Slumber: Static-Power Management for GPGPU Register Files

Devashree Tripathy
University of California, Riverside
Riverside, CA
devashree.tripathy@email.ucr.edu

Hadi Zamani
University of California, Riverside
Riverside, CA
hzama001@ucr.edu

Debiprasanna Sahoo
Indian Institute of Technology
Bhubaneswar
ds12@iitbbs.ac.in

Laxmi N. Bhuyan
University of California, Riverside
Riverside, CA
bhuyan@cs.ucr.edu

Manoranjan Satpathy
Indian Institute of Technology
Bhubaneswar
manoranjan@iitbbs.ac.in

ABSTRACT

The leakage power dissipation has become one of the major concerns with technology scaling. The GPGPU register file has grown in size over last decade in order to support the parallel execution of thousands of threads. Given that each thread has its own dedicated set of physical registers, these registers remain idle when corresponding threads go for long latency operation. Existing research shows that the leakage energy consumption of the register file can be reduced by under volting the idle registers to a data-retentive low-leakage voltage (Drowsy Voltage) to ensure that the data is not lost while not in use. In this paper, we develop a realistic model for determining the wake-up time of registers from various under-volting and power gating modes. Next, we propose a hybrid energy saving technique where a combination of power-gating and under-volting can be used to save optimum energy depending on the idle period of the registers with a negligible performance penalty. Our simulation shows that the hybrid energy-saving technique results in 94% leakage energy savings in register files on an average when compared with the conventional clock gating technique and 9% higher leakage energy saving compared to the state-of-art technique.

1 INTRODUCTION

One of the main limitations of the General Purpose Graphic Processor Units (GPGPUs) is the heterogeneity in workloads with different degrees of parallelism resulting in major resource under-utilization [2, 18]. This results in wasted energy especially in case of the register files, which constitute 18% of the total GPU chip power consumption and 32% of the streaming multi-processor's (SM) leakage power [8, 9]. This number has likely grown as the register file size has increased with the recent GPU architectures. One way to save energy is to power gate the unused components when they are not in use. However, the memory components like register files lose data when power-gated.

Authors of [1, 7] propose a drowsy mode, where the state of the memory cells or passive units are retained by lowering the voltage to a minimum voltage, called drowsy voltage. The active units or the functional units are turned off, i.e. read and write accesses cannot be performed. However, switching the component to deep sleep (drowsy) mode can result in performance penalty due to high wake-up latency. Under-volting is a technique to set an optimum voltage level (less than V_{DD}) that has less wake-up latency as compared to the deep sleep state that mitigates the performance penalties. However, the undervolting level of the functional elements will be different.

In this work, we analyze under-volting and power gating using a detailed device level modelling. Our detailed modelling of under-volting the registers using the trimodal switch [12] with HPSICE [15] shows that the drowsy voltage transition times can be as high as 13 clock cycles (cc), therefore, the performance penalty incurred can not be neglected. Hence, we argue that there is a need for an intermediate under-volting level with less transition time. We use two undervolting levels of 0.3 Volt and 0.9 Volt referred to as deep sleep and shallow sleep state respectively. The register is under-volted to a state retentive voltage if (a) next access to the register is a read access or (b) idle period length is larger than the corresponding energy break-even time (EBT). We use static compiler analysis to determine the type of register's next access and, thereby, predict the idle period length of the register. Then we determine the under-volting level based on the idle period length and EBT. Though all the prior approaches [1, 16] under-volt the registers associated with a warp, we show that the registers can instead be power-gated if the contents of the registers are not useful anymore. Since the register inter-access distance is about 789 clock cycles on an average [1], power-gating the idle registers which are waiting to be written leads to significant leakage energy benefits. The registers can be power-gated only under the following circumstances: (a) register is not allocated to any thread block (TB), (b) next pending access to the register is "Write", because the write will reset the register file contents. (c) The warp has finished executing the kernel. Since the register shall be re-allocated when next thread-block is allocated to the SM, the physical registers for the warp can be power-gated till all the other warps for the TBs finish executing the kernel. It should be noted that power-gating is only possible if the idle-period length is more than the power-gating break-even time.

