JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 12, No. 3, September 2018, pp. 106-114

Exploring GPU Data Cache Leakage Management Techniques

Hao Wen and Wei Zhang*
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
wenh2@vcu.edu, wzhang4@vcu.edu

Abstract

In this paper, we study how to reduce cache leakage energy efficiently for GPU data caches (L1 and L2). The access pattern of GPU cache is different from that of the CPU, which usually has little locality and high miss rate. In addition, GPU can hide memory latency more effectively due to multi-threading. Because of the above reasons, we find it is possible to place cache lines of GPU data caches into the low power mode more aggressively than traditional leakage management for CPU caches that can reduce more energy leakage without significant performance degradation. In some cases, we find it is possible for the GPU to bypass the L1 data cache to save 100% energy leakage while generating better performance. Also, we propose to combine the drowsy and gated-VDD techniques that can exploit short and long access intervals to minimize energy leakage with insignificant performance overhead. Interestingly, we may achieve better performance for some benchmarks due to the different cache access patterns after applying the leakage reduction method.

Category: Cloud Computing / High Performance Computing

Keywords: GPU; Cache; leakage reduction; Two-level low power mode

I. INTRODUCTION

Traditionally, dynamic energy dominates total energy consumption of integrated circuits. However, as the size of transistors is scaled down, the leaked energy becomes a larger portion of the total energy consumption. Graphics processing units (GPUs), originally designed for fast graphical computation, have rapidly become a popular choice for high-performance computing. To support the concurrent execution of a massive number of threads, a GPU consists of an array of highly threaded streaming multiprocessors (SMs). Each SM has its own L1 cache and usually shares an L2 cache. For example, in the NVIDIA Fermi GTX480 architecture, there are 15 SMs. They share a 768 kB L2 data cache and each SM has its own L1 data cache that can be configured to 16 kB or 48 kB. L1 and L2 data caches occupy a large on-chip area, which are good targets for energy leakage reduction.

GPUs are also promising for high-performance embedded computing due to their high throughput and energy efficiency. For example, GPUs have been used in computation-intensive real-time and safety-critical applications such as medical data processing [1], autonomous auto navigation [2], and vision-based aircraft controls [3]. All these applications must meet strict deadlines and require high system throughput, making GPUs ideal computing engines for them. For embedded systems powered by battery, it is particularly critical to increase energy efficiency of computing for extended battery lifetime.

Many techniques have been proposed in the past to reduce CPU cache energy leakage with insignificant performance gain and dynamic energy overheads, for example, cache decay [4] and drowsy cache [5, 6]. In cache decay, a cache line is supply-gated if it has not been accessed for a period, which can completely remove cache leakage dissipation but does not preserve the state

Open Access http://dx.doi.org/10.5626/JCSE.2018.12.3.106 http://jcse.kiise.org

Received 28 March 2018; **Revised** 09 June 2018; **Accepted** 16 August 2018
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

in the decayed cache line. In contrast, drowsy cache is a state-preserving mechanism that uses dynamic voltage scaling to reduce leakage in unused portions of the cache memory.

Cache decay [4] and drowsy cache [5, 6], however, are proposed based on CPU caches. Since GPU caches exhibit quite different access patterns from CPU caches (e.g., little locality, high miss rate, high memory level parallelism due to multi-threading), existing leakage management developed for CPU caches should be revisited or adapted to specifically and efficiently reduce energy leakage for the GPU cache.

The major contributions of this paper include the following:

- We find that in the GPU architecture, the cache lines (L1 and L2) can be placed into the low power modes more aggressively (e.g., putting the cache line into the drowsy mode immediately after it is accessed) to reduce more energy leakage without incurring significant performance overhead.
- We re-evaluate a hybrid drowsy-gated VDD (HDG) technique proposed for a hybrid SPM-cache architecture [7], which can put cache lines into the drowsy mode for short idle intervals and put the cache lines into the decay mode for long idle intervals to maximize leaked energy reduction.
- There are some interesting GPU data cache behaviors different from CPU that favor cache leakage reduction for GPUs. For example, (1) bypassing the L1 cache to entirely save energy leakage while also have better performance, and (2) after the leakage reduction method is applied, GPU will have a different access pattern due to warp scheduling that may improve performance based on the benchmarks we studied.

The remainder of the paper is organized as follows. Section II presents different cache leakage management techniques. The evaluation methodology is described in Section III and the experimental results are given in Section IV. We discuss related work in Section V. Finally, we make conclusions in Section VI.

II. EXPLORING CACHE LEAKAGE REDUCTION TECHNIQUES

Energy leakage is primarily caused by small amount of current that leaks between source and drain terminals of a transistor. So in general, cache energy leakage can be reduced by either completely turning off the supply voltage or scaling down the supply voltage to a low power mode. The former is a state-destroying mechanism that can maximally reduce energy leakage, but the valid data of the gated cache line are lost, and so it may introduce extra cache misses. In contrast, the voltage scaling down technique is a state-preserving mechanism that puts cache lines into the drowsy mode to reduce energy leakage by lowering the supply voltage level while the content of the cache line remains valid. When the drowsy cache line is accessed in the future, it must be awakened from the drowsy mode first, which only takes 1 or 2 clock cycles. So compared to the state-destroying mechanism, the state-preserving mechanism has a relatively smaller performance penalty, but allows less leaked energy in the same period of time.

