Run-time Power-gating in Caches of GPUs for Leakage Energy Savings

Yue Wang
CSE Department
University of South Florida
Tampa, Fl.
yuewang@mail.usf.edu

Soumyaroop Roy
Advanced Micro Devices, Inc.
Austin, Tex.
Soumyaroop.Roy@amd.com

Nagarajan Ranganathan
CSE Department
University of South Florida
Tampa, Fl.
ranganat@cse.usf.edu

Abstract—**In this paper, we propose a novel microarchitectural technique for run-time power-gating caches of GPUs to save leakage energy. The L1 cache (private to a core) can be put in a low-leakage *sleep* mode when there are no ready threads to be scheduled, and the L2 cache can be put in *sleep* mode when there is no memory request. The *sleep* mode is state-retentive, which precludes the necessity to flush the caches after they are *woken up*. The primary reason for the effectiveness our technique lies in the fact that the latency of detecting cache inactivity, putting a cache to *sleep* and waking it up before it is accessed, is completely hidden microarchitecturally. The technique incurs insignificant overheads in terms of power and area. Experiments were performed using the GPGPU-Sim simulator on benchmarks that was set up using the CUDA framework. The power and latency modeling of the cache arrays for measuring the wake-up latency and the break-even periods is performed using a 32-nm SOI IBM technology model. Based on experiments on 16 different GPU workloads, the average energy savings achieved by the proposed technique is 54%.**

Keywords – power-gating, GPU, cache, SRAM, leakage power

I. INTRODUCTION AND MOTIVATION

With ever-increasing demand for richer visual experience in computing tasks in all personal computing devices starting from PCs down to tablets and smartphones, graphics processing units (GPUs) are looking to become more pervasive in such devices. The current and future roadmap of the big x86 market players, Intel [2] and AMD [1], include client systems with CPUs and GPUs on the same die. The system-on-chip (SOC) modules for all the mobile platforms, particularly for the passively cooled (fanless) platforms, have to operate under extremely stringent thermal envelopes. Further, due to the demand for longer battery life, aggressive idle (leakage) power management techniques are also applied in these systems. At the core level, the biggest sources of leakage power in GPUs, much like CPUs, are the cache arrays.

In this work, a novel run-time microarchitectural technique is proposed to achieve savings in leakage energy in the caches of GPUs when they are idle during workload execution. Our proposed technique is based on the following salient features:

1. The latencies (*mode-transition latency*) to switch in and out of the low-leakage (*sleep*) mode are microarchitecturally hidden so there is no performance degradation in the execution of a workload.

2. The low-leakage mode that the caches are placed in is state-retentive so the contents of the caches are not lost.

3. The *break-even* period, the minimum period for which a circuit block should stay in the low-leakage mode such that leakage savings break even with the dynamic energy overhead involved in mode switching, is short so the net energy savings are maximized.

To the best of our knowledge there is no prior work that provides a run-time solution to save leakage in caches of GPUs, and our work is the first of its kind. The rest of this paper is organized as follows. Section II provides a brief background of GPU architectures and fundamentals of the power-gating technique. The proposed microarchitectural technique to power-gate the caches is described in Section III. Finally, the experimental set-up and the results are presented in Section IV, followed by conclusions in Section V.

II. BACKGROUND AND RELATED WORK

A. Basics of GPU Architecture

Architecturally, a GPU is essentially a multi-core SIMD engine for executing data-parallel kernel functions (in this paper, we use NVIDIA terminology while using terms related to GPU architecture and tasks). A *grid* is a set of thread blocks that execute a kernel function. Each thread block resides in a *stream multi-processor* (SM), as shown in Figure 2. A group of a fixed number of threads within a block is called a *warp*. Threads within a warp are executed concurrently. Each of the threads resides in one core (in Figure 2) of an SM, processing one data element at a time. L1 cache (shown in Figure 2) is private to an SM, while L2 cache (shown in Figure 1) is shared by all the SM's on the GPU.

