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 of 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 were 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]](image-url)
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, $M_{d1}$ and $M_{d2}$, 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.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Control Signal</th>
<th>Mode Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>cs1 cs2</td>
<td>Working normally</td>
</tr>
<tr>
<td>sleep</td>
<td>0 1</td>
<td>×</td>
</tr>
<tr>
<td>off</td>
<td>0 0</td>
<td>×</td>
</tr>
</tbody>
</table>

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, 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 \textit{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 \textit{Stall} is issued. The power-gate controller knows whether a \textit{Stall} is issued in directly from warp scheduler. As shown in Figure 2, there are three pipeline stages: \textit{instruction dispatch}, \textit{register file read}, and \textit{execute} between warp scheduling and \textit{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 \textit{Stall} of three or less cycles is issued to the pipelines, the power-gate controller can accurately see the length of the \textit{Stall}, so it chooses not to power-gate such a \textit{Stall} because it is not adequate for the overhead of two mode transitions. If a \textit{Stall} of four or more cycles is issued, the power-gate controller chooses to power-gate such a \textit{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 \textit{Stall}, so that the L1 can be woken up in time. For a \textit{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 \textit{Nop} command will be issued. Simply, as shown in Figure 4(b), if the current command issued by memory controller is \textit{Nop}, the L2 array is turned into sleep; if the current command is not a \textit{Nop}, the L2 is turned into active mode.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{power-gate.png}
\caption{Power-gating strategy for (a) L1 cache, and (b) L2 cache}
\end{figure}

L2 array needs to start working on a non-\textit{Nop} command followed by a \textit{Nop}, yet the wake-up 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.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Specification} & \textbf{Fermi} & \textbf{Our Model} \\
\hline
Clock speed (core) & 1400 MHz & 325 MHz \footnote{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]} \\
\hline
Clock speed (memory) & 3700 MHz & L2: 650 MHz, DRAM: 800 MHz \\
\hline
Number of SM’s & 16 & 30 \\
\hline
Number of cores per SM & 32 & 8 \\
\hline
Register size per SM & 128 KB & 32 KB \\
\hline
L1 size per SM & 48 KB & 48 KB \footnote{4-way set assoc.} \\
\hline
Shared memory size & 16 KB & 16 KB \\
\hline
Memory bus width per channel & 64 bits & 64 bits \\
\hline
Number of memory channels & 6 & 8 \\
\hline
L2 total size & 768 KB & 768 KB \footnote{8-way set assoc.} \\
\hline
\end{tabular}
\caption{ARCHITECTURE COMPARISON}
\end{table}

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 \textit{Stall} and \textit{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 \textit{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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Abbrev.</th>
<th>Total Exec. Cycles</th>
<th>L1 (30 arrays in total)</th>
<th></th>
<th>L2 (8 arrays in total)</th>
<th></th>
<th>Total Leakage Saving (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cache Arrays in sleep per cycle</td>
<td>Cache Arrays in off per cycle</td>
<td>Leakage Saving (%)</td>
<td>Cache Arrays in sleep per cycle</td>
<td>Leakage Saving (%)</td>
<td>Cache Arrays in off per cycle</td>
</tr>
<tr>
<td>AES Cryptography</td>
<td>AES</td>
<td>33,214</td>
<td>3.6297</td>
<td>1.7589</td>
<td>15.12</td>
<td>7.7707</td>
<td>74.99</td>
</tr>
<tr>
<td>Ray Tracing</td>
<td>RAY</td>
<td>90,885</td>
<td>0.8289</td>
<td>2.0086</td>
<td>8.73</td>
<td>6.7314</td>
<td>64.96</td>
</tr>
<tr>
<td>Coulombic Potential</td>
<td>CP</td>
<td>166,020</td>
<td>0.0769</td>
<td>8.5749</td>
<td>28.35</td>
<td>7.9813</td>
<td>77.02</td>
</tr>
<tr>
<td>StoreGPU</td>
<td>STO</td>
<td>121,219</td>
<td>0.1011</td>
<td>0.5987</td>
<td>2.23</td>
<td>7.9587</td>
<td>76.80</td>
</tr>
<tr>
<td>3D Laplace Solver</td>
<td>LPS</td>
<td>140,016</td>
<td>0.1328</td>
<td>5.5171</td>
<td>18.46</td>
<td>6.3066</td>
<td>60.86</td>
</tr>
<tr>
<td>MUMmerGPU</td>
<td>MUM</td>
<td>749,289</td>
<td>7.5922</td>
<td>3.7711</td>
<td>31.92</td>
<td>6.3973</td>
<td>61.73</td>
</tr>
<tr>
<td>Bitonic Sort</td>
<td>BS</td>
<td>7,491</td>
<td>0.0068</td>
<td>29.0653</td>
<td>95.45</td>
<td>7.9972</td>
<td>77.17</td>
</tr>
<tr>
<td>CUDA Histogram</td>
<td>HIS</td>
<td>1,576,856</td>
<td>0.6779</td>
<td>0.0575</td>
<td>1.93</td>
<td>6.9941</td>
<td>67.49</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>MM</td>
<td>4,175</td>
<td>1.9986</td>
<td>11.0778</td>
<td>41.52</td>
<td>7.3266</td>
<td>70.70</td>
</tr>
<tr>
<td>Scalar Product</td>
<td>SP</td>
<td>53,941</td>
<td>7.0884</td>
<td>6.4558</td>
<td>39.44</td>
<td>6.3551</td>
<td>51.03</td>
</tr>
<tr>
<td>Simple Texture</td>
<td>ST</td>
<td>127,436</td>
<td>0.6746</td>
<td>0.5938</td>
<td>3.69</td>
<td>7.1130</td>
<td>68.64</td>
</tr>
<tr>
<td>Mersenne Twister</td>
<td>MT</td>
<td>3,897,714</td>
<td>0.4217</td>
<td>22.1734</td>
<td>73.89</td>
<td>6.3055</td>
<td>60.85</td>
</tr>
<tr>
<td>Neural Network Digit Recognition</td>
<td>NN</td>
<td>945,362</td>
<td>0.8574</td>
<td>0.9117</td>
<td>5.20</td>
<td>7.7715</td>
<td>75.00</td>
</tr>
<tr>
<td>CUDA Separable Convolution</td>
<td>CS</td>
<td>3,926,840</td>
<td>1.0660</td>
<td>0.0940</td>
<td>3.05</td>
<td>6.6311</td>
<td>63.99</td>
</tr>
<tr>
<td>Matrix Transpose</td>
<td>TRAN</td>
<td>8,348,870</td>
<td>6.2360</td>
<td>10.2288</td>
<td>49.63</td>
<td>6.2443</td>
<td>60.26</td>
</tr>
<tr>
<td>Breadth First Search</td>
<td>BFS</td>
<td>2,145,020</td>
<td>3.3234</td>
<td>0.5987</td>
<td>2.23</td>
<td>7.9587</td>
<td>76.80</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td>2.1695</td>
<td>7.0036</td>
<td>28.58</td>
<td>6.8772</td>
<td>66.36</td>
</tr>
</tbody>
</table>

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 break-even period of cache entering and exiting low-leakage mode per cycle, 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