A. Periodic Drowsy Method

In drowsy cache [5], a drowsy circuit associated with each cache line is used to turn the cache line into a low power drowsy mode. In the drowsy mode, most of the leaked energy is saved while the content of the cache line is preserved but cannot be accessed immediately. To access a drowsy cache line, it has to be awakened from the drowsy mode first. The activation process takes 1 or 2 cycles. So if there is a cache hit on the drowsy cache line, it generally introduces performance penalty. But if the access is a miss on the drowsy cache line, the wake up penalty can be hidden behind miss handling processes.

The periodic drowsy (PD) method puts all cache lines into the drowsy mode periodically, for example, every 2,000 cycles. The drowsy cache line is awakened whenever it is accessed. Choosing a drowsy period is important because the tradeoff between leaked energy reduction and performance overhead must be considered. Shorter drowsy periods can put cache lines into the drowsy mode for a longer period of time, which can save more leaked energy at the cost of larger performance overhead due to the larger number of cache activations from the drowsy mode. A longer drowsy period keeps more cache lines in the active mode, so less energy leakage is saved, but it also reduces performance overhead.

For the CPU cache, the drowsy period cannot be too short for performance consideration, and so some opportunities to minimize leakage consumption are lost. However, the GPU cache generally has a higher miss rate to hide the drowsy line activation latency, indicating that the PD method can be applied to the GPU cache more aggressively.

B. Cache Decay

Instead of scaling down supply voltage, cache decay [4] is a gated-VDD method that completely turns off supply voltage. This method tries to fully remove leakage consumption at the dead time of cache generations [8]. A cache generation begins at a cache miss on a cache line, then a number of accesses will hit that cache line after a new memory block is filled into the cache line because of temporal locality. A cache generation ends when the cache line is evicted and a new generation begins. As shown in Fig. 1, the access interval is the duration between two consecutive accesses during live time, whereas

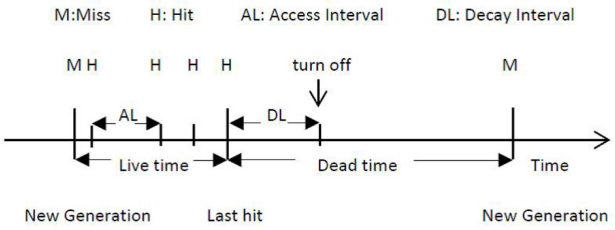


Fig. 1. Cache generations.

the duration between the last hit and the beginning of the next generation is called dead time. During the dead time, which is typically a long interval, since there is no access to that cache line, supply voltage can be gated to save energy leakage.

In the cache decay, a counter is associated with each cache line to count time that has elapsed since the last access. The counter has a pre-determined decay interval value, to predict if a cache line enters dead time. Whenever a cache line is accessed, the corresponding counter will be reset and begins counting simulation cycles. If the interval between two consecutive accesses is greater than the decay interval, the counter will reach the decay interval and switch off the cache line voltage supply. Hardware and energy overheads of counter-based cache decay were shown to be small [4].

The ideal case of cache decay is to switch off the cache line after the last hit within a cache generation that will not introduce extra misses. Otherwise, the processor must fetch data from the lower level memory that takes much longer than accessing the L1 cache. Extra accesses to the lower level memory also introduce dynamic energy overhead. To reduce the probability of pre-maturely putting a cache line into the decay mode while its content must be accessed again, the decay interval is usually set long. Conversely, if the decay interval is too long, we will lose the opportunity to save more energy leakage.

C. Aggressive Method

The GPU data caches usually have little locality and high miss rate, indicating that we can put the cache line into the drowsy mode more frequently than the CPU cache without incurring significant performance degradation. In the extreme case, we propose to use an aggressive (AG) method that activates a cache line whenever it is accessed and turn it into the drowsy mode immediately after the access. The reason is that the activation penalty is small, and it can be hidden behind the cache miss handling cycles. The higher the miss rate, the more activation penalties are hidden, and therefore the AG method does not significantly increase overall performance overhead. Table 1 shows the L1 and L2 cache miss rates of different benchmarks, most of which are high as compared to typical CPU cache miss rates.