In summary, we propose *Slumber*, a static-power management technique, for the register files of GPUs which uses various under-volting levels and power-gating at the run-time to save maximum leakage power. This work makes the following contributions:

ISLPED '20, August 10–12, 2020, Boston, MA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7053-0/20/08…$15.00
https://doi.org/10.1145/3370748.3406577

- We propose a novel power management technique named *"Slumber"* that enables multiple levels of under-volting as well as power-gating based on a static compiler analysis to determine the type of next register access. The length of the idle period is determined using our run-time technique.
- We implement the proposed techniques using the GPGPUSim and obtain a static energy saving of 94% with negligible performance degradation of 1.2% on an average. (Section-4)

The paper is organized as follows: Section 2 describes GPGPU register file architecture and motivation for this work. Section 3 discusses the device level power modeling of the GPU registers to determine the wake-up latency for power-gating and different under-volting levels. Section 4 discusses the proposed power-management technique for the GPU Register File. Our results are discussed in Section 5 and we conclude in Section 6.

2 MOTIVATION

The current state-of-art GPU design focuses on the performance and throughput of the applications being executed. The warp scheduler strives to make the best use of all the available resources by fast context-switching between the warps. When one warp is stalled, another warp swaps in and starts executing to boost the resource utilization. However, in most practical scenarios the GPGPU resources are under-utilized primarily due to imbalanced workload distribution. In the current work, we focus on the register file which consumes 32% of the Streaming Multi-Processor's (SM) total leakage power and hence, is the biggest contributor of the static power in a GPGPU core [9]. The behavior of the applications not only lead to resource under-utilization but also sub-optimal power usage. In the existing literature, there is almost no power-saving features besides the coarse-grain DVFS and clock-gating [4, 6].

Utilization is calculated as the ratio of the number of clock cycles the unit was accessed and the total clock cycles taken for the execution of the application. GPU registers are heavily under-utilized. The less the utilisation, more the opportunity of energy saving. If the accesses are less apart from each other with time, then the idle period length is less. The register file can be under-volted at different levels of granularity: whole register file, register file bank, registers allocated per thread, or individual 32-bit Register. A more fine-grained under-volting leads to maximum static power savings in the component but it comes at the cost of extra hardware complexity. Hence, the granularity at which under-volting needs to be done has a significant impact on the power usage and performance. Our experiments show that the average utilization of the Register File, register Bank, all registers associated with a warp, and each 128 Byte register are 32.7%, 16.3%, 3.9% and 0.2% respectively. This imbalance is mainly because of the load imbalance across the GPU cores, warp branch divergence, irregular memory access patterns and cache contention. In this work, we target on saving leakage energy at the register granularity as they have the least utilization.

The ***registers to be read*** in next accesses can not be power-gated as they are state-retentive, hence they are ***undervolted***. However, the ***registers to be written*** in the next access can be ***power-gated or undervolted*** depending on the idle period length. If the idleness period is more than the power gating break-even time then they are power-gated else they are undervolted.

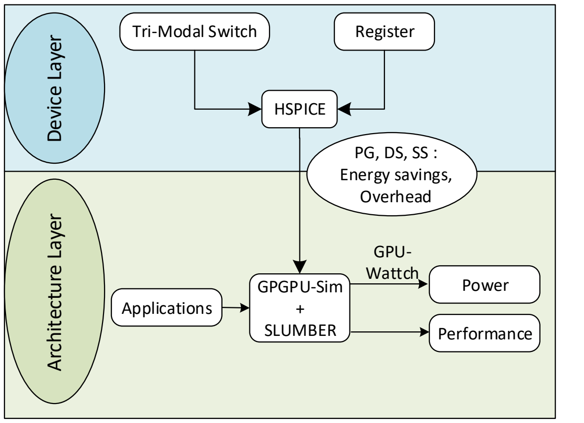

Figure 1: Slumber Overview showing Cross-layer methodology.

The main challenge in under-volting or power gating is determining the time to initiate the wake-up of the component; inefficient wake-up policy can lead to sleeping on the registers in the critical path of the GPU pipeline execution and hence, incur heavy performance penalty.

