JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 14, No. 3, September 2020, pp. 102-111

GPGPU Functional Units Power Gating for Leakage Energy Reduction

Xin Wang
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
wangx44@vcu.edu

Wei Zhang*
Department of Computer Science and Engineering, University of Louisville, Louisville, KY, USA
wei.zhang@louisville.edu

Abstract
The execution units of GPUs (graphics processing units) have been observed to produce many idle cycles that could be a tremendous waste of energy consumption which meanwhile provides a hint to build a more energy-efficient system to operate GPUs if idle cycles can be appropriately taken care of. However, power-gating without foresight can be dangerous since inaccurate decisions on power-gating will introduce unaffordable overhead on both energy consumption and performance. In this paper, we examine the length of execution units' idle cycles for several representative GPGPU applications and evaluate the distribution of the idleness durations. We then propose the energy-saving strategies with focus on discovering potential execution units' power-gating opportunities. The idle durations are recorded in the runtime for various computing units in streaming multiprocessors (SMs) including integer units and floating units in streaming processors (SPs) and special function units (SFUs). By analyzing the observed idleness, we propose to enhance the energy efficiency through two execution units' power-gating policies, the immediate power-gating (IPG) and idle detect power-gating (ID-PG). Furthermore, we examine the policies with various parameter settings to offer insights on possible gains and losses of the power-gating techniques. Besides, by noticing that integer units are the most popular computing units for many applications, we introduce the power-aware SP(s) to increase the throughput of integer instructions. It was observed that the power-aware SP can provide performance enhancement as well as the leakage energy reduction for several applications. The experimental results show that both the policies can result in satisfactory leakage energy saving on execution units. The IPG can reduce the execution unit's leakage energy by 84.3% when the break-even time is set to 5 cycles. Even if the break-even time goes up to 20 cycles, the ID-PG can save 67.1% of the total execution units' leakage energy. Moreover, involving power-aware SP(s) can improve the performance by up to 14.4% and 2.7% on average.

Category: Embedded Systems

Keywords: GPGPUs; Energy-efficiency; Execution units; Power-gating

Open Access **http://dx.doi.org/10.5626/JCSE.2020.14.3.102** **http://jcse.kiise.org**

Received 12 July 2020; **Revised** 26 July 2020; **Accepted** 07 August 2020
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

I. INTRODUCTION

Graphics processing units (GPUs) has become popular accelerators for the data-parallel general-purpose applications such as compute-intensive scientific computing programs [1–3]. The software layer such as NVIDIA CUDA [4] and AMD OpendCL [5] can utilize the GPU's hardware effectively to accelerate the applications tremendously. There are a wide range of existing parallelizable applications formerly running on CPUs that can be tailored to GPUs for significant performance benefit. The CUDA programming language allows the programmer to define the paralleled portion of an application as several kernels, each consisting of thousands of threads executing in parallel [4]. One GPGPU application usually contains multiple CUDA kernels packed with a group of thread blocks, and is called concurrent thread array (CTA). A vector, matrix or volume computation domain can be defined as one-dimensional, two-dimensional or three-dimensional thread blocks. The sub-level of the thread block is a warp which is the basic execution unit. A warp contains 32 threads with consecutive thread IDs and is executed in a single instruction multiple-threads (SIMT) style. The 32 threads in the same warp execute the same instruction and work on different data fragments. For each warp, only one PC is handled and thus a single instruction is fetched and decoded. Due to the fact that threads in the same warp are fed with different data for processing, they can access different memory addresses and follow different control flow paths. A GPGPU can overlap long latency through massive thread level parallelism and therefore allows significant performance improvement for the applications running on it.

Although the GPGPU is powerful in boosting the performance, the energy efficiency becomes an inevitable concern. In GPGPUs, thousands of threads or more run concurrently in a single instruction multiple data (SIMD) pattern. The massive concurrency is supported by a large number of computation units and a huge size of the register files, while both the hardware resources consume a considerable portion of the GPU's total energy. To host more concurrent threads, these hardware resources keep on increasing. Consequently, both dynamic and leakage energy of GPGPUs are on the rise and further performance improvement is compromised. The worst part is that the hardware resources are only demanded for peak performance requirement, although many applications cannot take full advantage of them. Subsequently, a great portion of components are left idle without any function leading to energy wastage [6–13].

In this paper, we study the idleness patterns of GPGPU execution units and attempt to uncover the inherent opportunities for power-gating in energy-efficiency enhancement without involving any re-schedule techniques. Researches in [9, 14] depend on re-scheduling instructions to reconstruct the instruction sequences and intentionally generate long idle durations. Unlike active utilization of the re-schedule techniques as reported in the previous studies, this work focuses on analyzing the existing idleness of the execution units with respect to the default instruction sequences. According to the inherent idleness, we explore appropriate power-gating strategies to save GPGPU leakage energy. The aim of this paper is to guide the future GPU energy studies regarding the execution units' power-gating. Instead of involving additional microarchitectures for complex rescheduling logic, only basic counters are required for the proposed method which is applicable with low hardware overhead. Besides the power-gating strategies, the idleness pattern also inspires us to increase the number of integer units in a streaming multiprocessor (SM).

In short, this paper makes the following contributions. First, the distribution and the length of execution units idle cycles are recorded and analyzed. To be specific, the idle durations of integer units, floating point units, and special function units (SFUs) have been recorded during run-time for analysis. According to the observed idleness pattern, the execution unit power-gating strategies have demonstrated ability to maximize leakage energy reduction. Second, we propose two simple execution units' power-gating policies—immediate power-gating (IPG) and idle detect power-gating (ID-PG)—and evaluate their effectiveness on the leakage energy saving, respectively. Both policies operate the execution units' power-gating at a fine granularity. Third, we further evaluate the two policies with various parameter settings. Based on the experimental results, we make two suggestions to maximize the leakage energy saving and avoid the side-effect of the power-gating: (1) when the power-gating overhead is unaffordable, an idle detect time is set up to avoid short-term idleness, and (2) when the power-gating overhead is acceptable, the execution units are put into power-gated mode immediately to save as much leakage energy as possible.