Table 1. GPU data cache miss rate (%)

Benchmark	L1 miss rate	L2 miss rate
BFS	69.26	5.40
CP	50.00	24.87
LIB	66.83	79.33
LPS	70.51	36.37
MUM	15.46	17.75
HOTSPOT	94.36	23.56
NN	1.90	4.30
NW	97.69	97.42
BACKPROP	36.85	27.99
LUD	60.88	2.52

Multi-threading and warp scheduling also contribute to insignificant performance overhead of the aggressive method. If many threads hit the drowsy cache line at the same cycle, the activation penalties overlap and it may contribute only 1 cycle to overall performance overhead. The warp is the unit of thread scheduling in SMs. An SM is designed to execute all threads in a warp following the same instruction but operating on multiple data (SIMD model). When an instruction executed by the warp must wait for the result of a previous long-latency operation (e.g., wait for memory), this warp will not be executed. Instead, another warp will be selected for execution. This mechanism can tolerate memory latency with work from other threads. When we apply the leakage management method, data needed for the thread may be delayed (in a drowsy cache, we have to activate the drowsy cache line first, and in the decay mode, we may get extra cache misses). So, the warp dependent on the delayed warp will not be selected for execution at the time when it would have been executed in the original order without using the leakage management technique. As a result, another ready warp is executed instead. So the access pattern is changed, which may lead to counter-intuitive performance results. For example, previous work [5, 6] demonstrates that drowsy techniques applying on the CPU cache always have positive performance overhead, but on the GPU cache, our experimental results indicate that it will actually improve performance for some benchmarks.

Take the *BFS* benchmark for example. Consider a simple graph as shown in Fig. 2, The *BFS* searches all the vertices starting from source vertex 0. For simplicity, suppose we have a GPU that only has two cores, each of which can only execute one thread. In the first round, one thread finds all the neighbors of vertex 0 (vertices 1, 3, and 4). In the second round, assume that two threads search neighbors of vertex 1 and vertex 3 concurrently and vertex 2 is found. In the third round, vertex 5 is discovered. So 3 rounds are needed to finish the *BFS*

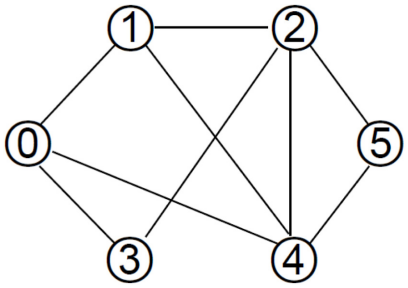


Fig. 2. BFS example.

search. After the leakage management method is applied, discovery of vertex 3 may be delayed because of activation from the low power mode. So different from the original scheduling order, one thread is scheduled to search from vertex 4 in the second round and only two rounds are needed to touch all the vertices. So, overall performance is improved.

D. Hybrid Drowsy-Gated VDD (HDG)

Hybrid drowsy-gated VDD (HDG) [7] is a hybrid method that combines advantages of drowsy cache and cache decay techniques. This is because a decay interval cannot capture short dead times, which will lose opportunities for leakage reduction during dead time. Although adaptive decay [4] works better than fixed decay interval for this situation, it cannot reduce energy leakage during live time and waiting time (waiting time is the decay interval after the last hit, during which the cache line is kept active until the decay interval elapses). Ideally, an optimal method should be able to switch off the cache line immediately after the last hit.

Effectiveness of leakage management techniques is dependent on cache access patterns. As an example, Fig. 3 shows the access pattern of different GPU L1 data cache lines monitored from *BFS*. Access intervals between consecutive hits are usually thousands of cycles (i.e., short idle intervals). The interval between the last hit and the next miss is usually 10 thousand cycles (i.e., long idle intervals. e.g., the interval between 227424 and 243707 in the second graph is 16283 cycles). Decay interval is chosen to be longer than most of the short idle intervals but smaller than the long idle interval so that it will not introduce many cache misses. However, we lose the opportunity to save energy leakage of short idle intervals.

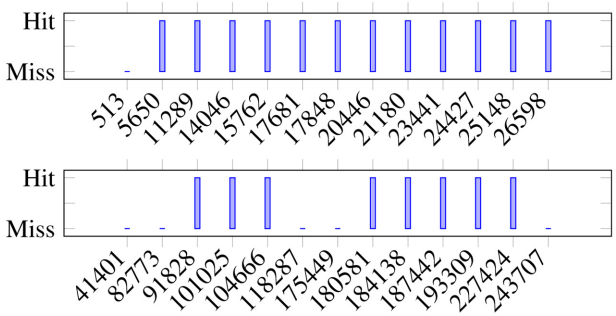


Fig. 3. Access pattern of different cache lines monitored from BFS benchmark. Horizontal coordinates are execution time in cycles.

To save more energy leakage by taking advantage of various access intervals of live time and waiting time, two-level low power modes are used. The first level is the drowsy mode for short idle intervals, and the second level is gated-VDD for long idle intervals. In previous work [7] for CPU caches, two different decay intervals are proposed, when the counter reaches the short interval (for example, 1k cycles), the cache line is placed into the drowsy mode by scaling down supply voltage, and the counter keeps increasing. When the counter reaches the longer decay interval (for example, 16k cycles), it switches off power to the cache line. Instead of using the short interval, we propose to use the AG method on the first-level low power mode for GPU data caches, which puts the cache line into the drowsy mode immediately after the cache line is accessed. Since the first-level drowsy mode preserves content in the cache line, no extra misses are introduced. The decay mode works on the second level and switches off the cache line.