Conservative wake-up: Latency of the non-memory instructions execution (ALU-Int/Float, SFU) are deterministic. So, the output register is switched to "Sleep mode" depending on the instruction execution latency and the wake-up is initiated at time calculated as instruction latency - Wakeup latency of sleep mode. However, determining the wake-up initiation time for the memory instructions is challenging as the memory instruction execution latency is non-deterministic and depends on various factors like mshr-free-entries, L1 hit/miss, L2 miss, DRAM access and queuing delay. Hence, the status of the memory instruction is tracked [L1 hit/miss, L2 miss] and the memory operation latency is assumed to be the access latency of L1 (for L1 hit), access latency of L2 (for L1 miss), or access latency of DRAM (for L2 miss). Accordingly, wake-up is initiated at the minimum memory access latency - wakeup latency of sleep mode.

Slumber Overview: The overview of the proposed static power reduction technique has been shown in Figure 1. The device layer implementation (details in Section 3) consists of the accurate modeling of the different power reduction modes (PG: Power gating, DS: Deep Sleep and SS: Shallow Sleep) and the associated leakage energy savings and overhead in terms of wake-up latency to the ON mode using the Tri-modal switch [12] and 128 Byte (32 set of 6T SRAM) register using 45 nm technology. The architecture layer consists of modifying the baseline GPGPU-Sim [3] to incorporate the architectural modifications of Slumber and integrate the different power modes and overheads from the device layer simulation. We evaluate various applications from commonly used GPU Benchmark suites [3, 5, 11, 13] using the modified GPGPU-Sim for the power and performance results.

3 STATIC POWER REDUCTION METRICS ESTIMATION

Static power can be reduced by two techniques: **Power gating (PG)** and **Under-volting (UV)**. We shall discuss the static power savings metrics: performance and power overhead associated with each technique in the following section. Figure 2 shows a tri-modal switch used to apply different voltages (Virtual V_{DD}) across the target register [12]. For power-gating, the Virtual V_{DD} is set to "Zero" such that the voltage drop across the register is negligible and we have maximum leakage power savings. For under-volting, Virtual V_{DD} is set to a voltage such that (0 < Virtual V_{DD} < V_{DD}); the lower the Virtual V_{DD}, the more is the leakage power saving. The output of the tri-modal switch (Virtual V_{DD}) is controlled by the signals "Sleep" and "Drowsy" and the width of the transistor MS as described in [12].

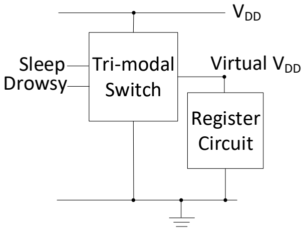

Figure 2: Varying the voltage across the register using Tri-modal switch.

When the target register is idle, a *sleep* signal = 1 and *drowsy* signal = 0 are applied to the power gating switch so that the Virtual V_{DD} is set to *zero*; the target circuit is turned off (OFF state). Alternatively, while waking up the target circuit, the sleep signal = 0 so as to set Virtual V_{DD} to V_{DD}. However, power-gating introduces the energy overhead due to the switching of transistors to OFF/sleep state. This implies that the target circuit should now sleep for at least break-even time ($t_{breakeven}$) to compensate for the energy overhead incurred ($E_{overhead}$). For example, Figure-3 shows the voltage transition when the circuit is switched from ON (Voltage = V_{DD}) to OFF (Voltage = 0) at time t_2 and back to ON at time t_4. The target circuit does not switch to OFF state immediately, the voltage across the capacitive load in the target circuit completely discharges at t_3. The circuit stays in OFF state from t_3 till t_4 and then the wake up is initiated at t_4. As seen in the ON to OFF transition, the circuit capacitance charges to V_{DD} slowly and the circuit is completely ON at t_5. The break even time can calculated by taking the difference of t_4 and t_2, i.e., $t_{breakeven} = t_4 - t_2$; $t_{breakeven}$ is defined as the minimum time period the target circuit should sleep in-order to save power. At $t_{breakeven}$, energy saved (E_{saved}) is same as energy overhead ($E_{overhead}$). $t_{detect}(= t_1)$ is the time taken by the control circuit to make a decision to power gate the target circuit. Finally, $t_{fall}(= t_3 - t_2)$ is the transition time to turn-off and $t_{wake-up}(= t_5 - t_4)$ is the transition time to wake up the target circuit.