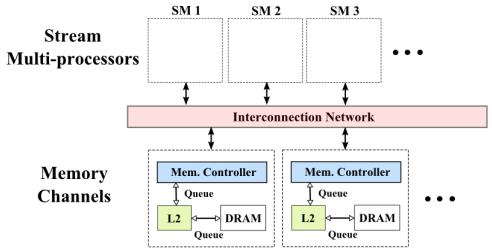

Figure 1. Sketch of GPU architecture [4]

978-3-9810801-8-6/DATE12/©2012 EDAA

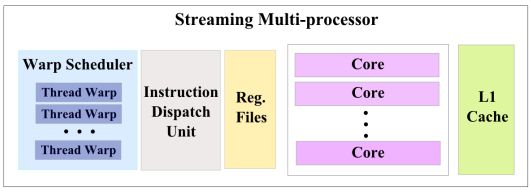

Figure 2. Modules in a stream multi-processor [4]

B. Background of Power-gating Technique

Power-gating is a remarkably effective technique for reducing leakage in idle circuits and is a routine power-management technique in commercial products. One or more high-threshold voltage transistors, known as *sleep transistors*, are inserted between the actual ground and the circuit ground. When the circuit block is idle, the sleep transistor is shut off by two controlling signals, thereby cutting off the leakage path between V_{dd} and ground. This state of the circuit block is called the *sleep* mode. Leakage power will be saved; however, as a trade-off, some of the circuit functions will be affected or restrained. The *mode-transition latency* is the period required for switching the circuit block between *active* and *sleep* mode. Performance degradation will exist if the mode-transition latency is too large. The minimum period for which a circuit block should stay in the low-leakage mode such that leakage savings break even with the dynamic energy overhead involved in mode switching is called *break-even period*.

III. Proposed Microarchitectural Technique

We used a circuit implementation of power-gating proposed in [5]. Two equally sized sleep transistors, M_{s1} and M_{s2}, are placed between the cache array and the ground. Based on the control signals, *cs1* and *cs2* supplied to the gates of the two transistors, the cache can be put in three modes: *active*, *sleep*, and *off*, as shown in Table I. To make the decision of mode transition and to supply the control signals, power-gating controllers are added into the GPU as shown in Figure 3.

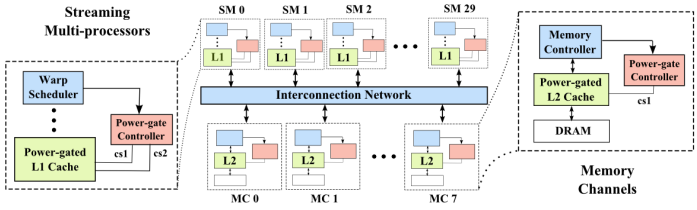

Figure 3. Sketch of power-gate controlling mechnism

A. Power-gating of L1

Each L1 cache array is private to an SM. L1 cache is frequently accessed by the ALU's of each core, except under any of the following two conditions:

- No ready warp to be scheduled. A thread warp is switched out by the scheduler and stays in waiting status if one or more threads within the warp incur an L1 miss. Required data is fetched from lower-level storage (such as L2 cache, or DRAM). The waiting warps become ready once the cycle, a *Stall* instruction will be issued into the pipelines. If none of the warps are ready during a Devices including L1 cache will be idle once encounters one or a series of *Stall* instructions. Therefore, we switch it from *active* into *sleep* mode on this occasion. In this way, leakage power can be saved and the contents are still kept for later accessing when the *Stall* ends.
- An SM finishes its workload. During the execution of a GPU workload, each SM works on its own share of work distribution. They finish at different wallclock times. So when an SM has finished all its assigned threads, it then simply becomes idle, waiting for other SMs to finish. During that period of time, we switch L1 of the finished SM from *active* into *off* mode because the stored contents will not be used any more.

TABLE I. Three Leakage Modes of Cache

Mode	Control Signal		Mode Properties		
	cs1	*cs2*	*Working normally*	*Leakage power saved*	*State-retentive*
active	1	1	√	None	√
sleep	0	1	×	High	√
off	0	0	×	Maximum	×

B. Power-gating of L2

The L2 cache is not accessed as frequently as the L1 cache. Therefore, power-gating the L2 cache may yield more benefits on leakage power saving. As in the case of the L1 cache, two sleep transistors are used to power-gate the L2 cache, but one of the transistors is always on. This means we only have two modes: *active* and *sleep*.