III. EVALUATION METHODOLOGY

We use GPGPU-Sim [9] to implement and evaluate different methods to reduce energy leakage. The configuration for GPGPU-Sim is based on Fermi GTX480 architecture. Detailed architectural parameters of the GPU can be seen in Table 2. Energy results are obtained by using the GPUWattch [10]. Benchmarks are chosen from ISPASS2009 [11] and Rodinia 2.4 [12].

Periodic method and AG method operate on the drowsy cache. Performance overhead is caused by waking up a drowsy line. HDG may have extra cycles of waking up a drowsy cache line and accessing next level memory. Different GPU access patterns also have impact on overall

Table 2. Default GPGPU-Sim configuration

Number of SMs	15
Size of L1 data cache per SM (kB)	48
L1 & L2 data cache block size (B)	128
L1 data cache associativity	4
Size of shared memory per SM (kB)	16
Size of L2 cache (kB)	768
L2 data cache associativity	8
Core clock frequency (MHz)	700

performance that may improve or degrade performance.

Energy overhead of the drowsy cache includes two parts. The first is extra energy leakage consumption during extra execution cycles. The second is the dynamic energy of transition between active and drowsy modes. In the gated-VDD technique, besides extra energy leakage consumption during extra execution cycles, energy overhead is mainly caused by extra dynamic energy to access next level memory. In HDG methods, dynamic energy caused by counters is negligible when using gray code counters to tick at a much coarser level [4].

In this paper, normalized leakage reduction (NLR) is calculated by Eq. (1), wherein N is the number of cache lines, T is the original execution cycles without using any leakage management method, T_i and T_i' are execution cycles of the i-th cache line in the active and drowsy mode, respectively, E is original energy leakage consumption, and E' is extra dynamic energy of transition between active and drowsy modes. We assume that when a cache line is in drowsy mode, it only consumes 10% of original leaked energy [13]. In this equation, we only focus on the NLR for the cache and do not consider dynamic energy overhead of accessing next level memory. Dynamic energy overhead is included by measuring total GPU energy consumption. Overall normalized energy consumption after applying leakage reduction methods is illustrated in Section IV-B.

$$L_r = 1 - \frac{\sum_{i=1}^{N} T_i \cdot \frac{E}{NT} + \sum_{i=1}^{N} T_i' \cdot \frac{0.1E}{NT} + E'}{E} \quad (1)$$

Normalized performance overhead (NPO) is calculated by Eq. (2), wherein T_e is the extra execution cycles after the leakage reduction method is applied (may be negative).

$$P_o = T_e / T \quad (2)$$

IV. EXPERIMENT RESULTS

A. HDG Method

Fig. 4 shows how decay intervals influence the miss rate and total GPU energy consumption. We apply the HDG method on L1 and L2 independently to yield results separately. Although a shorter decay interval can save more energy leakage (This is shown in Figs. 5 and 6), it also increases the miss rate and total GPU energy. Increased energy mainly comes from dynamic energy to access lower level memory. So it is important to choose the decay interval. If it is too short, there is too much dynamic energy overhead due to extra misses, and if it is too long, we lose the opportunity to save more energy leakage.

Decay intervals do not impact on *CP* and L1 cache of *LPS*. This is the situation that all the access intervals between hits are shorter than 1k cycles, so the HDG method does not introduce extra misses.

Since the access pattern is changed after the HDG method is applied, total energy may not strictly follow the trend of miss rate (e.g., L1 of *LIB*, L2 of *LPS*). The different access pattern may improve overall performance and so is more energy efficient. This behavior is different from previous research on the CPU cache that a shorter decay interval always increases dynamic energy overhead [4].

Figs. 5 and 6 show the NLR and NPO of the HDG method on the L1 and L2 caches, respectively. Shorter decay intervals can reduce more energy leakage. Also, on average, it increases performance overhead. For L1 cache, average NLR only increases 94.81% to 96.12% while average NPO increases -0.04% to 11.51% as the decay interval decreases from 10k to 1k. Obviously, 10k is a proper decay interval for L1. L2 cache is similar, where average NLR increases 94.56% to 98.83% as the decay interval decreases from 15k to 1k, but average NPO increases 0.73% to 3.69%.

For some benchmarks, performance overhead increases dramatically when decay interval is decreased. The reason is that the miss rate increases significantly. For example, the miss rate of the *MUM* on L1 increases 13.36% to 32.63% when the decay interval decreases 10k to 1k and the miss rate of the *LUD* on L2 increases 2.53% to 26.25% when the decay interval decreases 15k to 1k. Note that the increased miss rate may not totally contribute to overall performance overhead, because the warp scheduling may hide memory latency well. For example, in Fig. 4, we observe that the miss rate of *MUM* on L2 increases significantly when the decay interval is decreased, but performance overhead only increases 0.71% to 2.27%.