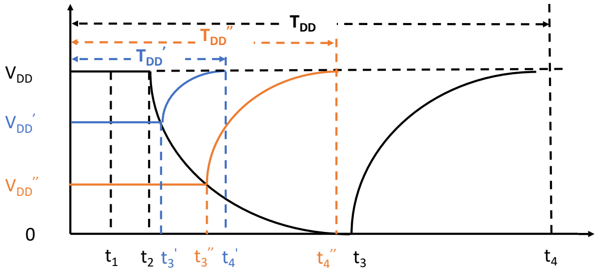

Figure 3: Target circuit state transition during power gating interval.

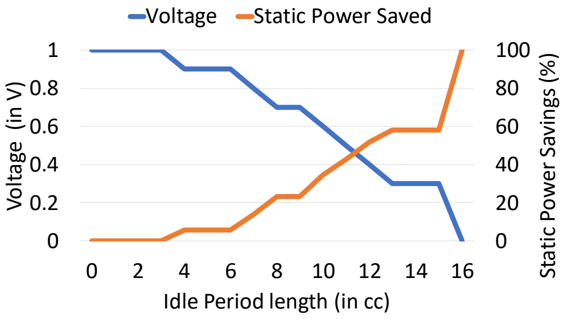

Figure 4: Determination of optimal under-volting level and associated static power savings based on the idle period length.

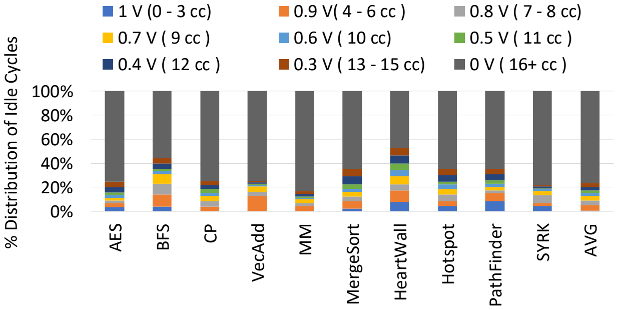

Figure 5: Idle Period Distribution based on length

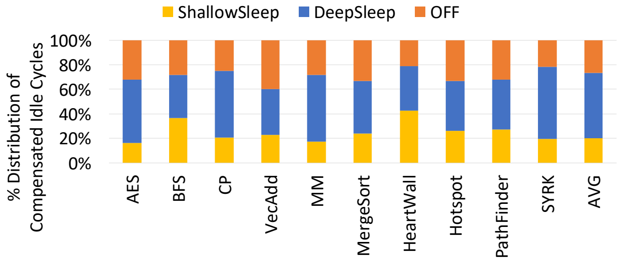

Figure 6: Idle Period Distribution across Sleep States

On the other hand, if we undervolt to a level V_{DD}' as shown in Figure 3, the transition times (t_3'-t_2') and (t_5'-t_4') are much shorter. Then the required idle period (T_{DD}') can be considerably reduced at the expense of a little more energy. Target circuit state transition during power gating interval is shown in Figure-3. We obtain the values of t_{detect}, t_{fall} and $t_{wake-up}$ for the components using HSPICE simulations and calculate the break even time (T_{DD}) for power-gating, T_{DD}' for undervolting to level V_{DD}' and T_{DD}'' for undervolting to level V_{DD}''. The undervolting level V_{DD}' is termed as shallow sleep state and V_{DD}'' is called deep sleep state in our work.

The two main considerations for any static power reduction technique are: (a) typically, there is a performance overhead associated with transitioning between different low-power states; ideally, the transition overhead should not affect the overall performance of the application, (b) the storage structures should be state-retentive at the low power mode to preserve the content in such a way that it is ready to use when the circuit is powered back to ON state.

Table 1: Power management modes for Register File.