Finally, we propose to enhance the performance as well as the energy efficiency by increasing the number of integer units per SM.

The rest of this paper is organized as follows. Section II presents the background of this work. The related work is discussed in Section III. Section IV describes the motivation behind this work. Section V introduces the implementation details of our technique. The experimental results are presented in Section VI. Finally, Section VII concludes the paper.

II. BACKGROUND

In this section, we introduce the baseline GPU architecture and describe the CUDA programming model. We further explain how the hardware and software lead to underutilization of high hardware resources. It is believed

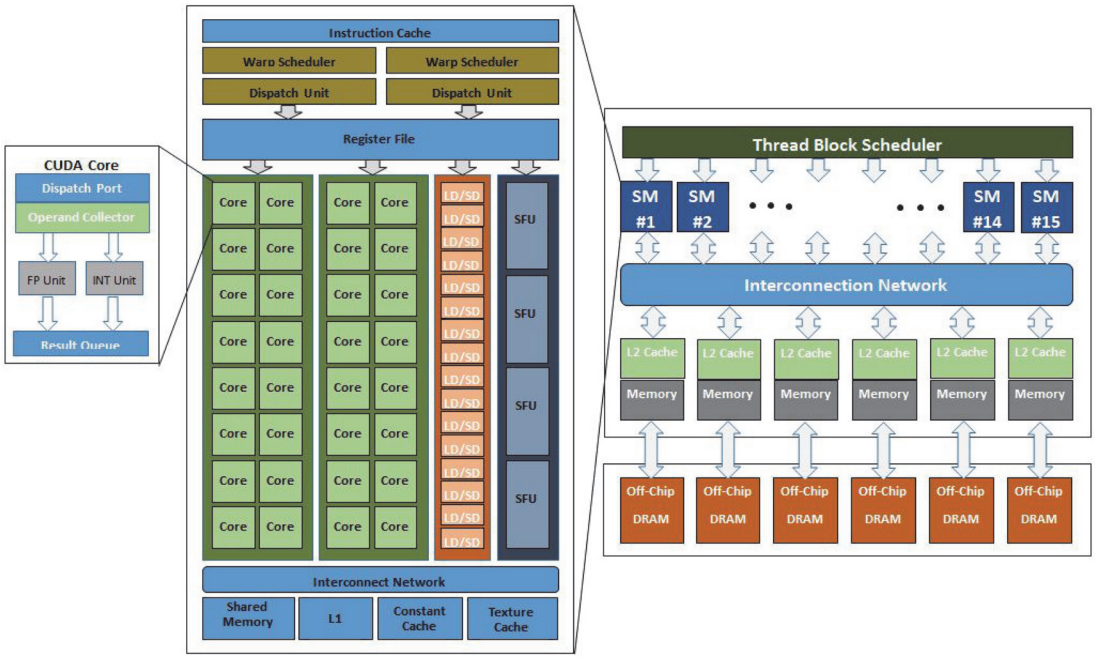


Fig. 1. Baseline GPU architecture.

that the power-gating techniques can utilize the under-utilization of the hardware resources to mitigate the growing energy consumption problem on GPUs.

A. Baseline GPU Architecture

We evaluate our work over the baseline GPU architecture that is shown in Fig. 1. The baseline architecture is similar with NVIDIA GTX480 GPUs [15] which consists of 15 GPU cores called streaming multiprocessors. Each SM has its own private L1 data caches, read-only texture caches, constant caches, and software-managed shared memories (i.e., scratch-pad memories). According to the configuration controlled by the software (Table 1), the L1 cache and the shared memory can be set to 16 kB L1 cache and 48 kB shared memory or 48 kB L1 cache and 16 kB shared memory. A single SM core consists of 32 single instruction multiple data execution units, 16 load/store units, and 4 SFUs. A unified L2 cache is shared by all the SMs via an on-chip network and is partitioned into 6 tiles. In an SM, there are two warp schedulers and two instruction dispatch units, which allow issuance of two independent instructions from two different warps at a single cycle. The two independent instructions can then be simultaneously executed by two streaming processors (SPs) each containing 16 execution lanes called the SIMT lanes. Since the execution units operate at double clock frequency of the SMs, it is hypothesized that 32 threads can co-run on a single SP. As shown in Fig. 1, the SP can address both integer and floating-point instructions owing to the fact that it is featured with an integer unit and a floating-point unit. The SFUs are typically responsible for complex calculations such as sin, cosine, reciprocal, and square root. However, they can also execute integer and floating-point instructions in case of the unavailability of both the SPs in the SM.

Table 1. Simulated GPU architecture configuration

CPU	15 SMs, 700 MHz
SM configuration	16 thread blocks/SM, 32 threads/warp 2 warp schedulers, 1024 ROB entries, 32 SIMD width, 5-stage pipeline
Register file	128 kB per SM 32 banks, dual-ported (1 read port and 1 write port) for each bank 4 kB register per bank, 256-bit wide entry, 128 entries per bank
L1 cache	16 kB, 4-way set-associative, line size 128 byte, 128 MSHR entries
L2 cache	768 kB, 8-way set-associative, line size 128 byte
Shared memory	48 kB, 32 banks, 1 cycle latency