An L2 cache array is switched into *sleep* mode when the corresponding memory controller issues a *Nop* (no operation) command, which is mainly caused by three reasons:

- The L2-to-memory-controller queue is full when trying to serve a DRAM output. The data fetched from a DRAM will be sent to the corresponding L2 array. However, if the L2-to-memory-controller queue is full, the L2 array cannot process the current DRAM output. The corresponding memory controller will issue a *Nop*.
- The L2-to-DRAM queue is full when trying to push a memory request to DRAM. The memory request for data is sent on an L1 miss from SM down to the L2 through the interconnection network. If the L2 does not have the required data either, the memory request will be sent further down to DRAM via being pushed into L2-to-DRAM queue. If the L2-to-DRAM queue is already filled up, then instead of the pushing, a *Nop* will be issued.
- There is no memory request to be served. This is most commonly seen because the L2 cache is a low-level storage, the request is infrequent and the L2 array is idle for most of the execution cycles.

The power-gate controllers that send control signals to L1 and L2 cache arrays in our technique impose a very small overhead in terms of the logical complexity involved: for L1 cache arrays, eight states are needed in total to model the length of a *Stall* between *warp scheduling* stage and *memory* stage in both *active→sleep* and *sleep→active* procedures. Therefore, a 1-bit input (indicating whether a *Stall* is currently issued), 1-bit output (*cs1*) sequential logic circuit with eight states is enough to control the transitions between *active* and *sleep*. To control the transitions between *active* and *off*, we can use 1-bit output for both *cs1* and *cs2* and 1-bit flag as input, indicating whether an SM finishes its workload and whether the grid initialization starts.

For L2 cache arrays, a 1-bit signal indicating whether the currently issued command is *Nop* will be taken as input and directly used to set and clear the 1-bit output (cs1).

C. Mode-transition Latency Hiding

There is a latency associated with the switching between the different leakage modes. In our design, mode-transition latency is restricted two cycles (design details in Section IVB). However, our microarchitectural design hides the mode-transition latencies very effectively, thereby avoiding any performance degradation. Following are the different cases of latency-hiding:

- *Hiding Transition Latency Between Off and Active in L1*

L1 cache goes from *active* to *off* when the SM it belongs to finishes all its work. So there is no performance degradation if this transition procedure takes a two-cycle delay because the L1 array has no incoming job to do. If a new program starts to be executed, the L1 cache can get woken up from *off* mode to *active* during the grid initialization, which takes far more than two cycles.

- *Hiding Transition Latency Between Sleep and Active in L1*

L1 goes to *sleep* mode from *active* once a *Stall* is issued. The power-gate controller knows whether a *Stall* is issued in directly from warp scheduler. As shown in Figure 2, there are three pipeline stages: *instruction dispatch*, *register file read*, and *execute* between *warp scheduling* and *memory* stage (in which L1 cache is accessed). This mechanism lets the power-gate controller of L1 send out the control signals after these three stages. As shown in Figure 4(a): if a *Stall* of three or less cycles is issued to the pipelines, the power-gate controller can accurately see the length of the *Stall*, so it chooses not to power-gate such a *Stall* because it is not adequate for the overhead of two mode transitions. If a *Stall* of four or more cycles is issued, the power-gate controller chooses to power-gate such a *Stall* without knowing the actual length of the stalled interval. However, the power-gate controller is still capable of signaling the L1 array into *active* mode in the last two cycles of the *Stall*, so that the L1 can be woken up in time. For a *Stall* longer than four cycles, the L1 stays in the *sleep* mode for at least one cycle, excluding the overhead of the two mode transitions. In these cycles, leakage power of L1 cache is effectively saved.

- *Hiding Transition Latency Between Sleep and Active in L2*

The activity of L2 and DRAM on the same channel is controlled by a memory controller. If the memory controller detects that its corresponding L2 array is not ready to process any data during a cycle, then a *Nop* command will be issued. Simply, as shown in Figure 4(b), if the current command issued by memory controller is *Nop*, the L2 array is turned into *sleep*; if the current command is not a *Nop*, the L2 is turned into *active* mode.

