Using Synchronization Stalls in Power-aware Accelerators

Ali Jooya and Amirali Baniasadi
Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C., Canada
E-mail:{jooya, amirali}@ece.uvic.ca

***Abstract*—GPUs spend significant time on synchronization stalls. Such stalls provide ample opportunity to save leakage energy in GPU structures left idle during such periods. In this paper we focus on the register file structure of NVIDIA GPUs and introduce sync-aware low leakage solutions to reduce power. Accordingly, we show that applying the power gating technique to the register file during synchronization stalls can improve power efficiency without considerable performance loss. To this end, we equip the register file with two leakage power saving modes with different levels of power saving and wakeup latencies.**

I. Introduction

GPUs are designed to achieve high and steady throughput. This is achieved by employing hundreds of processing elements and running thousands of concurrent threads resulting in peak power dissipation up to 300 Watt [1]. Moreover, the current trend in GPU design is towards increasing the number of processing elements and concurrent threads, further increasing power dissipation. Maintaining a high number of concurrent threads requires using very large register files. Therefore, GPUs use highly banked register files to provide sufficient bandwidth for concurrent threads.

Previous studies have shown that the register file is one of the critical components in the overall energy consumption [2], [3]. However, the register file consumes static power without any performance contribution during periods when threads stall on cache misses or synchronization instructions. In this work we address this inefficiency and propose to apply power gating technique to the unused portions of the register file, which are associated with the threads stalled on synchronization instructions.

Kernels executing on GPUs are composed of thousands of threads which are grouped into blocks and assigned to different shaders. Each thread block is divided into groups of 32 threads, referred to as warps in the NVIDIA terminology. The warp scheduler executes warps in a round robin manner on the SIMD pipeline, which is composed of 32 Processing Elements (PEs). In a given clock cycle, all PEs execute the same instruction for the threads in a warp. The compiler assigns a fix number of registers to each thread and each thread accesses only specific entries of the register file during the entire kernel execution. As threads in a warp proceed together, each warp can access certain entries of each bank. Therefore, when a warp stalls on a synchronization instruction, the register file portions associated with the stalled warp are no longer accessed. The idle section of the register file can be placed into low leakage modes to improve the power efficiency of the GPU.

Our solution, $Sync-aware\ low\ leakage\ Register\ File$ or SLG, uses two low leakage modes, one with low level of power savings and low weakup latency ($SLG1$) and the other with higher leakage power savings and higher wakeup latency ($SLG2$). Based on the predicted register file idle time we place the appropriate section of all register banks into one of the low leakage modes. We evaluate our solution under different timing scenarios reporting both power and performance.

The rest of the paper is organized as follows: Section II presents background. Section III discusses SLG. Section IV presents the experimental setup and results. In Section V we review related works and finally Section VI offers conclusions.

II. Background

A. GPU Architecture

The GPU used in this paper is composed of 30 shaders, which are connected to the memory controllers through interconnection network. Memory controllers provide access to off-chip DRAM memories. Shaders contain 32 PEs, forming a wide SIMD pipeline executing 32 parallel threads simultaneously. The memory system dedicated to each shader (level one data, instruction, constant and texture caches and shared memory) and global memory, which is shared among all shaders, provide instructions and data to execute GPU kernels.

B. Register File Structure

In this section we present more details on the GPU register file architecture [4]. As Figure 1 shows, each shader contains a register address unit that provides the register file an instruction and its operands. The collector unit checks the type of the instruction to determine the execution unit that can execute the instruction. The collector unit sends the requests to the bank request arbitration unit. The arbitration unit sends the read request for each operand to the proper bank. Each bank is

978-3-9815370-0-0/DATE13/©2013 EDAA

128-bit wide and accommodates four 32-bit operands and has one read and one write port.

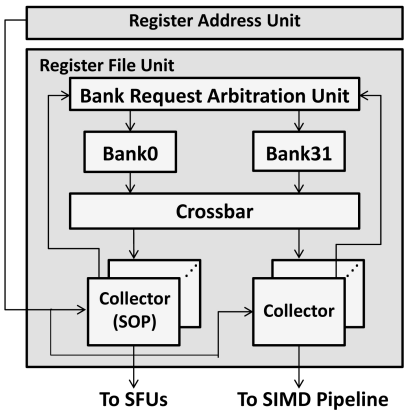

Fig. 1: Shader register file structure.