B. CUDA Programming Model

To parallelize the application and tailor it for running on GPUs, the CUDA programming model allows the programmer to define C functions as several kernels. Each kernel consists of thousands of concurrent threads [4]. Each thread in a kernel can be identified with a unique thread ID which is accessible through the built-in **threadIdx** variable. Every unit of 32 threads with consecutive thread IDs is grouped as a warp. A warp is a basic unit of the concurrency and all the threads within it execute simultaneously in an SIMT way. Officially, the maximum number of concurrent threads in an SM is restricted by the NVIDIA. Specifically, for Fermi architecture, an SM can host 1,536 active threads (48 active warps) to the maximum. Moreover, the number of thread blocks is limited to 8 per SM. To support the execution environment and conserve execution data, each thread takes a piece of hardware resources such as register files and the shared memory. Consequently, the capacity of hardware resources of an SM also limits the number of concurrent threads per SM (i.e., occupancy). An SM can accommodate more than 1,536 lightweight threads which are supported with few hardware resources. However, except for touching the full capacity of an SM set by NVIDA, redundant hardware resources may well stay idle, resulting in a great waste of energy. On the other hand, if an SM is running with threads that heavily rely on hardware resources, low underutilization of the hardware resources could still be possible. For example, an SM can only serve one thread block that consumes two-thirds of the total register files. In this case, the rest one-third register files which are insufficient to support another thread block needs to stay idle. Besides, the way of GPUs handling branches also leads to significant underutilization of the hardware resources (i.e., execution units) and harms the energy efficiency. CPUs are capable of dealing with complex branch prediction logic, while GPUs avoid branch prediction and simplify the logic to accomplish overwhelming multithreading. To this end, an active mask vector is introduced to identify the divergence of execution paths. The active mask vector expresses whether the branch instruction is taken or not for the threads in a warp. The threads are then directed to different paths accordingly. To be specific, the threads will be executed and retired if the corresponding bits are set in the active mask vector, while other threads with reset bits will discard the execution. Threads with taken and not-taken paths are executed independently and sequentially and will be joined at the convergence point automatically. Due to the divergence, threads in a warp run alternately and the execution lanes cannot be always fully occupied. Consequently, the presence of idle execution lanes harms the energy efficiency.

III. RELATED WORK

GPGPUs are becoming promising platforms for accommodating parallelizable compute-intensive applications. Indeed, the support of a massive number of execution units and abundant bandwidth enable GPGPUs to provide an extremely high throughput by concurrently executing thousands of threads; however, the energy efficiency can become an urgent concern if the GPGPUs attempt to further improve the performance. Moreover, a huge amount of execution units is responsible for a considerable portion of the total energy consumption and they unfortunately produce many idle cycles resulting in energy wasting. Many researchers have studied GPGPU's energy-efficiency [6, 7, 9, 11, 14, 16–20]. Numerous researches discovered underutilization of various GPU components such as register files and execution units and proposed power gating the idle resources for energy saving. Several studies [6, 7, 16] propose to shut down unused fragments or put them into the low power modes to reduce leakage energy consumption of GPU register files. The authors of [17] implements the power-gating at the SM level granularity. They monitor the activity of entire SMs and shut down the unoccupied SMs to improve energy-efficiency. Some other solutions operate the power-gating at a much finer granularity [9, 14]. They explore the idleness of the execution units and design strategies to apply power-gating down to per SIMT lane and minimize the leakage energy wasting. By comparison, this paper studies the inherent idleness of the integer and floating-point units of SP as well as the SFU. To reduce GPU leakage energy dissipation, we have studied two different policies operating the power-gating adaptively and efficiently on different execution units.

IV. MOTIVATION

A. Execution Units Energy Consumption

The execution units in GPUs are responsible for 20.1% of the total energy consumption [21]. The energy consumption of execution units in NVIDIA GTX480 GPUs is broken down in [9] and the results show that leakage energy accounts for around 50% of the total energy consumed in integer execution units, and more than 90% in floating-point units. In this work, we focus on evaluating the execution units' leakage energy consumption. The experimental results show that the integer units consume only negligible leakage energy (around 0.1% of the total GPU's energy) and 9.5% of the total GPU's energy is contributed to the leakage energy consumed by floating-point units. Furthermore, the leakage energy of SFUs accounts for 2.2% of the total GPU's energy. Overall, the leakage energy consumption for all the executions units (integer units, floating-points units, and SFUs) is 11.6%

of the total GPU's energy consumption. Therefore, it is hypothesized that intellectual strategies will lead to positive impact on the overall GPU energy efficiency if they can effectively reduce the leakage energy spent on execution units, especially the floating-point units.

B. Execution Units Underutilization

We examine the underutilization of three execution units (integer units, floating-point units, and SFUs) in 10 GPGPU benchmarks. The experimental results show a great waste of execution resources. As shown in Fig. 2, the integer units stay idle during around 61% of the total execution time and the floating-point units are unused in 92% of execution cycles. SFUs are free in 75% of the execution period. The remarkable leakage energy consumption and the low utilization of execution units convince us to power-gate spared execution units intelligently and help GPUs to relieve the serious energy dissipation issue.