B. Comparison of Different Methods

In order not to introduce too much dynamic energy, we must select a proper decay interval that introduces insignificant extra misses for GPU caches. We choose 10k cycles for the L1 data cache and 15k cycles for the L2 cache. Decay interval for L2 is longer because the L2 cache is accessed less frequently, so access intervals between L2 hits are longer. For the PD method, in the previous study of [5], a drowsy period of 4,000 cycles (determined empirically) reaches a reasonable compromise between leakage energy savings and performance on CPU caches. But for GPU caches, our experiment suggests that a shorter period of 1,000 cycles is a more reasonable choice (it can save more energy leakage without introducing significant performance overhead due to the reasons discussed in Section II-C). Since access patterns may be changed and the HDG method may increase dynamic energy consumption to lower level memory, to compare different methods, we subtract the saved leaked energy from the measured total GPU energy and then normalize

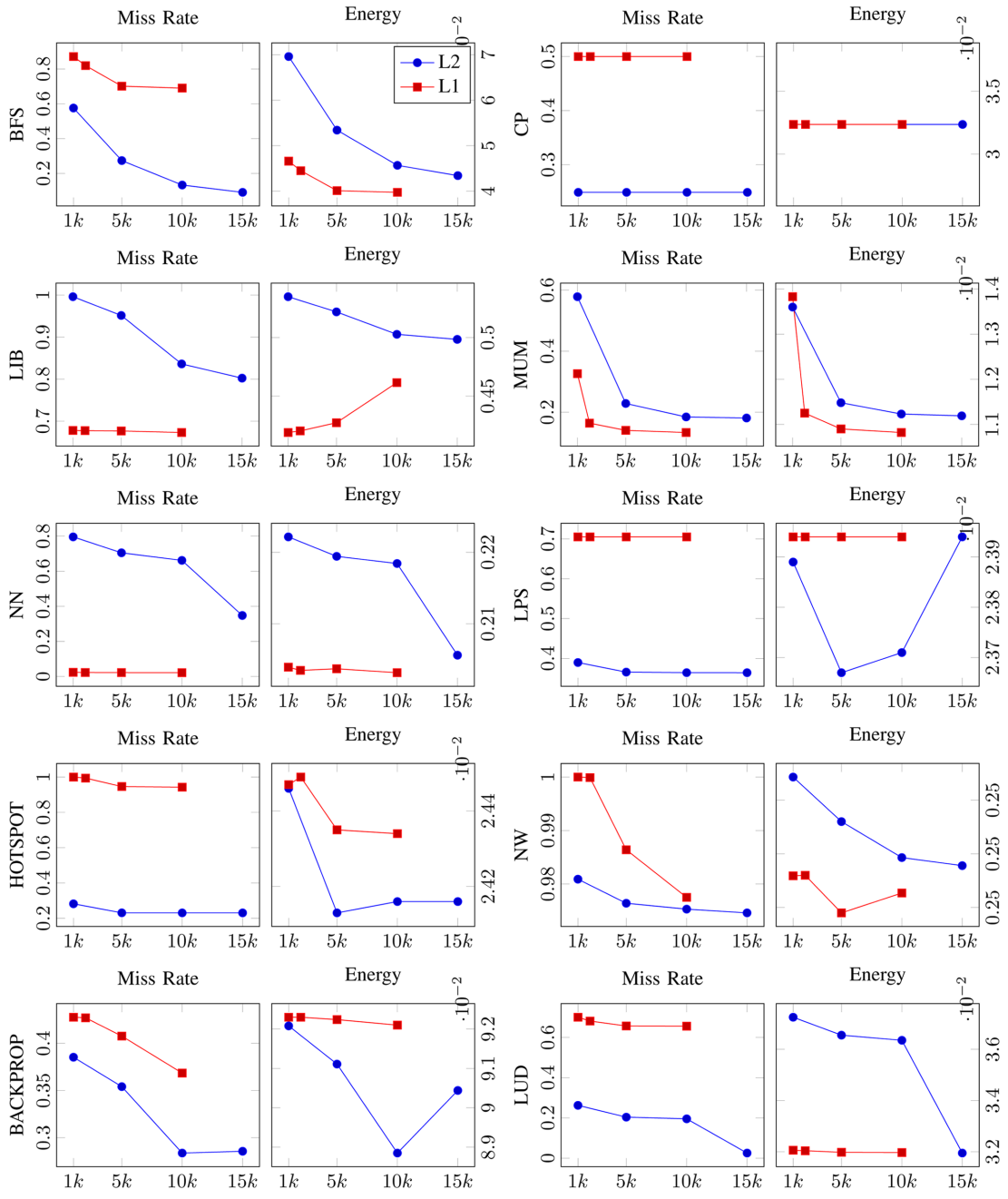


Fig. 4. Miss rate and total GPU energy consumption results of different benchmarks for the HDG method. Horizontal coordinates are different decay intervals in cycles.

them to the original total GPU energy.

