observations: The Distinct Access Patterns Between The User and Kernel Segments

When enabling our user and kernel partition scheme explored in Section II, we observe quite different accesses behaviors to the user and the kernel segments, which makes multi-retention STT-RAM a great choice for our newly developed L2 cache design.

Fig. 5 profiles the percentage of the kernel and user write accesses in all the L2 cache accesses across the investigated benchmarks (details in Section IV). As it shows, the kernel has quite few write operations to the L2, which only accounts for 3% of the total L2 accesses and is much lower than the user write percentage (~8%). This is because the kernel is mostly instructions and instructions are read-only.

Since each miss will generate a cache fill that is essentially a write operation, Fig. 6 further investigates the number of misses for the user and the kernel accesses. We observe that the number of kernel misses is about 12% of all the L2 misses, and the remaining 88% misses are caused by the user accesses. This is because user space includes high percentage of data accesses which have much worse locality compared with instructions. Filtered by the L1 data cache, the data accesses in the L2 cache have bad locality, leading to high data miss rate.

Finally, we profile the percentage of long retention blocks contained in both user and kernel segments. In our study, the long retention blocks are defined as blocks that stay in the segment for more than 1s from its fill to the eviction. If the same block is inserted many times in the segment, it is deemed as multiple blocks. Fig. 7 shows that the kernel segment has higher percentage of long retention blocks (around 7%) while it is only 1% for the user segment. The reason is that the kernel accesses in mobile applications are usually generated by the GUI display, Internet access, file reading, audio playing and so on. The drivers and other system components for these operations will be constantly accessed. Thus, their cache blocks will have long retention time. On the other hand, the user program usually comprises of more diversified tasks and the code/data reuse is less frequent than the kernel.

C. The Idea and Implementation

Considering that the kernel segment has much fewer writes, fewer cache misses, and more long retention blocks as compared to the user segment, we propose using short-retention STT-RAM for the user segment and the long-retention STT-RAM for the kernel segment. By doing this, the kernel blocks can be preserved for a long time with minimal write energy overhead, and the high write energy consumption caused by the frequent write and fill operations in the user segment can be effectively alleviated by the short-retention STT-RAM. In

TABLE I: Detailed Machine Configurations.

Processor	ARM Cortex-A7, 2GHz, Out of order
L1 I-cache	32K, 4-way, 64B cache line, write back, write allocate
L1 D-cache	32K, 4-way, 64B cache line, write back, write allocate
L2 cache	1M, 16-way, 64B cache line, degree 8 stride prefetcher
Memory	100 cycles latency

TABLE II: The configurations for the SRAM and the STT-RAM. "STT-RAM (L)": long retention STT-RAM; "STT-RAM (S)": short retention STT-RAM.

	SRAM	SRAM	STT(L)	STT(L)	STT(S)
Cache size	1M	448K	1M	448K	448K
Read latency(cycles)	12	8	12	8	8
Write latency(cycles)	12	8	65	48	23
Read Energy(nJ)	0.69	0.34	0.67	0.32	0.32
Write Energy(nJ)	0.67	0.33	3.4	2.3	1.1
Leakage Power(mW)	284	124	26	11	11
Retention Time	-	-	4yr	4yr	3.24s

order to prevent the data loss, a refreshing unit is needed to monitor the access of all the cache blocks. If a cache block is not accessed before its data expire, the refreshing unit will refresh the cache block. As shown in Fig. 7, only 1% of the user blocks need to be refreshed, which is negligible overhead.

IV. Evaluations

A. Experimental Setup

We use Gem5 [1] as the simulator to model the ARM Cortex-A7 architecture, and run full system simulation with the Android Ice Cream Sandwich (ICS) OS. We use the Moby benchmark [8] suite to evaluate our techniques. Applications contained in Moby are popular mobile applications and have high downloads on Google play store. They span various categories, including web browser, social network, email, and so on. The detailed machine configuration is summarized in Table I. Our power model is built on top of McPat [10] and the STT-RAM is modeled by the NVSim [5]. We focus on the 32nm technology, which is common for mobile platform. The refreshment of a cache block involves one cache read and one cache write, and we model this overhead in our experiments. The detailed configurations of the power model are shown in Table II.