V. POWER-GATING STRATEGIES

Power-gating technique has been widely demonstrated to be effective for the leakage energy reduction. In this work, we use the traditional power-gating strategies to save execution units' leakage energy on GPGPUs. We operate the power-gating at a fine granularity and shut down the integer unit and floating-point unit per SIMT lane individually. The four SFUs in an SM can also be power-gated separately. t_{break_even}, t_{wakeup}, and t_{idle_detect} are the three parameters related to the execution units' power-gating technique. The t_{break_even} represents the number of cycles in the power-gated mode. The leakage energy saving during t_{break_even} is equal to the energy overhead of turning off and on the execution units. As long as the power-gated mode lasts longer than t_{break_even}, the leakage energy reduction is attained; otherwise, the energy overhead of the power-gating exceeds the insufficient saving and negatively affects the energy efficiency

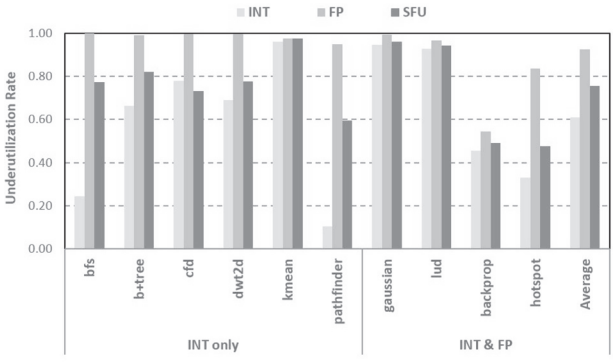


Fig. 2. The underutilization rate of execution units.

instead. The t_{wakeup} is the number of cycles spent on waking up the execution units. These extra cycles can lead to negative performance impact. Both t_{break_even} and t_{wakeup} can be calculated by using the formulas in [17] and vary with parameters of circuit components. The typical value for t_{break_even} lie between 9 to 19 cycles and t_{wakeup} is from 3 to 9 cycles [22]. In this paper, the evaluation starts with t_{break_even} of 10 cycles and t_{wakeup} of 3 cycles and continues with other t_{break_even} (5 and 20 cycles).

The power-gating decisions are made according to the threshold t_{idle_detect}. We can tune the t_{idle_detect} to achieve the balance between the leakage energy reduction and performance loss. A large t_{idle_detect} filters the short idleness of execution units to avoid performance degradation, but consequently misses opportunities of saving ever more leakage energy. We first evaluate with t_{idle_detect} of 0 cycles called immediate power-gating (IPG) in this paper. The IPG places the execution units into power-gated mode immediately as soon as they are not occupied. The IPG is capable of maximizing the leakage energy reduction if most of the idleness is longer than t_{break_even}; otherwise, the IPG could hurt both energy efficiency and performance if great extent of short-term idleness shows up. Due to the various idleness patterns, the optimal t_{idle_detect} can be varied for different execution units. In order to achieve further leakage energy saving, we examine the variation in energy reduction with different t_{idle_detect} (2 and 5 cycles), called as idle detect power-gating (ID-PG) in this paper. Based on how busy a certain type of execution unit is, the length distribution of the execution unit idleness is different and t_{idle_detect} varies accordingly.

Consequently, the power-gating strategies studied in this paper can be completely supported by several simple counters by avoiding complex microarchitecture and logic, and the hardware overhead is negligible. Moreover, the performance overhead that is mostly caused by the wake-up latency t_{wakeup} is negligible as well. An issued instruction is not going to be executed immediately. Instead, several cycles are taken into account to request operands of the instruction from the register file and collect them in the operand collector. The instruction is not forwarded to the execution unit until all the operands are ready in the operand collector. By noticing the time gap between the issue and execution stage, the SIMT lanes that will be used to execute the instruction can be woken up in advance right after the instruction has been issued. Getting operand ready can typically take up to 10 cycles, which is longer than t_{wakeup}. Therefore, t_{wakeup} successes to overlap itself with the period of fetching and collecting operands and thus is not expected to have a negative impact on performance. Moreover, since there are no warps and instructions rescheduling in our methods, the performance is not influenced by any rescheduling policy. Overall, both hardware and performance overheads of the power-gating strategies are negligible, which is also confirmed in our experiments.

Table 2. GPU benchmarks

Benchmark	Instruction type
bfs, b+tree, cfd, dwt2d, kmean, pathfinder	INT
gaussian, lud, backprop, hotspot	INT & FP

VI. METHODOLOGY & EXPERIMENTAL RESULTS

We use GPGPU-Sim v3.2.2 [23] to evaluate the execution units' power-gating schemes. GPGPU-Sim is a cycle-accurate GPU performance simulator that focuses on general-purpose computation on GPUs. GPUWattch [21] is used to measure the energy consumption and is integrated with GPGPU-Sim. As shown in Fig. 1, the baseline GPU architecture is modeled based on NVIDIA GTX480 GPUs [15]. There are 15 SMs and each SM consists of two SPs and four SFUs. An SP contains 16 CUDA cores and each core serves an SIMT lane. The CUDA core is able to execute both integer instructions and floating-point instructions. We evaluate the power-gating strategies with 10 benchmarks from Rodinia benchmarks suit [24]. Table 2 lists all the benchmarks. Four benchmarks include both integer and floating-point instructions (`gaussian`, `lud`, `backprop`, and `hotspot`). Other six benchmarks contain only integer instructions (`bfs`, `b+tree`, `cfd`, `dwt2d`, `kmean`, and `pathfinder`).

We have evaluated the leakage energy reduction of execution units' power-gating strategies proposed in this paper and compared them with the baseline `Ideal` (ideal leakage energy saving). We assume that the `Ideal` strategy recognizes the length of all idleness and can make perfect decisions based upon this knowledge. The `Ideal` only shuts down the execution units with the idleness longer than t_{break_even} and makes the power-gating action at the beginning of the idleness rather than waiting till t_{idle_detect} elapses. The leakage energy saving achieved by `Ideal` represents the best case that any power-gating strategies can ever attain.

A. Leakage Energy Reduction for Different Types of Execution Units

Fig. 3 shows that the leakage energy of integer units can be reduced up to 93% and 24.9% on average, when t_{break_even} = 10 and t_{idle_detect} = 0. For benchmarks such as `bfs`, `pathfinder`, `backprop`, and `hotspot`, the setting of t_{idle_detect} = 0 leads to more leakage energy dissipation from integer units, thereby hurting the energy efficiency. This is because numerous short-term idleness generates inevitable energy overheads.