Figs. 7 and 8 are the normalized energy and performance overhead results of different methods on L1 cache and L2 cache, respectively. For the GPU L1 data cache, the HDG method is the most energy-efficient, and performance overhead is -0.04% (i.e., improved) on average. For the L2 cache, the PD method leads to the least energy consumption. While AG and HDG methods can reduce total energy consumption for all benchmarks except *BFS*, they may lead to more dynamic energy consumption to access lower-level memory (which is especially significant for *BFS*). The HDG method is less effective on L2 cache in two ways: first, it loses opportunities to save more energy leakage in the drowsy mode due to the longer decay interval; second, extra misses of L2 introduced by the HDG method result in accesses to main memory, which are more energy inefficient than extra misses of L1 that will access the L2 (possibly will access main memory too).

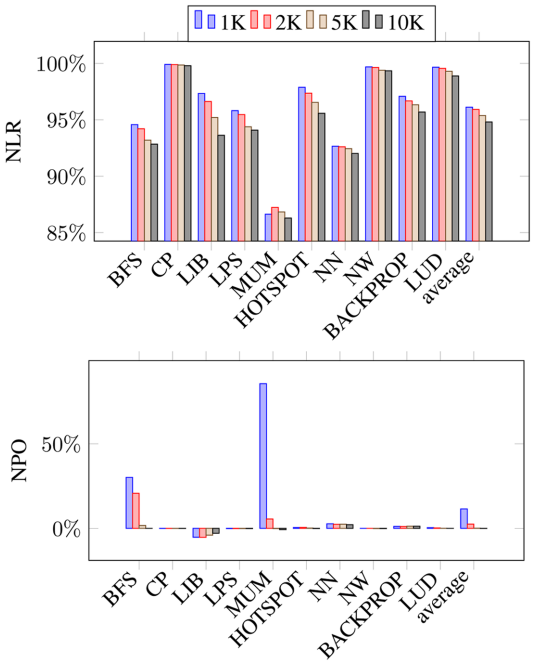


Fig. 5. Comparison of the normalized leakage reduction (NLR) and normalized performance overhead (NPO) of the HDG method on L1 data cache with different decay intervals.

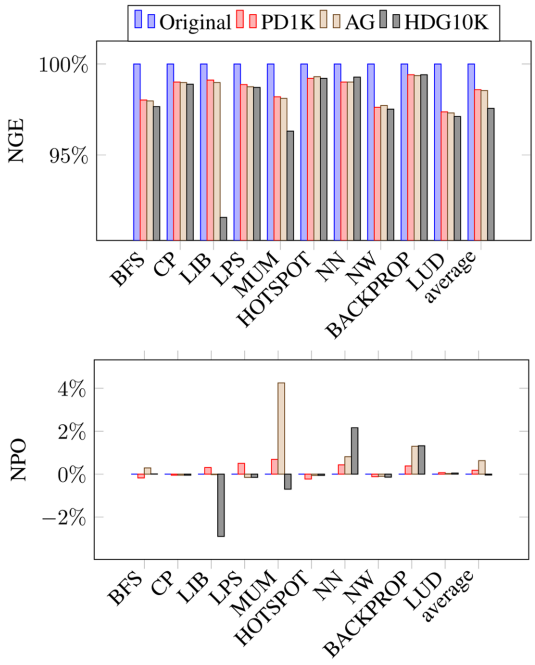


Fig. 7. Comparison of the normalized total GPU energy (NGE) and normalized performance overhead (NPO) when different methods applied to L1 data cache.

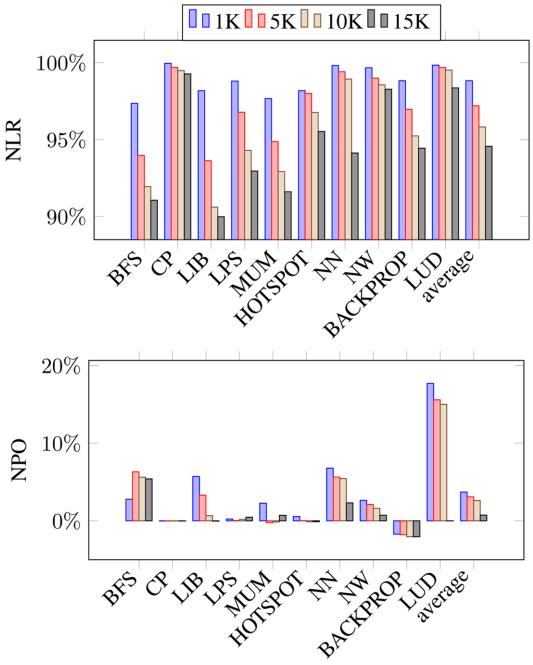


Fig. 6. Comparison of the normalized leakage reduction (NLR) and normalized performance overhead (NPO) of the HDG method on L2 data cache with different decay intervals.