State	Wakeup Delay (in clock cycles)	Static Power
ON	0	100%
Shallow Sleep (SS)	4	94%
Deep Sleep (DS)	13	42%
Power Gating (PG)	16	0%

We explore the state transition delay of the registers at various voltage levels using the methodology described above. The transition time from different voltages to ON voltage (1V) and the static power saving for the registers as a percentage of the maximum static power consumed in the ON state are shown in Figure 4. Primary Y -axis shows voltage applied across the Register (Virtual V_{DD}) for optimal static power savings. Secondary Y -axis shows static power consumed by the registers as a percentage of the maximum static power consumed while at V_{DD} in ON state. The drowsy state, considered in the previous papers [1, 7], corresponds to a voltage of 0.3 V that takes an idle period of 13 cycles to be enabled. The % distribution of the idle cycles based on their length are shown in Figure 5. Over 75% of the idle cycles have a length of more than 16 cc and rest of the idle cycles are evenly divided across the voltage categories of 0.3 V - 0.9 V. Although we can theoretically put a register to power gating after 16 cycles, that is not possible in case the next access is a register read. We have to put it in deep sleep mode. Also, incorporating multiple sleep states shall incur high area overhead, hence, we define a shallow sleep state (SS) that works at 0.9 V, takes only 4 cycles but can save 6% static power, compared to when the register is on. We call the drowsy state as the deep sleep (DS) state.

Table 1 shows different power management states used in SLUMBER, their wake-up delays and static power consumed. The voltage across the component in Figure 2 for the ON state is 1V, shallow sleep state is 0.9V, deep sleep state is 0.3V and power gating or OFF state is 0V. Considering these states, the latency distribution for different applications is shown in Figure 6. The distribution gives an idea about the possible energy saving in Slumber. On a average, 26.3% of the compensated idle cycles are in OFF state, 53.5% are in Deep Sleep State and 20.1% are in shallow sleep state. Even though the idle cycle length of over 75% of the idle cycles is more than 16 cycles, but only 26.3% can be power gated, as for the read accesses, the register cannot be power-gated.

4 SLUMBER DESIGN

4.1 Power-Gating and Under-volting Opportunities in Register File (RF).

Since the idle cycles for register read cannot be power-gated irrespective of the idle cycle length, we insert 1 bit flags per register at the compile time stating the type of the next access to the operand registers is read/write. We observed that registers can be undervolted and power-gated in primarily three different conditions: **(1) Register write waiting on a compute operation:** The compute operations have a deterministic execution latency once they enter the functional unit pipeline. For example, the latency of Integer multiplication operation is 6 cc [3] (predicted idle period of the corresponding register is 6 cc), hence, the corresponding registers are switched to the shallow sleep (SS) state where there is a energy saving of 2 cc after accounting for the switching overhead to SS state. Similarly, in case of integer division operation where the latency is 153 cc [3], the registers are switched to OFF state. There can be additional wait periods at the collector unit prior to the execution in functional unit as only one out of multiple ready warps can be issued per cycle to initiate execution in the functional unit. The warps also wait at the dispatch stage if there is a write conflict at the result bus since the result bus is shared between all the compute functional units. We do not account for these delays as we use a conservative wake-up policy. **(2) Register write waiting on a memory operation:** At the write back stage, the scoreboard releases the output registers for a warp after writing into the registers for the finished warp. A 2 bit-counter per warp (MSHR Tracker) can be used to keep track of the L1-hit, L1-miss and L2-miss for a warp. Upon a L1-hit, counter is set to 0, in case of a L1-miss the counter is set to 1 and in case of a L2 miss it is set to 2. Usually, the hit access latency of L2 cache and DRAM is in order of hundred cycles. Since the access latency for L2 and DRAM cache is much higher than the wake-up latency of the OFF state, the registers are switched OFF in case of a L1-miss or L2-miss. No state change is there in case of LI-hit. The counter values are used to determine the minimum memory access latency. The idle period of the corresponding registers is predicted to be same as this minimum memory access latency. The register is woken up after a time = *minimum memory access latency - OFF state latency*. The minimum memory access latency of L1, L2 and DRAM was determined using microbenchmarking [17]. **(3) Register idle as the warp has finished executing the kernel:** The registers associated with a warp become idle after the warp has finished executing the kernel and waiting for the other warps in the same thread-Block to finish execution. The physical registers for the warp can be power-gated till all the warps for the thread-blocks finish executing the kernel. This is because the register contents shall be re-allocated when next thread-block is allocated to the Streaming Multiprocessor (SM). Another scenario when the registers are idle is when the last batch of thread-blocks are executing in another SM. Since all the thread-blocks in the current SM have finished execution, they can be power-gated.

4.2 Power Mode Transition

The Finite State Machine for the various power mode transitions in Slumber is shown in Figure 7.