The portion of the idleness over 10 cycles for integer units is shown in Fig. 4. Compared to other benchmarks, `bfs`, `pathfinder`, and `hotspot` show a smaller portion of idleness lying above 10 cycles (9.0%, 2.9%, and 11.5%, respectively). For `backprop`, its idleness mostly lies at the FPU, and not at the integer units or SFU. As a result, `bfs`, `pathfinder`, `backprop`, and `hotspot` fail to contribute towards energy efficiency due to the lack of long-term idleness. Moreover, the execution units have to be frequently switched on and off, thereby resulting in the overwhelming energy overhead and impacting the energy efficiency negatively, particularly for the integer units. This can be seen in Fig. 3, where the integer unit's leakage energy reduction is -28%, -22%, -44%, and -22% for `bfs`, `pathfinder`, `backprop`, and `hotspot`, respectively.

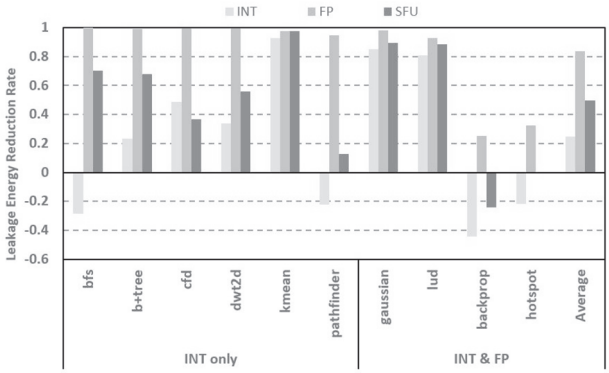


Fig. 3. Leakage energy reduction for different types of execution units (t_{break_even} = 10, t_{idle_detect} = 0).

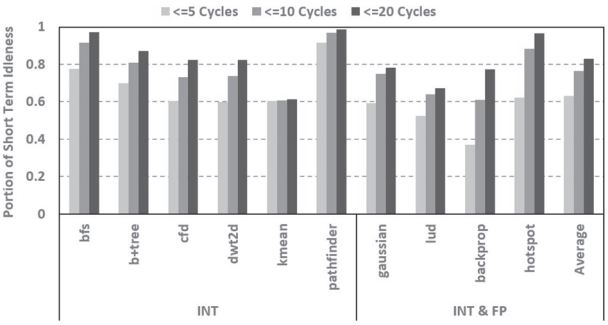


Fig. 4. The portion of the idleness below 5, 10, and 20 cycles for integer units (t_{break_even} = 10, t_{idle_detect} = 0).

As compared to the integer units, the power-gating works much better on reducing the leakage energy of floating-point units and SFUs (83.9% and 49.5% leakage energy saving for floating-point units and SFUs are reduced, respectively). Since 93% of the idleness of SFUs is shorter than 10 cycles for `backprop`, the very high occurrence of short-term idle periods causes the increase of SFU leakage energy consumption for `backprop`.

B. Leakage Energy Reduction for Different t_{idle_detect}

We next evaluate the power-gating strategies with

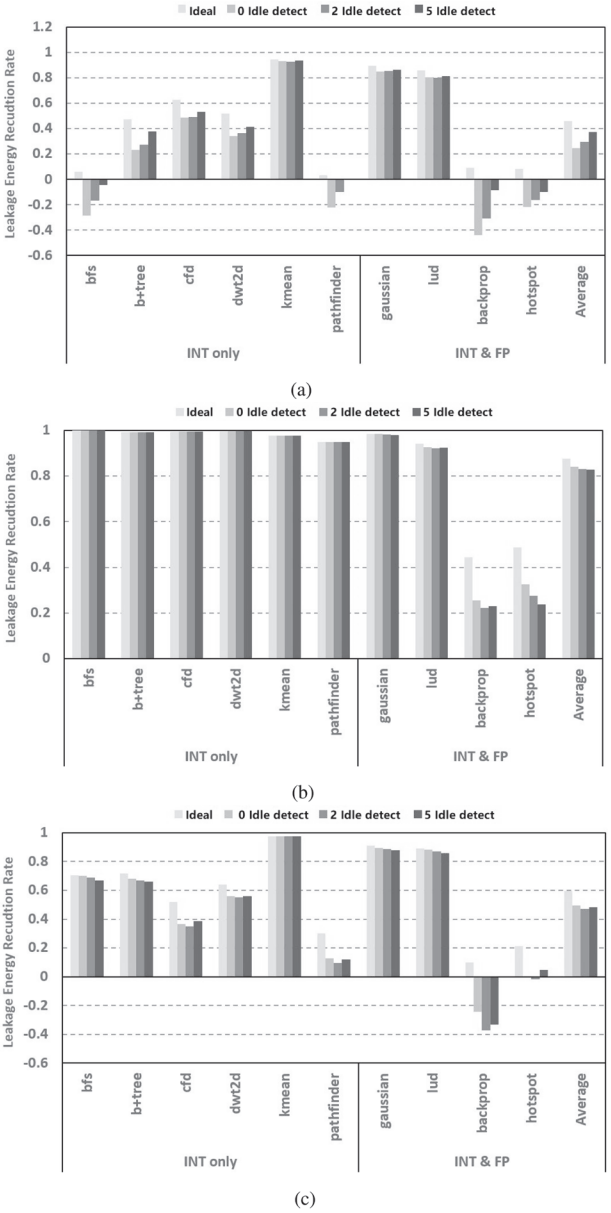


Fig. 5. Execution units leakage energy reduction rate for different idle detect time (t_{idle_detect} = 0, 2: and 5). (a) integer unit, (b) floating-point unit, and (c) SFUs.