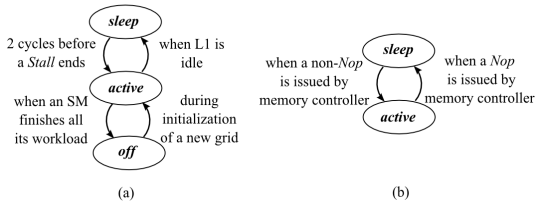

Figure 4. Power-gating strategy for (a) L1 cache, and (b) L2 cache

L2 array needs to start working on a non-*Nop* command followed by a *Nop*, yet the wakeup overhead is two cycles, so it appears that in this case, our strategy needs extra cycles in dealing with the wake-up process, which leads to performance degradation. However, the first step of L2 when it is back to *active* is queue-accessing (either fetching the memory request from memory controller-L2 queue, or fetching DRAM output from DRAM-L2 queue). Since the L2 array need not stay in *active* mode during the fetch period, the wake-up latency of the L2 can be effectively hidden.

IV. Experimental Set-up and Results

The experimental set-up is comprised of two aspects – the functional and timing simulation of GPU workloads using a GPU simulator; and the timing and power modeling of the caches. GPGPU-Sim [6], which is a detailed performance simulator for GPUs, is used in this work. The timing and the power modeling of the caches are done using a 32-nm IBM technology [7] node in HSPICE.

A. Benchmarks and Simulator Set-up

We experimented with sixteen benchmarks, eight of which are provided along with GPGPU-Sim, and the rest are from NVIDIA's CUDA software development kit (SDK) [8]. To emulate the architecture of Fermi [3] to the greatest extent feasible, we learned all the relevant configuration options that GPGPU-Sim provides and set them as in Table II. The number of cores we modeled is smaller than in real hardware due to the limitation of computing capability.

TABLE II. Architecture Comparison

Specification	Fermi	Our Model
Clock speed *(core)*	1400 MHz	325 MHz[a]
Clock speed *(memory)*	3700 MHz	L2: 650 MHz DRAM: 800 MHz
Number of SM's	16	30
Number of cores per SM	32	8
Register size per SM	128 KB	32 KB
L1 size per SM	48 KB	48 KB 4-way set assoc.
Shared memory size	16 KB	16 KB
Memory bus width per channel	64 bits	64 bits
Number of memory channels	6	8
L2 total size	768 KB	768 KB 8-way set assoc.

a. GPGPU-Sim "models the superpipelined stages in NVIDIA's SM running at a fast clock rate (1GHz+) with a single-slower pipeline stage running at 1/4 the frequency." [9]

The simulator was modified extensively to add the microarchitectural modeling proposed in this work. By default, the simulator provides only the total number of execution cycles, the total number of *Stall* and *Nop* cycles. To get more specific information, such as the length of each stalled interval for L1, the length of each idle period due to *Nop* in L2, and the time stamp of when each SM finishes its workload, which are essential for the construction of our power-gating scheme, we implement our own functions and add them into the source code of GPGPU-Sim.

TABLE III. LEAKAGE SAVING RESULTS

Benchmark	Abbr.	Total Exe. Cycles	L1 (30 arrays in total)			L2 (8 arrays in total)		Total Leakage Saving (%)
			Cache Arrays in sleep per cycle	*Cache Arrays in off per cycle*	*Leakage Saving (%)*	*Cache Arrays in sleep per cycle*	*Leakage Saving (%)*	
AES Cryptography	AES	33,214	3.6297	1.7589	15.12	7.7707	74.99	55.03
Ray Tracing	RAY	90,885	0.8289	2.0086	8.73	6.7314	64.96	46.21
Coulombic Potential	CP	166,020	0.0769	8.5749	28.35	7.9813	77.02	60.80
StoreGPU	STO	121,219	0.1011	0.5987	2.23	7.9587	76.80	51.94
3D Laplace Solver	LPS	140,016	0.1328	5.5171	18.46	6.3066	60.86	46.72
MUMmerGPU	MUM	749,289	7.5922	3.7711	31.92	6.3973	61.73	51.80
Bitonic Sort	BS	7,491	0.0068	29.0655	95.45	7.9972	77.17	83.27
CUDA Histogram	HIS	1,576,856	0.6779	0.0575	1.93	6.9941	67.49	45.64
Matrix Multiplication	MM	4,175	1.9986	11.0778	41.52	7.3266	70.70	60.97
Scalar Product	SP	53,941	7.0884	6.4558	39.44	6.3551	61.33	54.03
Simple Texture	ST	127,456	0.6746	0.5938	3.69	7.1130	68.64	46.99
Mersenne Twister	MT	3,897,714	0.4217	22.1734	73.89	6.3055	60.85	65.19
Neural Network Digit Recognition	NN	945,362	0.8574	0.9117	5.20	7.7715	75.00	51.73
CUDA Separable Convolution	CS	3,926,840	1.0660	0.0940	3.05	6.6311	63.99	43.68
Matrix Transpose	TRAN	8,348,870	6.2360	10.2288	49.63	6.2443	60.26	56.72
Breadth First Search	BFS	2,145,020	3.3234	9.1708	38.66	4.1507	40.05	39.59
Average			2.1695	7.0036	28.58	6.8772	66.36	53.77