B. Results

In this section, we first evaluate the effectiveness of partitioning the user and kernel in the L2 cache. Then we justify the way reduction scheme in energy saving for emerging mobile platforms. As shown in Fig. 8, the L2 miss rate is improved by 5% on average by assigning 9 ways for the user segment and 7 ways for the kernel segment due to the alleviated interferences. Since the partition scheme effectively reduces the miss rate, we remove some ways to save power and still keep comparable miss rate. We identify the best scheme is "7U 7K" (i.e., 7-way user and 7-way kernel segments), which increases the L2 miss rate only by 1% and thus causes almost no performance loss as shown in Fig. 8. Meanwhile, as shown in Fig. 9, "7U 7K" SRAM design can reduce the energy consumption by

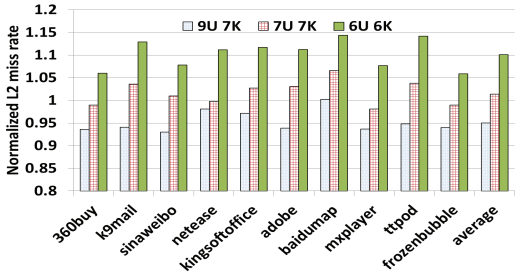


Fig. 8: The L2 miss rate with the separated user and kernel segments. (xU yK: x ways for the user and y ways for the kernel.).

around 15% comparing with the baseline case without any optimization. Further reducing to "6U 6K" will lead to 10% L2 miss rate increase, which is not desirable.

Among all the benchmarks, netease and baidumap show very small improvement or no improvement. It is because they have longer reuse distance and need more ways to prevent the thrashing. When assigning 11 ways for the user segment, these two benchmarks show 6% and 3% improvement on the L2 miss rate, respectively. In addition, our "7U 7K" partition scheme causes a relatively high L2 miss rate in k9mail, adobe, baidumap and ttpod's. The reason is that these benchmarks have higher user miss rate compared with other benchmarks, so further shrinking the size of the user segment will considerably hurt the L2 miss rate.

Fig. 9 also shows the energy consumption when applying the basic STT-RAM (i.e., long-retention STT-RAM) to build both 7-way user and 7-way kernel segments, and when our multi-retention STT-RAM scheme is enabled to those two segments. The results are normalized to the baseline case that uses SRAM without any optimization. As it shows, our "7U 7K" multi-retention STT-RAM design can further save another 60% energy when comparing with the "7U 7K" SRAM design. Moreover, it also outperforms the "7U 7K" basic STT- RAM design by reducing the energy consumption by 25%. Fig. 10 shows the performance of the basic STT-RAM and our multi-retention STT-RAM design. The results are also normalized to the baseline SRAM case. Comparing with the 9% performance loss caused by the basic STT-RAM design, our multi-retention cache design only suffers ~2% performance penalty.

Overall, our multi-retention STT-RAM design can achieve 75% L2 energy consumption with only 2% performance loss when comparing with the default SRAM-based and unified L2 cache design.

V. Related Work

Several mobile benchmarks, like BBench [7], MobileBench [3] and Moby [8], are proposed for mobile architecture study. Pandiyan el at. [4] investigate and quantify the energy cost of data movement for emerging smart phone workloads. Zhu et al. propose a DVFS technique [15] and "Webcore" architecture [16] for energy efficient web browsing. In the past, cache partitioning strategy has been studied, e.g. programming semantics

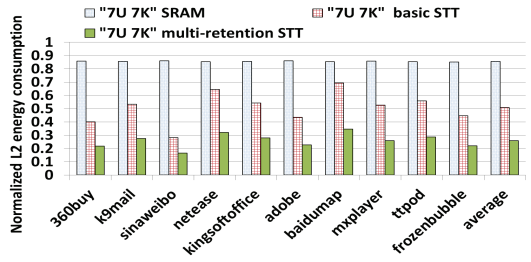


Fig. 9: The L2 energy consumption of the SRAM based, basic STT based, and multi-retention STT-based "7U 7K" partition schemes (all normalized to the baseline case using SRAM without any optimization).