To provide all the required operands of an instruction in a single clock cycle, each bank stores the same-named operand for four adjacent threads. Therefore, the arbitration unit sends read requests to multiple banks to provide all requested operands of an instruction. Figure 2 shows the organization of operands in register banks. As the figure depicts, we assume that each SM holds 96 threads which are divided into three warps. Each thread shares with its neighbor threads a portion of the register file, depending on the register usage of the kernel. The operands of the threads sharing the same register bank are stored in the same entries of the register bank. Therefore, in a single clock cycle, all threads read the required operands from the banks or write back the results. For example, in Figure 2, threads 4 to 7 share register banks 4 to 7. To execute an instruction with three source operands, the first operand of each thread is read from Bank4 and the second operands are read from Bank5 and the third operands are read from Bank6.

Register banks are connected to the operand collector units via crossbar. There are two types of collectors, one for special operations (SOP) and one for the rest of operations. The number of collector units depends on the number of SFUs and the SIMD pipeline width. The crossbar outputs the operands from each bank to different operand collectors. When the operand collector receives all requested operands, it dispatches operands to execution units. When instruction execution is done, the results are written back into banks storing destination operands. Note that,

C. Power Gating

Power gating is a well-known technique used to reduce leakage power of circuits in their idle mode. This is achieved by using a high threshold voltage transistor (T_{Sleep} in Figure 3(b)) between the actual ground and the circuit ground (called virtual ground). When the circuit is in standby state, by turning off the sleep transistor, the virtual ground is charged up to a value close to VDD. This cuts off the leakage path between the supply and ground and the data in storage elements is

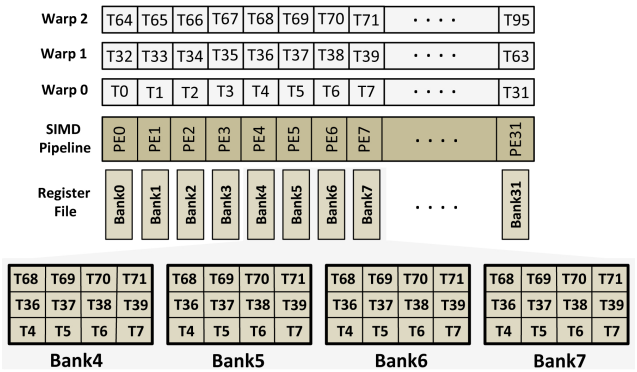

Fig. 2: Data organization in register banks.

lost. By turning the sleep transistor on, the virtual ground is restored to its nominal voltage which requires additional time to discharge the virtual ground node. The extra time required is referred to as wakeup latency.

III. SLG

The shader scheduler selects warps to execute on the SIMD pipeline. All warps of a thread block assigned to a shader progress with the same speed until the SIMD pipeline reaches a diverging control flow or a memory access instruction. Such instructions can harm performance if they result in threads having different control flow paths or different memory response times [5]. This also reduces SIMD pipeline utilization. Therefore, it is quite common that after a while, some warps fall behind other warps in a shader. Considering the fact that warps may progress with different speeds, the programmer needs to use synchronization instructions whenever necessary to sync all warps of a thread block.

When threads of a warp execute a synchronization instruction, they have to wait until all warps of the thread block reach the same synchronization instruction. Based on how far behind (in terms of instruction count) other warps are, a warp may have to wait for several cycles. When the last warp of the thread block executes the synchronization instruction, all waiting warps can continue their execution from the next cycle. Based on the order in which warps reach the synchronization instruction they experience different stall durations. Based on the predicted stall duration we put the register file assigned to idle warps into one of power saving modes. Both $SLG1$ and $SLG2$ modes maintain register file contents and differ in the leakage power reduction level and the wakeup latency. $SLG1$ saves less power while $SLG2$ has higher leakage power reduction. We evaluate various wakeup latencies for each power saving mode. The wakeup latency of $SLG1$ varies between 1 and 24 cycles and $SLG2$'s wakeup latency varies between 2 and 35 cycles in this study.

When a warp executes a synchronization instruction, initially, we assume a long stall and put its register file in the $SLG2$ mode. When the last warp of the thread block executes the synchronization instruction the warps of the thread block have been synchronized and they can continue their execution

from the next cycle. Therefore, anticipating an end to the idle period, we change all warps' leakage saving modes from $SLG2$ to $SLG1$.

Note that as the first warp is wakened the rest of the warps put their register file into $SLG1$ mode requiring a period equal to the difference of $SLG2$ and $SLG1$ latencies to wakeup. When the next (second) warp is selected for execution, we signal its register file to change its mode from $SLG1$ to normal which requires only cycles equal to $SLG1$ wakeup latency (part of this latency may be covered by the latency of pipeline stages between fetch and register file access).

A. Hardware Implementation

To put the register file of the warp waiting on a synchronization stall (or S-stall) in one of the SLG modes, we select the corresponding entries of the register file (using the warp ID) and apply SLG to reduce leakage power dissipation.

