ITAP: Idle-Time-Aware Power Management for GPU Execution Units

MOHAMMAD SADROSADATI, Sharif University of Technology
SEYED BORNA EHSANI, Sharif University of Technology
HAJAR FALAHATI, IPM
RACHATA AUSAVARUNGNIRUN, Carnegie Mellon University, KMUTNB
ARASH TAVAKKOL, ETH Zürich
MOJTABA ABAEE, IPM
LOIS OROSA, University of Campinas, ETH Zürich
YAOHUA WANG, ETH Zürich, National University of Defense Technology
HAMID SARBAZI-AZAD, Sharif University of Technology, IPM
ONUR MUTLU, ETH Zürich, Carnegie Mellon University

Graphics Processing Units (GPUs) are widely used as the accelerator of choice for applications with massively data-parallel tasks. However, recent studies show that GPUs suffer heavily from resource under-utilization, which, combined with their large static power consumption, imposes a significant power overhead. One of the most power-hungry components of a GPU, the execution units, frequently experience idleness when (1) an under-utilized warp is issued to the execution units, leading to partial lane idleness, and (2) there is no active warp to be issued for the execution due to warp stalls (e.g., waiting for memory access and synchronization). While large in total, the idle time of execution units actually comes from short but frequent stalls, leaving little potential for common power saving techniques, such as power-gating.

In this paper, we propose a novel technique, called *Idle-Time-Aware Power Management* (*ITAP*), which aims to effectively reduce the static energy consumption of GPU execution units. By taking advantage of different power management techniques (i.e., power-gating and different levels of voltage scaling), *ITAP* employs three static power reduction modes with different overheads and capabilities of static power reduction. *ITAP* estimates the idle period length of execution units using prediction and peek-ahead techniques in a synergistic way and then, applies the most appropriate static power reduction mode based on the estimated idle period length. We design *ITAP* to be *power-aggressive* or *performance-aggressive*, not both at the same time. Our experimental results on several workloads show that the power-aggressive design of *ITAP* outperforms the state-of-the-art solution by an average of 27.6% in terms of static energy savings, with less than 2.1% performance overhead. On the other hand, the performance-aggressive design of *ITAP* improves the static energy savings by an average of 16.9%, while keeping the GPU performance almost unaffected (i.e., up to 0.4% performance overhead), compared to the state-of-the-art static energy savings mechanism.

CCS Concepts: • **Hardware** → **Power and energy**; • **Computer systems organization** → *Single instruction, multiple data*;

Additional Key Words and Phrases: GPUs, Execution Units, Static Power, Power-Gating, Voltage-Scaling

Mohammad Sadrosadati performed a part of this work at ETH Zürich.
Lois Orosa was supported by FAPESP fellowship 2016/18929-4.

XXXX-XXXX/2018/12-ART1
https://doi.org/10.1145/3291606

ACM Reference format:
Mohammad Sadrosadati, Seyed Borna Ehsani, Hajar Falahati, Rachata Ausavarungnirun, Arash Tavakkol, Mojtaba Abaee, Lois Orosa, Yaohua Wang, Hamid Sarbazi-Azad, and Onur Mutlu. 2018. ITAP: Idle-Time-Aware Power Management for GPU Execution Units. *ACM Transactions on Architecture and Code Optimization* 16, 4, Article 1 (December 2018), 25 pages.
https://doi.org/10.1145/3291606

1 INTRODUCTION

Graphics Processing Units provide a very large number of processing resources capable of running thousands of concurrent threads. The Single-Instruction-Multiple-Thread (SIMT) execution model employed by the GPUs allows simpler control logic and enables the concurrent execution of thousands of threads performing the same instruction over different data elements.

Due to their significant parallelism capabilities, ease of programming, and high *performance-per-watt*, GPUs have become a viable option for executing general-purpose applications. However, as previous work also shows, general-purpose applications often lack the proper parallelism that GPUs were designed for, leading to a large amount of resource under-utilization [11, 39, 40, 100, 101]. Resource under-utilization in GPUs negatively affects power efficiency, which is an increasingly serious concern [1, 2, 28–30, 36, 60, 105].

One of the main GPU resources that is frequently under-utilized is the execution units. Idle execution units consume significant static power, which increases the total GPU power consumption [2, 3, 46, 105]. Reducing the static power of the GPU execution units is a major challenge for two reasons. First, a considerable part of the GPU die area is dedicated to the execution units, making them one of the main power-consuming resources in a GPU. As an example, previous work finds that the execution units of NVIDIA GTX 480 GPUs [74, 78] are the most power-consuming components of the architecture, contributing to 20% of the total GPU power [60]. Second, about 50% of the power consumption in the execution units is due to the static power [2, 3, 60, 105].

The execution units in a GPU consists of several Single-Instruction-Multiple-Data (SIMD) lanes, each of which executing a single thread instruction (i.e., single-instruction-single-data execution). When GPUs execute code, it is possible that some or all lanes of the execution units become idle, referred to as partial or full-lane idleness. Partial-lane idleness happens when the threads inside a *warp*, a fixed size group of threads executed in lock step, follow different execution paths due to executing a conditional branch instruction, known as *Branch Divergence*. Branch divergence leads to the serial execution of the two control flow paths after a branch. The threads contributing to the path are only active while the threads of the other path remain idle, resulting in *partial-lane idleness*. *Full-lane idleness*, on the other hand, occurs when there are no active warps to be scheduled for execution. Warp deactivation can occur as a result of long memory access latency, inter-warp synchronization, and resource contention [2, 11, 60].