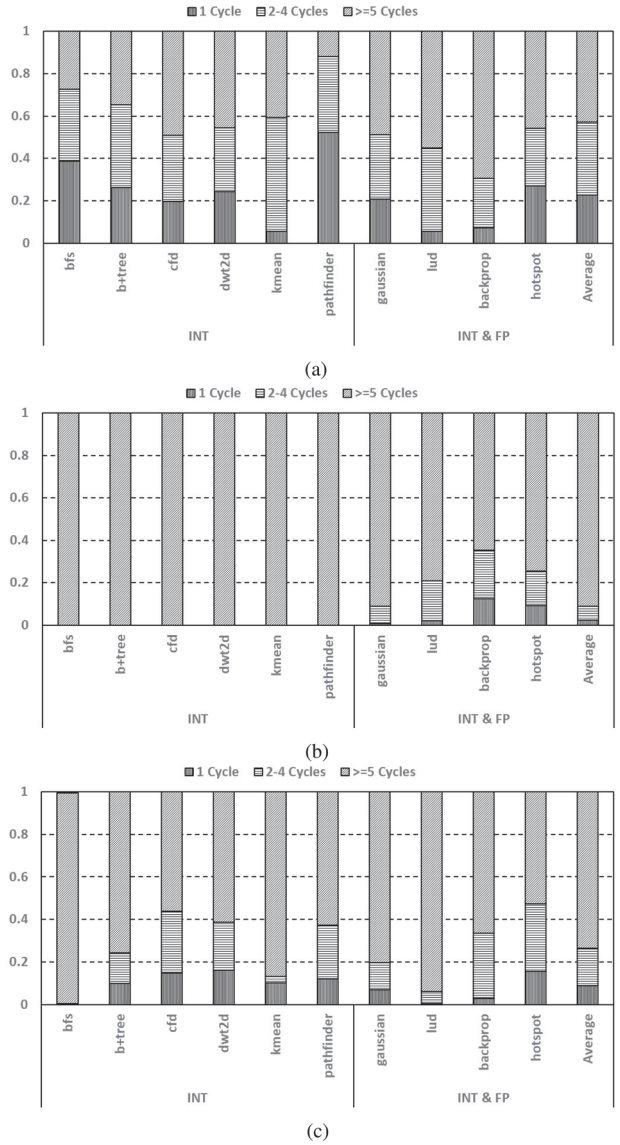


Fig. 6. Idleness breakdown of (a) integer unit, (b) floating-point unit, and (c) SFUs.

different idle detect time (t_{idle_detect} = 0, 2, and 5 cycles). Fig. 5(a) depicts that increasing t_{idle_detect} can improve the integer unit's leakage energy saving, as more and more short-term idle intervals are filtered by longer t_{idle_detect} and the energy overhead is compromised. Increase in t_{idle_detect} from 0 to 5 cycles, the average leakage energy reduction boosts from 24.9% to 37.2%. The breakdown of idleness for integer units is shown in Fig. 6(a). When t_{idle_detect} increases from 0 to 2 cycles, the 2-cycle threshold can avoid power-gating for 1-cycle idleness. The overhead can be remarkably removed owing to the fact that 23% of total idle durations on average is only 1 cycle. With a further increase in t_{idle_detect} to 5 cycles, even more short-term idleness with intervals of 2, 3 or 4 cycles can be filtered. As shown in Fig. 6(a), 34% of total idleness on average is 2, 3 or 4 cycles and therefore the energy overhead can be further reduced.

Fig. 5(b) and 5(c) show that operating the power-gating with shorter t_{idle_detect} can lead to better energy-efficiency for floating-point units and SFUs. Fig. 6(b) and 6(c) explain that only limited number of short idle intervals can be gated by t_{idle_detect} when it becomes longer. Specifically, setting t_{idle_detect} to 5 cycles can only remove the power-gating overhead from 2.5% (1 cycle) and 6.5% (2–4 cycles) of the total idleness for floating-point units. And, for SFUs, 26% more idleness can be found in total (9% of 1 cycle idleness and 17% of 2–4 cycles idleness).