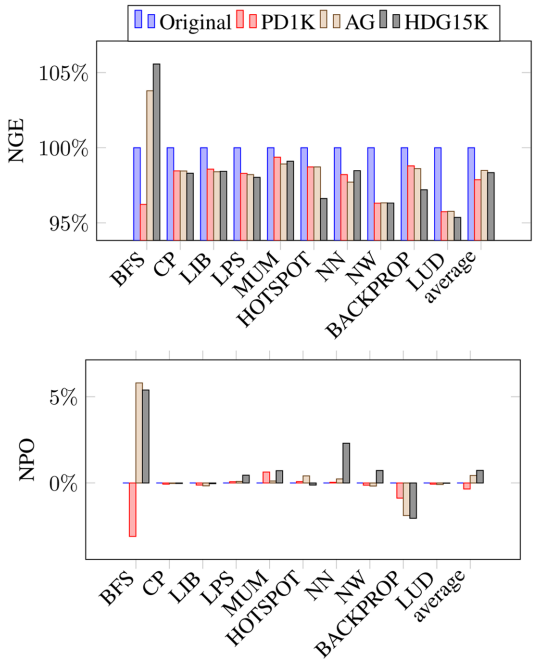


Fig. 8. Comparison of the normalized total GPU energy (NGE) and normalized performance overhead (NPO) when different methods applied to L2 data cache.

C. Bypass the L1 Cache

While the L1 data cache can generally reduce the number of accesses to the lower-level memory hierarchy if there are sufficient data locality, in GPU architecture, loading data into the L1 data cache may increase memory

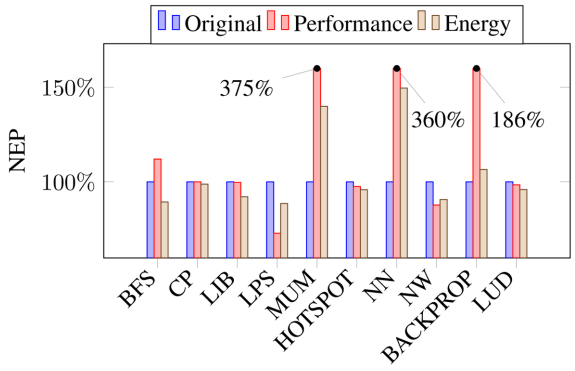


Fig. 9. Normalized energy and performance (NEP) results for bypassing L1 data cache.

bandwidth pressure and degrade performance. This is because in the CUDA programming model, if required data are cached in the L1 and L2 data caches, memory accesses are done by 128-byte transactions. However, if data are only stored into the L2 cache (i.e., bypassing the L1 data cache), 32-byte transactions are used instead [14]. So, if data are not successfully reused for the cache, using only 32-byte transactions can reduce over-fetching of useless data and decrease memory traffic. Bypassing the L1 data cache for such kinds of load operations may be a better choice to reduce memory bandwidth pressure and attain better performance. Also, it's more energy efficient.

Fig. 9 shows normalized energy and performance results for bypassing the L1 cache. *LIB*, *LPS*, *HOTSPORT*, *NW* and *LUD* have better performance and less energy consumption when the L1 cache is bypassed.

V. RELATED WORK

Cache leakage energy reduction has been studied extensively in the past. Besides cache decay [4] and drowsy cache techniques [5, 6], there are also other studies. Meng et al. [15] also combine the drowsy mode and the gated-VDD to reduce cache energy leakage. However, their study is an offline method that assumes perfect knowledge of cache access patterns such that data can be pre-fetched just before needed to avoid performance overhead. With this assumption, it becomes possible to separate the power consumption problem from the performance problem, enabling theoretical study of the limits of leakage reduction on caches. In contrast, our study is an online and more realistic method that considers leakage reduction and performance overhead, without perfect knowledge of cache access patterns. Particularly, we focus on GPU caches that have different access pattern from the CPU caches.

For GPU energy efficiency based on cache management, Chen et al. [16] applied bypass policy based on dynamically detecting cache contention, coordinated with warp throttling, to improve performance and energy efficiency. Wang et al. [17] proposed run-time power-gating to put the L1 data cache into the low-leakage sleep mode when there are no ready threads to be scheduled and to put the L2 cache into the sleep mode when there is no memory request. In their study, caches are put into the off mode (gated-VDD) when an SM finishes its workload. In contrast, our method directly exploits access intervals of each cache line, so we do not need extra work to detect cache inactivity.

VI. CONCLUSION

Reducing cache energy leakage is critical for conserving total energy dissipation for microprocessors. While there are many studies on optimizing energy leakage consumption for CPU cache memories, it is largely unknown how to minimize cache energy leakage in the GPU architecture wherein cache access patterns and impact on performance can be quite different.