When the Kernel starts execution, all the registers are in OFF state (T8). The thread-block scheduler allocates the physical registers in an Streaming Multiprocessor(SM) to a Thread-block. Once the instruction buffer issues the Instruction Fetch request (PC_1) for a warp, the registers used by PC_1 in the specific warp are transitioned to *Shallow Sleep (SS)* state through T1. The default warp scheduling policy is modified to determine the three future warps to be scheduled in-addition to the current scheduled warp in order to hide the SS latency of 4 clock cycles. The registers for the determined future warps are switched from the SS to *ON* state (T4). If the access to register is read, the register provides the values to the warp in the collector unit and switches (T5) back to any of the 3 states SS (T5), DS (T3) or OFF (T6) depending on the idle period length predicted at run-time and type of the next access to the register (read/write) determined by static compiler analysis.

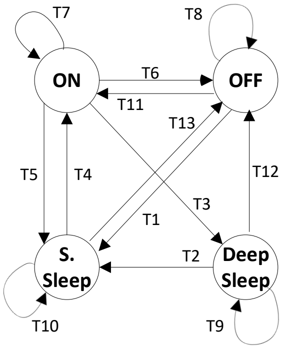

Figure 7: FSM of the Slumber Algorithm

From the ON state in Figure 7, it is switched to (a) SS through T5: If the warp using the register is not among the ready warps in the warp scheduler but corresponding instruction or Program Counter (PC) exists in the instruction buffer or L1 Instruction Cache. (b) DS through T3: If condition for SS state are not met and type of next access is "read",(c) OFF through T6: If condition for SS state are not met and type of next access is "write". The register resumes to stay in its current state (T10/T9/T8) till the instruction buffer (for DS and OFF)/warp scheduler(for SS) notifies it about the time of the future register access. However, if the access to the register is write, then it remains in the ON state (T7) till the latency of the pending register write operation is determined. If the write operation is an ALU operation, the register is switched to SS (T5), DS (T1) or OFF (T6) state depending on the deterministic latency of the operation. For example, the latency of an Integer Add is 6 clock cycle, hence, the register will be switched to SS state . However, if the write operation is load operation, then the register stays in ON state till the L1 cache access status is determined. In case the L1 cache access is a hit, the register is updated; In case of a L1 miss, the register transitions (T6) into OFF state since the L2 access latency is in order hundreds of clock cycles. The register is transitioned (T11) to ON state before the memory request completes based on the technique mentioned in Section 4.1. It may be reminded here that the previous warped register-file technique used to keep a register ON until the write is performed, thus wasting a lot of energy. When a warp finishes execution of the kernel, all the registers are transitioned from any other state (T6, T12, T13) to OFF state.

4.3 Re-Architecting Register File for Slumber

In this section, we illustrate how the baseline register file architecture is modified to incorporate SLUMBER power modes shown in Figure 8. The power modes are enabled by connecting each 128 byte register to a multiplexer (mux). Each register is associated with a 10-bit counter to wake-up after the predicted idle period since the maximum idle period length results from DRAM access is 400-600 cc [3]. The mux has 4 input signals: Vdd (Supply Voltage), Deep Sleep Voltage, Shallow Sleep Voltage (Both generated by varying the configuration of the Trimodal switch), 0 (Ground) ; 2-bit control signal (from the Slumber Control Logic) to select between 4 different power modes. The Slumber Control Logic decides the Power mode for a given register based on the input from the MSHR tracker (L1 or L2 miss), Instruction Buffer (next registers to be in the Shallow Sleep mode), Warp Scheduler (registers for the future warps in On mode), Operand collector and Thread-Block Scheduler. We introduce a hardware fetch-unit-list to determine whether a PC exists in L1I. The 16-entry fetch-unit-list to keep track of the starting PC address in each cache line, where each entry is 8 Bytes (PC size). Each cache line in L1I serves 16 consecutive PC requests. If there is an L1I hit, the instruction gets loaded into instruction buffer in 2 clock cycles. However, wake-up delay from OFF or DS state is much higher. So, we use a similar mechanism as in Section 4.1.2 to determine the idle period length of the registers and initiate their transition into SS state.