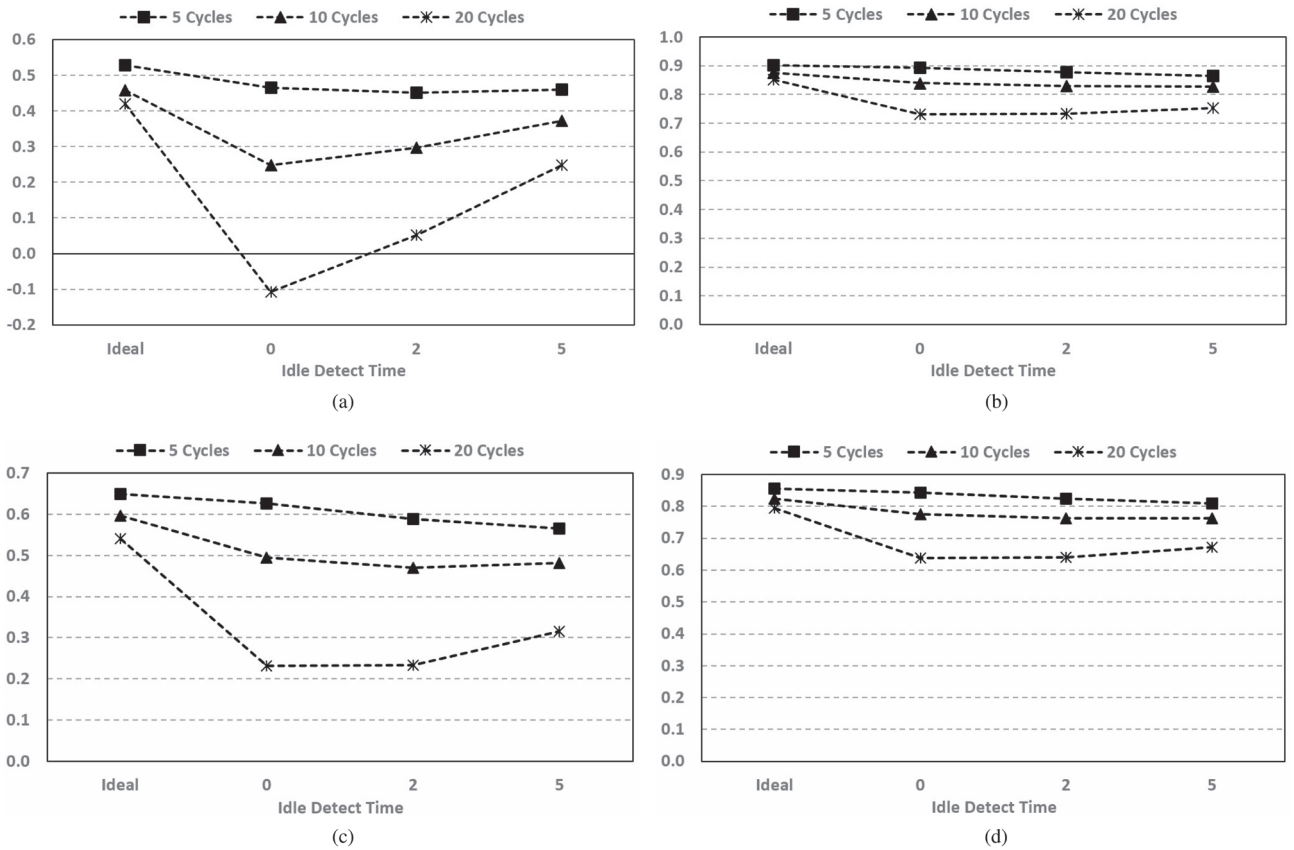


Fig. 7. Leakage energy reduction with different t_{break_even}: (a) integer unit, (b) floating-point unit, (c) SFUs, and (d) overall leakage energy reduction of all execution units.

Once the longer t_{idle_detect} is unable to dig more enough short-term idleness to optimize the power-gating strategies, it leads to wastage of additional cycles on waiting the power-gating decision rather than power-gates immediately to enhance the energy efficiency.

C. Sensitivity Analysis of t_{break_even}

As shown in Fig. 7, the longer idle detect time becomes the optimal choice for all the execution units, when t_{break_even} goes up from 5 cycles to 20 cycles. Especially, the power-gating technique leads to maximum reduction in the leakage energy of floating-point units when 5 cycles idle detect time is chosen instead of 0 cycle. With an increase in t_{break_even}, the energy budget of power-gating execution units increases. In this case, the energy overhead reduction by avoiding short-term idle periods exceeds the energy cost of long-term idle periods spent on additional cycles to reach the idle detect time.

As aforementioned in Section IV, only 0.1% and 2.2% of the total GPU energy is consumed by the integer units and SFUs. On the other hand, the leakage energy of floating-point units contribute to 9.5% of the total GPU energy. Obviously, the power-gating strategies working on floating-point units offer the most effective way for productive leakage energy saving. Fortunately, based on the results shown in Fig. 7(b), the evaluated power-gating schemes hold the potential to approach a satisfactory leakage energy saving of floating-point units. The average leakage energy reduction is very close to that of the ideal case. However, increase in the break-even time weakens the power of the power-gating and pulls the leakage energy saving against the best case. Not surprisingly, the overall leakage energy saving of all the execution units follow a trend similar with the floating-point leakage energy reduction as represented in Fig. 7(d). As compared to the ideal case (85.5%, 82.4%, and 79.3%), 84.3%, 77.5%, and 63.1% reduction of execution unit's leakage energy can be reached for the break-even time of 5, 10, and 20 cycles, respectively.

D. Add Simple SP(s) for Enhancement of Both Performance and Energy Efficiency

According to the experimental results presented in Fig. 2, the integer units are the overall demanding resources. Therefore, increasing the number of integer units in an SM can relieve the pressure of integer computation and potentially improve the performance. Moreover, by adding simple SPs which are dedicated to integer computations,

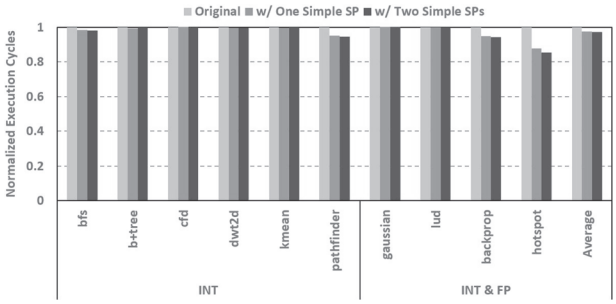


Fig. 8. Normalized execution cycles with simple SP(s).

more normal SPs can yield floating-point computation. For the applications including both integer and floating-point computation, the waiting line of floating-point units might be cut down and the performance could be boosted. The energy overhead of introducing more integer units can be negligible as only 0.1% of the total GPU energy is consumed by the integer units. In this paper, we evaluate the performance when one or two simple SPs are added for every two normal SPs in an SM. Fig. 8 shows the results of normalized execution cycles with simple SP(s). For the applications working on integer computation only, there is only slight performance improvement. However, `bfs` and `pathfinder` enjoy a little benefit on the performance (1.8% and 5.2% improvement, respectively) due to their extreme demand of integer units. As shown in Fig. 2, integer units are unoccupied during only 24.5% and 10.6% of the total execution time for `bfs` and `pathfinder`, respectively. For applications including both integer and floating-point computation, extra integer units can be more helpful in boosting performance as the simple SPs can free the normal SPs for floating-point computation and the floating-point instructions that are stalled for lack of normal SPs can acquire the resources earlier. For example, extra integer units eventually achieve 5.6% and 14.4% performance improvement for `backprop` and `hotspot` that rely on both integer and floating-point units.