In this paper, we study effectiveness of using traditional cache leakage management techniques including drowsy cache and cache decay to reduce energy leakage for GPU data caches. While these techniques are useful, we find that we can aggressively put GPU cache lines into the low power mode without significant overhead due to the memory latency tolerance ability of GPU architectures. We also explore the possibility of bypassing the L1 data cache to obtain better performance while saving 100% of the energy leakage.

We re-evaluate the HDG techniques aggressively, which can exploit short and long idle intervals to put cache lines into the drowsy and decay mode, respectively. Our experiments indicate that the HDG method can reduce energy leakage equivalent to 2.44% and 1.66% of the total GPU energy on L1 and L2 caches, respectively, with the performance overhead -0.04% on L1 and 0.73% on L2 on average.

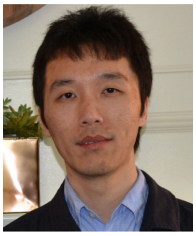
ACKNOWLEDGMENTS

This study was funded in part by the NSF grant (No. CNS-1421577).

REFERENCES

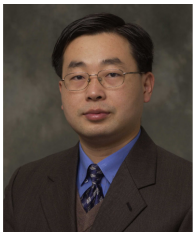
1. U. Verner, A. Schuster, and M. Silberstein, "Processing data streams with hard real-time constraints on heterogeneous systems," in *Proceedings of the International Conference on Supercomputing*, Tucson, AZ, 2011, pp. 120-129.
2. G. A. Elliott and J. H. Anderson, "Robust real-time multi-processor interrupt handling motivated by GPUs," in

Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS), Pisa, Italy, 2012, pp. 267-276.
3. A. Kurdila, M. Nechyba, R. Prazenica, W. Dahmen, P. Binev, R. DeVore, and R. Sharpley, "Vision-based control of micro-air-vehicles: progress and problems in estimation," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Nassau, Bahamas, 2004, pp. 1635-1642.
4. S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, Goteborg, Sweden, 2001, pp. 240-251.
5. K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, Anchorage, AK, 2002, pp. 148-157.
6. N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, Istanbul, Turkey, 2002, pp. 219-230.
7. H. Wen and W. Zhang, "Reducing cache leakage energy for hybrid SPM-cache architectures," in *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, New Delhi, India, 2014.
8. D. A. Wood, M. D. Hill, and R. E. Kessler, "A model for estimating trace-sample miss ratios," *ACM SIGMETRICS Performance Evaluation Review*, vol. 19, no. 1, pp. 79-89, 1991.
9. GPGPU-Sim, http://www.gpgpu-sim.org/.
10. GPUWattch Energy Model Manual, http://www.gpgpu-sim.org/gpuwattch/.
11. A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, 2009, pp. 163-174.
12. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: a benchmark suite for heterogeneous computing," in *Proceedings of IEEE International Symposium on Workload Characterization*, Austin, TX, 2009, pp. 44-54.
13. CACTI Homepage, http://quid.hpl.hp.com:9081/cacti/.
14. NVIDIA CUDA Programming Guide version 5.5, https://developer.nvidia.com/cuda-toolkit-55-archive.
15. Y. Meng, T. Sherwood, and R. Kastner, "On the limits of leakage power reduction in caches," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture,* San Francisco, CA, 2005, pp. 154-165.
16. X. Chen, L. W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W. M. Hwu, "Adaptive cache management for energy-efficient GPU computing," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Cambridge, UK, 2014, pp. 343-355.
17. Y. Wang, S. Roy, and N. Ranganathan, "Run-time power-gating in caches of GPUs for leakage energy savings," in *Proceedings of 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012, pp. 300-303.



Hao Wen

Hao Wen is a Ph.D. student at the Department of Electrical and Computer Engineering of Virginia Commonwealth University. He received his Bachelor's degree in Electrical Engineering in July 2007 from Southeast University, Nanjing, China, and Master's degree in microelectronics in July 2010 from Peking University, Beijing, China. He worked as an IC verification engineer in VIMICRO (Beijing) and Spreadtrum (Shanghai) from 2010 to 2013. His research focuses on computer architectures, and GPU computing.



Wei Zhang

Dr. Wei Zhang is a professor in the Department of Electrical and Computer Engineering at Virginia Commonwealth University. Dr. Wei Zhang received his Ph.D. from the Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as an associate professor (tenured) at Southern Illinois University Carbondale. His research interests are in embedded and real-time computing systems, computer architecture, compiler, and low-power systems. Dr. Zhang has received numerous awards such as the 2016 Engineer of the Year award from RJEC (Richmond Joint Engineering Council), the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and 2007 IBM Real-time Innovation Award etc. Dr. Zhang has received 7 research grants from the National Science Foundation as the PI. In addition, his research and educational efforts have been supported by industry including leading IT companies such as IBM, Intel, Motorola, and Altera. Dr. Zhang has published more than 150 papers in refereed journals and conference proceedings. He is a senior member of the IEEE, and an associate editor of two international journals. He has served as a member of the organizing or program committees for several IEEE/ACM international conferences and workshops.