B. *Modeling of Power-gated SRAM Array*

The power characterization of the L1 and L2 arrays are estimated using a power-gated SRAM array model of 32 bytes, the size of L2 cache line, built with IBM 32-nm technology [7]. According to the simulation of mode-transition activity in HSPICE, the latency of *off→active* transition, which is the worst case for mode-switching latency, is 1.954 ns. Assuming a clock frequency of 1 GHz, this translates to two cycles as the mode-transition latency, and 32 bytes as the size of cache array controlled by one sleep transistor pair. Another thing we noticed is that the power overhead caused by cache entering and exiting low-leakage mode is so small that it can be counteracted by the power saving fraction during the two-cycle mode transition. So the break-even period of our technique actually overlaps with the two mode transitions for getting in and getting out of the power-saving mode. After being normalized, the leakage power due to sub-threshold conduction in *active*, *sleep*, and *off* modes is respectively 100%, 22.8%, and 1.5% through HSPICE testing.

C. *Results*

The results of the experiments are shown in Table III. In addition to the total number of execution cycles, we respectively list the average number of L1 cache arrays that are solidly in *sleep* mode per cycle, the average number of L1 cache arrays that are solidly in *off* mode per cycle, and the average number of L2 cache arrays that are solidly in *sleep* mode per cycle based on our power-gating mechanism. Combined with the data in HSPICE modeling, the leakage saving from power-gating L1, L2, and both are also calculated. The row in the end of the table shows the average data over all 16 benchmarks. We can see that the portion of L1 arrays we can actually power-gate per cycle is less than that of L2 arrays. On average, there are more than 20% (7.0036 divided by 30) of L1 arrays that are in *off* mode. This phenomenon proves that it is meaningful to have an *off* mode in our technique to power-gate L1 arrays in early-finished SM's.

V. CONCLUSION

In this paper, we propose a technique to save the leakage power of L1 and L2 cache in GPU, based on power-gating. Three working modes for cache are designed to fit different occasions during GPU processing. Most important, we formalize the strategy of manipulating the sleep transistors according to how L1, L2, and other relevant devices of GPU function. An analysis is presented, in which we argue that our latency-hiding scheme ensures no negative impact on performance. Based on the simulation of 16 benchmarks, we show that the idle cycles of L1 and L2 cache take up a considerable portion of the total execution cycles, which reveals the potential of leakage power saving.

REFERENCES

[1] B. Burgess, B. Cohen, M. Denman, J. Dundas, D. Kaplan, J. Rupley, "Bobcat: AMD's Low-Power x86 Processor," in Proc. IEEE Micro, vol. 31, no. 2, pp. 16-25, 2011.

[2] H. Jiang, T.A. Piazza, "Intel Next Generation Microarchitecture Code Named SandyBridge," in Intel Developer Forum, 2010.

[3] NVIDIA Corporation, *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*, version 1.1, 2010.

[4] NVIDIA Corperation, *NVIDIA CUDA Programming Guide*, 1.1 edition, 2007.

[5] S. Rusu et al., "Power reduction techniques for an 8-core Xeon processor," in Proc. European Solid-State Circuits Conf., pp. 340-343, 2009.

[6] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, T.M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in Proc. IEEE Int'l Symp. on Perfomance Analysis of System and Software, pp. 163-174, 2009.

[7] The MOSIS Service, IBM 32SOI SPICE Model Parameters, http://www.mosis.com/Technical/Testdata/ibm-32soi-prm.html.

[8] NVIDIA Corporation. NVIDIA CUDA SDK code samples. http://developer.nvidia.com/cuda-cc-sdk-code-samples.

[9] GPGPU-Sim Manual, version 1.0, https://dev.ece.ubc.ca/projects/gpgpu-sim/browser/v2.x/doc.