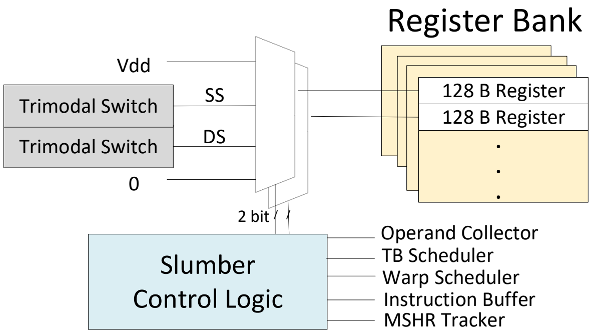

Figure 8: Illustration of GPU Registers connected to the voltage rail and tri-modal switch output using mux and Slumber control Logic in order to enable Power Gating (V = 0), Under-volting modes(SS: Shallow Sleep, DS: Deep Sleep) and ON state (V = V_{dd}).

Overheads: The counters associated with each 128 Byte register introduces the largest area overhead of 10 bit x 1024 i.e. 1.25 KB. The counters along with area overhead of Slumber Control Logic, MSHR Tracker (16 Bytes), multiplexers, tri-modal switches and the fetch-unit-list (128 Bytes) constitutes around 4% of the total register file size and less than 1% of the register file leakage power. The power overhead of these components is included in our results. The area and power overhead are calculated using HSPICE and CACTI v5.3 [14]. These overheads will reduce when we go for a coarser grained power saving technique at whole register-file, bank or warp granularity. However, since the utilization of these components is much high compared with the individual 128 Bytes register, the corresponding leakage energy gain is likely to reduce.

5 RESULTS AND DISCUSSION

Methodology: The proposed power management technique for saving leakage energy was evaluated using GPGPU-Sim v3.2.1 [3] based on Fermi-like configuration with 15 SM. Each SM comprises of 1024 registers of 128 Bytes each divided into 4 banks. Each SM has two warp schedulers using two-level warp scheduling policy [10]. In our experiments, 10 benchmarks used were selected from 4 different benchmark suites ISPASS [3], Nvidia CUDA SDK [11], Rodinia [5] and Polybench [13]. We enabled PTXPlus for accurate register file evaluations.

Comparison with prior work: Leakage energy saving of Slumber is 94% and Warped Register File (WRF) is 85% compared to the baseline is shown in Figure 9. The power gain of Slumber as compared to WRF come from the following scenarios: (1) During a long latency operation, the output register is in ON state due to

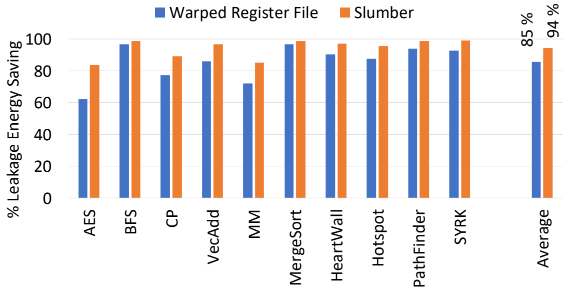

Figure 9: Leakage Energy Savings

the write-conservative policy of WRF. However, in Slumber the output register is power-gated during this time interval. (2) When a warp finishes executing the kernel, the registers associated with the warp and all the other warps in the same thread-block are in drowsy state. However, our technique leverages these power saving opportunities and power gates the registers of the finished warps. (3) The inter-access distance between subsequent accesses to the same register is very high, around 789 clock cycles on an average [1]. In the WRF, the registers are switched to drowsy mode after each access. However, slumber classifies the registers depending on the "next access type". If the next access is a register write (write register), then it is switched to OFF mode in case the re-use distance is high. Since the drowsy voltage still consumes a non-negligible portion of the register leakage power, Slumber outperforms WRF in terms of the leakage power savings by switching OFF the "write registers" aggressively.

Leakage energy efficiency improvement: The benchmarks AES, CP, MM and VecAdd have significantly high power saving compared to WRF. The additional power saving for these benchmarks are 23.2%, 14%, 14%, and 10.8%, respectively. The benchmarks with more number of power-gated idle cycles (idle cycle length more than 16 clock cycles) tend to fair better with Slumber. On the contrary, when the frequency of the shorter idle periods in a benchmark is high, it is switched to shallow sleep state (consume more leakage power than deep sleep), thereby reducing the energy gains. This leakage energy saving in register file accounts for about 30% leakage energy saving in streaming multiprocessor and 5% total power saving in the whole GPGPU assuming the dynamic power to leakage power ratio to be 2 to 1.