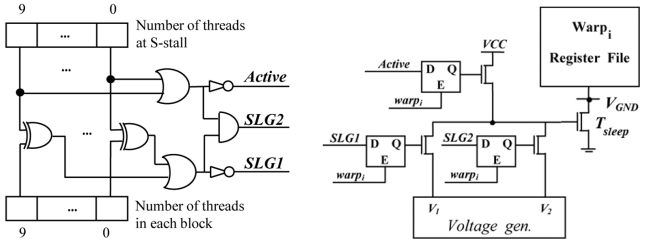

Fig. 3: (a) The control logic, which selects the proper mode for the register file and (b) Intermediate power gating circuit.

Figure 3(a) shows the circuit and control logic we use to implement SLG. There is a register in each shader which keeps track of the number of threads which are already waiting at an S-stall (Figure 3(a)). This is a 10-bit register as the maximum number of threads per shader is 1024 in this study. The control logic shown in Figure 3(a) uses the bits of this register to select the appropriate register mode for the corresponding warp. If the value of this register is zero the register file is in active mode (no thread is waiting on the S-stall). If there are non-zero bits in this register and the value is less than the number stored in the register that shows the maximum number of threads in each block, then there are warps waiting on the S-stall and the register file of the corresponding warp should be placed in $SLG2$ mode. When this register's value is equal to the maximum number of threads in each block, then all the threads of the corresponding thread block have executed the synchronization instruction and the control logic should bring all warps into the $SLG1$ mode. Figure 3(b) shows the intermediate power gating circuit. The register mode signals issued from control logic go through the D flip-flops to select the appropriate register mode. D flip-flops' enable signals is the same signal that shader control unit uses to access the appropriate register file entry for each warp. The details of the voltage generator unit can be found in [6]. The area overhead of the circuit is less than 0.5%. We also assume negligible power overhead associated with the circuit.

IV. Experimental Setup and Results

In this section we present the experimental setup and report leakage power reduction and performance. We analyze SLG's sensitivity to the wakeup latency of $SLG1$ and $SLG2$ modes.

TABLE I: GPU configuration.

Parameter	Value	Parameter	Value
Number of shaders	30	SIMD pipline width	32
Shader clock frequency	1.3 GHZ	Warp width	32
Max thread per shader	1024	Scheduling	PDOM

A. GPU Configuration and benchmark details

We use the GPGPU-Sim [7] simulator with the system configuration shown in Table I. For our evaluations we simulated the execution of benchmarks listed in Table II. The table also reports the number of kernels, thread blocks, warps in each thread block and synchronization instructions each thread executes during runtime.

TABLE II: Benchmarks' characteristics.

Benchmark	Num. of kernels	Thread Blocks	Warps/ Block	Stalls at sync.
Dynproc [4]	52	52	16	78
Scan [4]	1	64	8	19
Gaussian [4]	16	16	8	4
LPS [8]	1	1024	4	2048
Backprop [4]	2	1024	8	4
Srad [9]	4	64	8	4

B. Results

To study the sensitivity of SLG to wakeup latencies and the number of pipeline stages we use the $x.y.z$ notation and present different scenarios. Under this notation x and y represent $SLG1$ and $SLG2$ wakeup latencies, respectively. z represents the number of pipeline stages between instruction fetch and register file access. In Table III we report how relative values of x, y and z affect performance penalties associated with $SLG1$ and $SLG2$. As the table shows, $SLG2$'s latency is always larger than $SLG1$'s. We assume that register file dissipates 36% and 52% less leakage power while in $SLG1$ and $SLG2$ modes, respectively [10].

TABLE III: Performance penalty associated with different relative values of x, y, z.

low power modes	x <y <z	x <z <y	z <x <y
$SLG1$	0	0	x-z
$SLG2$	0	y-z	y-z

In Figure 4 we report performance loss and power reduction for various $x.y.z$ models. We choose 3, 4, 5, 13 and 24 for x, 4, 5, 6, 7, 17 and 35 for y and 3, 5, 10 and 20 for z. Note that performance is compared to a GPU where all registers are always active. As it can be seen from the figure, performance loss is more sensitive to $SLG1$ rather than $SLG2$ wakeup

latency. This is due to the fact that an increase in $SLG1$ wakeup latency, delays all warps moving to active mode. An increase in $SLG2$ wakeup latency, however, only stalls the first warp executing after an S-stall is finished (see Table III).

Processors employing deeper pipelines, i.e. processors with larger z, are less prone to such latencies as an increase in the number of stages between fetch and register access stages provides enough time to wake up sleeping registers in time.

As Figure 4 (b) shows some benchmarks, e.g., $Gassian$, show more power reduction growth by increasing the number of pipeline stages (z) and wakeup latencies (x and y) and some benchmarks, e.g., $Scan$, save less power. Power reduction depends on the share of each power saving mode of the total execution time. The higher the ratio of time register file spends in SLG modes, the higher power reduction.