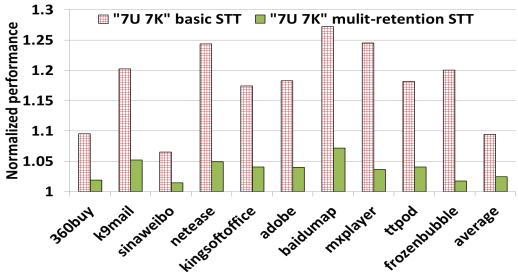


Fig. 10: The performance of the basic STT and the multi-retention STT based "7U 7K" partition schemes (all normalized to the baseline case using SRAM without any optimization).

based partitioning [9]. The distinction of the kernel and the user code is also explored. Chakraborty et al. [2] propose the computation spreading, which distributes dissimilar computation fragments to different cores while groups similar ones together, and also apply it to the kernel and user code. Li et al. [11] explore different issue widths for the OS and the user code. However, none of these work targets at the interference of the user and the kernel space for energy-efficient L2 cache design.

VI. Acknowledgement

This work is supported in part by NSF grants CCF-1320730, CCF-1351054 (CAREER), EPS-0903806 and matching support from the State of Kansas through the Kansas Board of Regents.

VII. Conclusions

In this paper, we focus on the energy-efficient cache design in emerging mobile platforms. We observe the serious interferences between the user and kernel blocks in the L2 cache, and propose to separate the L2 cache into the user and kernel segments. Meanwhile, we reduce the number of ways in each segment to optimize the energy without hurting the performance. We further find completely different access behaviors between the two separated kernel and user segments in our novel L2 cache design, and introduce the multi-retention STT to maximize the cache energy savings. Our proposed technique can reduce the L2 energy consumption by 75% with only 2% performance loss.

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[2] K. Chakraborty, P. M. Wells, and G. S. Sohi. Computation spreading: Employing hardware migration to specialize cmp cores on-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 283–292, New York, NY, USA, 2006. ACM.

[3] C. W. D. Pandiyan, S. Lee. Performance, energy characterizations and architectural implilcations of an emerging mobile platform benchmark suite - mobilebench. In *IEEE International Symposium on Workload Characterization*, 2013.

[4] C. W. D. Pandiyan, S. Lee. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *IEEE International Symposium on Workload Characterization*, 2014.

[5] X. Dong, C. Xu, Y. Xie, and N. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7):994–1007, July 2012.

[6] I. Gartner. Forecast: Pcs, ultramobiles, and mobile phones, worldwide, 2011-2018, 2q14 update, 2014.

[7] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-system analysis and characterization of interactive smartphone applications. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, IISWC '11, pages 81–90, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Y. Huang, Z. Zha, M. Chen, and L. Zhang. Moby: A mobile benchmark suite for architectural simulators. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 45–54, March 2014.

[9] H.-H. S. Lee and G. S. Tyson. Region-based caching: An energy-delay efficient memory architecture for embedded processors. In *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '00, pages 120–127, New York, NY, USA, 2000. ACM.

[10] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480, Dec 2009.

[11] T. Li and L. K. John. Routine based os-aware microprocessor resource adaptation for run-time operating system power saving. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, ISLPED '03, pages 241–246, New York, NY, USA, 2003. ACM.

[12] A. Shye, B. Scholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 168–178, New York, NY, USA, 2009. ACM.

[13] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu. Multi retention level stt-ram cache designs with a dynamic refresh scheme. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 329–338, New York, NY, USA, 2011. ACM.

[14] J. Wang, X. Dong, and Y. Xie. Oap: An obstruction-aware cache management policy for stt-ram last-level caches. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 847–852, March 2013.

[15] Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 13–24, Washington, DC, USA, 2013. IEEE Computer Society.

[16] Y. Zhu and V. J. Reddi. Webcore: Architectural support for mobile web browsing. In *Proceedings of the 41st International Symposium on Computer Architecture (ISCA)*, ISCA '14, 2014.