VII. CONCLUSION

The execution units' power-gating strategies evaluated in this paper are demonstrated to achieve considerable saving on leakage energy dissipation and improve the GPU energy-efficiency. No additional microarchitecture with complex control logic is required and both hardware and performance overheads are negligible due to the simplicity. By evaluating the different parameter settings of the power-gating strategies, we find that an idle detect time should be involved to identify and filter short-term idleness. On the other hand, the execution units can be put in the power-gated mode as long as they are unoccupied to maximize the leakage energy saving, if the overhead is affordable. For break-even time down to 5 cycles, the IPG can lead to 84.3% leakage energy reduction in execution units, which is almost the same with the ideal case (85.5%). For break-even time up to 20 cycles, the ID-PG can reduce the leakage energy of execution units by 63.1%.

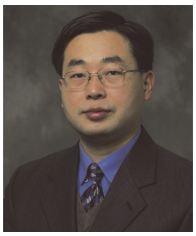
REFERENCES

1. K. Fatahalian and M. Houston, "A closer look at GPUs," *Communications of the ACM*, vol. 51, no. 10, pp. 50-57, 2008.
2. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370-1380, 2008.
3. P. Karnick, "GPGPU: general purpose computing on graphics hardware," 2006; http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=FE53DC39B11BC62568A5EB6C45B8AA7A?doi=10.1.1.184.5653&rep=rep1&type=pdf.
4. NVIDIA, "NVIDIA CUDA compute united device architecture: programming guide," 2008; http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf.
5. A. Munshi, "The openCL specification," in *Proceedings of 2009 IEEE Hot Chips 21 Symposium (HCS)*, 2009, Stanford, CA, pp. 1-314.
6. X. Wang and W. Zhang, "Drowsy register files for reducing GPU leakage energy," in *Proceedings* of *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, Shenzhen, China, 2017, pp. 632-639.
7. X. Wang and W. Zhang, "OWAR: operand-width-aware register packing for energy-efficient GPGPUs," Virginia Commonwealth University, Richmond, VA, 2017.
8. M. Rhu and M. Erez, "Maximizing SIMD resource utilization in GPGPUs with SIMD lane permutation," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, Tel-Aviv, Israel, 2013, pp. 356-367.
9. M. Abdel-Majeed and M. Annavaram, "Warped register file: a power efficient register file for GPGPUs," in *Proceedings of 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Shenzhen, China, 2013, pp. 412-423.
10. 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*, Saint-Malo, France, 2010, pp. 235-246.
11. 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 38th Annual International Symposium on Computer Architecture (ISCA)*, San Hose, CA, 2011, pp. 235-246.
12. W. W. Fun and T. M. Aamodt, "Thread block compaction for efficient SIMT control flow," in *Proceedings of 17th*

IEEE International Symposium on High Performance Computer Architecture, San Antonio, TX, 2011, pp. 25-36.
13. W. W. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic warp formation and scheduling for efficient GPU control flow," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Chicago, IL, 2007, pp. 407-420.
14. Q. Xu and M. Annavaram, "PATS: pattern aware scheduling and power gating for GPGPUs," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, Edmonton, Canada, 2014, pp. 225-236.
15. NVIDIA "NVIDIA's next generation CUDA compute architecture: Fermi," 2009; https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
16. M. Abdel-Majeed, D. Wong, and M. Annavaram, "Warped gates: gating aware scheduling and power gating for GPGPUs," in *Proceedings of 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Davis, CA, 2013, pp. 111-122.
17. P. H. Wang, C. L. Yang, Y. M. Chen, and Y. J. Cheng, "Power gating strategies on GPUs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 3, pp. 1-25, 2011.
18. S. Z. Gilani, N. S. Kim, and M. J. Schulte, "Power-efficient computing for compute-intensive GPGPU applications," in *Proceedings of IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Shenzhen, China, 2013, pp. 330-341.
19. M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A locality-aware memory hierarchy for energy-efficient GPU architectures," in *Proceedings of 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Davis, CA, 2013, pp. 86-98.
20. Y. Wang, S. Roy, and N. Ranganathan, "Run-time power-gating in caches of GPUs for leakage energy savings," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012, pp. 300-303.
21. J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: enabling energy optimizations in GPGPUs," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 487-498, 2013.
22. Z. Hu, A. Buyuktosunoglu,V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Newport Beach, CA, 2004, pp. 32-37.
23. A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, 2009, pp. 163-174.
24. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: a benchmark suite for heterogeneous computing," in *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, 2009, pp. 44-54.

Xin Wang

Xin Wang received his B.S. degree in electronic engineering from Peking University, China, in 2008. He is currently pursuing his Ph.D. degree in computer engineering at Virginia Commonwealth University, USA. His research interests include GPU and heterogeneous CPU-GPU architectures.



Wei Zhang https://orcid.org/0000-0003-1343-2817

Wei Zhang is a professor and Chair of the Department of Computer Engineering and Computer Science at the University of Louisville. He received his Ph.D. in Computer Science and Engineering from the Pennsylvania State University in 2003. Dr. Zhang served as an assistant/associate professor in Electrical and Computer Engineering at Southern Illinois University Carbondale (SIUC) from 2003 to 2010 and as an associate and full professor at Virginia Commonwealth University from 2010 to 2019. His research interests are in computer architecture, compiler, real-time computing, and hardware security. Dr. Zhang has led 8 NSF projects as the PI and has published 160+ papers in refereed journals and conference proceedings. He received the 2016 Engineer of the Year Award from the Richmond Joint Engineer Council, the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and the 2007 IBM Real-time Innovation Award.