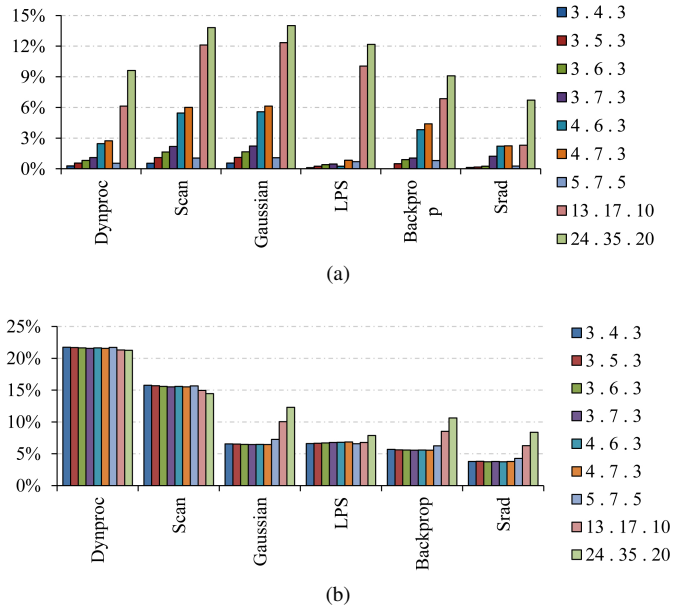

Fig. 4: (a) Performance loss and (b) power reduction for different timing scenarios. Performances are compared to GPUs where all registers are always active.

V. Related Works

Recently, power gating has been used as a primary power management technique in multicore processors. Kim et al. introduced the concept of intermediate power saving mode in [11]. They proposed and evaluated a power gating structure with two power saving modes. One is high leakage reduction without state retention and the other is intermediate leakage reduction with state retention.

Singh et al. proposed a circuit supporting multiple intermediate power saving modes [6]. Each mode represents a different trade-off between power leakage reduction and wakeup latency. The power reduction was determined by controlling the steady state V_{GND} in the sleep mode.

Roy et al. applied intermediate power saving technique with two sleep modes to the register file in multicore processors to reduce leakage power dissipation for memory pending instructions while saving the register values content. They evaluated their proposed technique for coarse-grained, fine-grained and simultaneous multithreading CMPs [10].

Gebhart et al. proposed two techniques to reduce energy in GPUs [12]. They proposed using a register file cache to replace accesses to the large main register file with accesses to a smaller structure which contains the immediate register working set of active threads. They also investigated a two-level thread scheduler which partitions threads into active and pending threads.

VI. Conclusion

In this paper we studied the stalls associated with synchronization instructions in GPUs. We introduced SLG which utilized intermediate power gating to reduce leakage power dissipation in the register file. The results showed that SLG reduced leakage power dissipation by up to a maximum of 22% while maintaining performance at a competitive level.

References

[1] N. Corporation. Tesla c2050 and tesla c2070 computing processor board. http://www.nvidia.com/docs/IO/43395/Tesla_C2050_Board_Specification.pdf.

[2] Y. Wang, S. Roy, and N. Ranganathan, "Run-time power-gating in caches of gpus for leakage energy savings," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 300 – 303.

[3] D. Li, S. Byna, and S. Chakradhar, "Energy-aware workload consolidation on gpu," in *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, sept. 2011, pp. 389 –398.

[4] N. Corporation. Nvidia cuda sdk code samples. http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html.

[5] J. Meng, D. Tarjan, and K. Skadron, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 235–246. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815992

[6] H. Singh, K. Agarwal, D. Sylvester, and K. Nowka, "Enhanced leakage reduction techniques using intermediate strength power gating," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 11, pp. 1215 –1224, nov. 2007.

[7] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, april 2009, pp. 163 –174.

[8] M. Giles. (2008, Apr) Jacobi iteration for a laplace discretisation on a 3d structured grid. http://people.maths.ox.ac.uk/œgilesm/hpc/NVIDIA/laplace3d.pdf.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, oct. 2009, pp. 44 –54.

[10] S. Roy, N. Ranganathan, and S. Katkoori, "State-retentive power gating of register files in multicore processors featuring multithreaded in-order cores," *Computers, IEEE Transactions on*, vol. 60, no. 11, pp. 1547 –1560, nov. 2011.

[11] S. Kim, S. Kosonocky, D. Knebel, and K. Stawiasz, "Experimental measurement of a novel power gating structure with intermediate power saving mode," in *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, aug. 2004, pp. 20 –25.

[12] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," in *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 235–246. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000093