Performance Penalty: Average performance penalty of Slumber is 1.2% compared to the baseline. In Slumber, when the average number of ready warps in the warp scheduler is less than 3, then the performance loss is experienced as the issued warp has to wait for the register to wake-up from the Shallow sleep state.

6 CONCLUSION

We propose SLUMBER multi-power mode management technique to efficiently reduce the static power consumption of GPU register file unit. The novel approach exploits their under utilization behavior and non-state retentive requirement of the register writes. *SLUMBER* employs three static power reduction modes, each with different static power saving abilities and wake-up overheads in order to reduce power consumption of idle registers effectively. Our experimental results show that by aggressively switching the registers to low-leakage modes, SLUMBER saves static energy by about 94% compared to baseline; it saves 9% more energy than Warped Register File work with minimal performance overhead of about 1.2% compared to baseline. We conclude that SLUMBER provides an effective framework for reducing static power consumption in modern GPU Register Files.

ACKNOWLEDGMENT

This work is supported by NSF Grant 1513201 and Govt. of India SPARC grant P712. The authors would like to thank the anonymous reviewers for their invaluable comments and suggestions.

REFERENCES

[1] Mohammad Abdel-Majeed and Murali Annavaram. 2013. Warped register file: A power efficient register file for GPGPUs . In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 412–423.

[2] AmirAli Abdolrashidi, Devashree Tripathy, Mehmet Esat Belviranli, Laxmi Narayan Bhuyan, and Daniel Wong. 2017. Wireframe: Supporting data-dependent parallelism through dependency graph execution in gpus. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 600–611.

[3] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*. IEEE, 163–174.

[4] Juan M Cebri'n, Gines D Guerrero, and Jose M Garcia. 2012. Energy efficiency analysis of GPUs. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 1014–1022.

[5] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 44–54.

[6] Hodjat Asghari Esfeden, Farzad Khorasani, Hyeran Jeon, Daniel Wong, and Nael Abu-Ghazaleh. 2019. CORF: Coalescing Operand Register File for GPUs. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[7] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. 2002. Drowsy caches: simple techniques for reducing leakage power. In *ACM SIGARCH Computer Architecture News*, Vol. 30. IEEE Computer Society, 148–157.

[8] H. Jeon, H. A. Esfeden, N. B. Abu-Ghazaleh, D. Wong, and S. Elango. 2019. Locality-Aware GPU Register File. *IEEE Computer Architecture Letters* 18, 2 (2019), 153–156.

[9] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. [n. d.]. GPUWattch: Enabling Energy Optimizations in GPGPUs. *SIGARCH Comput. Archit. News 2013* ([n. d.]).

[10] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, Rustam Miftakhutdinov, Onur Mutlu, and Yale N Patt. 2011. Improving GPU performance via large warps and two-level warp scheduling . In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 308–317.

[11] NVIDIA. 2007. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit. (2007).

[12] Ehsan Pakbaznia and Massoud Pedram. 2009. Design and application of multimodal power gating structures. In *2009 10th International Symposium on Quality Electronic Design*. IEEE, 120–126.

[13] Louis-Noël Pouchet. 2012. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench* (2012).

[14] Premkishore Shivakumar and Norman P Jouppi. 2001. Cacti 3.0: An integrated cache timing, power, and area model. (2001).

[15] I Synopsys. 2010. HSPICE User's Manual: Simulation and Analysis. *Synopsys Inc, California* (2010).

[16] Xin Wang and Wei Zhang. 2017. Drowsy register files for reducing gpu leakage energy. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 632–639.

[17] Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. 2010. Demystifying GPU microarchitecture through microbenchmarking. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 235–246.

[18] Hadi Zamani, Yuanlai Liu, Devashree Tripathy, Laxmi Bhuyan, and Zizhong Chen. 2019. GreenMM: energy efficient GPU matrix multiplication through undervolting. In *Proceedings of the ACM International Conference on Supercomputing*. 308–318.