To alleviate the static power overheads of partial- and full-lane idleness, previous proposals employ *power-gating*, which cuts off power to the idle execution units [1–3, 105]. Power-gating techniques intrinsically impose power and performance overheads, making them beneficial only when the idle periods are large enough (larger than the cost/benefit break-even point of power-gating; see Section 2.2). Previous studies [2, 3, 105] show that the idle time of GPU execution units is fragmented into short but frequent periods, seriously limiting the potential of power-gating. Blindly applying power-gating introduces more overhead than improvement, and defeats the purpose of power efficiency [2, 105]. Accordingly, previous proposals attempt to improve the opportunity of power-gating by defragmenting idle periods of the execution units [2, 3, 105]. For example, pattern-aware scheduling [105] proposes a warp scheduler to enlarge the length of the idle periods, which result from partial-lane idleness, in order to increase the opportunity of power-gating.

In this work, we show that there are two major limitations of techniques that use power-gating to reduce the static power of the idle execution units. First, the idle time of the execution units remains smaller than the power-gating break-even point, on average, for 85% of cases, even after using a state-of-the-art idle-time defragmentation technique [105]. Therefore, there is little or no chance to power-gate most of the idle periods. Second, for idle periods that are not significantly larger than the power-gating break-even point, voltage-scaling [1, 24, 88] can achieve a higher power efficiency than power gating.

To efficiently address the limitations of the previous proposals, we propose *ITAP*, a novel approach to reduce the static power of GPU execution units. *ITAP* predicts the length of the idle period and automatically applies appropriate power reduction mechanisms, such as power-gating and multiple levels of voltage-scaling. To achieve this goal, we first conduct a thorough analysis with respect to design space, overheads, and gains of different idle power reduction mechanisms in GPUs. Based on the analysis, we design *ITAP* to judiciously use three power management modes that are designed to cover all variations in idle period length: 1) voltage-scaling to 0.5 V for *short* idle periods, 2) voltage scaling to 0.3 V for *medium* idle periods, and 3) power-gating for *large* idle periods. Compared to idle-time defragmentation schemes, *ITAP* increases the opportunity of power reduction from an average of 4% up to 100%. To precisely estimate the length of the idle period, we use a *peek-ahead window* that predicts the exact time to enable each power saving mode. *ITAP* combines the idle length prediction and a peek-ahead window to effectively reduce the static energy of the execution units in GPUs with negligible performance overhead.

Through our extensive simulation experiments, we show that the *power-aggressive* and *performance-aggressive* variants of *ITAP* reduce the static energy consumption by 37.9% and 28.6%, on average, respectively, while the conventional and state-of-the-art power-gating techniques reduce the static energy by 2.5% and 14.2%, respectively. We show that *ITAP* incurs a small performance overhead of 1.2% on average in the *power-aggressive* mode, and a negligible average 0.2% overhead in the *performance-aggressive* mode. We also show that *ITAP* can be combined with previously proposed idleness defragmentation techniques, such as pattern-aware scheduling [105], which further improves the static energy savings of *ITAP* by 9.1%, on average.

We make the following contributions:

- We explore the design space of various idle power management modes to determine the most suitable ones based on the length of idle period in GPU execution units. As a consequence, we judiciously employ three static power reduction modes for GPU execution units to cover all idle periods with the lowest performance overhead.
- We devise a prediction scheme in combination with a peek-ahead approach to provide a highly accurate estimation of each idle period's length.
- We propose *ITAP*, a novel *idle-time-aware* power management technique for GPU execution units. The key idea is to leverage the estimated length of each idle period to apply the most effective power management mode.
- We show that *ITAP* significantly reduces static energy compared to the state-of-the-art approach, while incurring a negligible performance overhead.

2 BACKGROUND

The focus of this work is to provide an effective power management technique for GPU execution units. Therefore, we provide a description of basic GPU micro-architecture design (Section 2.1), and briefly explain common power reduction techniques (Section 2.2).

2.1 GPU Architecture

GPU kernels are composed of many Single-Program Multiple-Data (SPMD) threads grouped by the programmer into several Thread Blocks or Cooperative Thread Arrays (CTAs). Each CTA is assigned to a Streaming Multiprocessor (SM) upon thread launch. SMs are SIMD processing units with dedicated fast memory units. During execution, threads assigned to each SM are divided into multiple fixed length (e.g., 32) groups. Each group of threads is called a warp (NVIDIA terminology) or wavefront (AMD terminology).

Threads inside a warp are executed in parallel lock-step manner, where each thread executes the same instruction. SIMD units are time-multiplexed between different warps. In each cycle, the GPU selects one warp to be executed based on the GPU's warp scheduling policy following the SIMT model. In the SIMT model, all threads of a warp execute the same instruction on different data, but threads of a warp may take different control flow paths, leading to idle SIMD lanes (called *branch divergence*).

SMs are responsible for the execution of warps. Each SM is composed of several components, including SIMD integer/floating-point units, load/store units, special function units, L1 data and instruction caches, local shared memory, and a register file that is responsible for maintaining the context of all threads inside the SM. Figure 1 shows the GPU architecture evaluated in our study, modeled after the NVIDIA Pascal GPU architecture [76]. SMs are connected to memory nodes (MNs) with an on-chip network. Each MN consists of a memory controller, which is connected to the main memory (DRAM), and two Last Level Cache (LLC) banks.

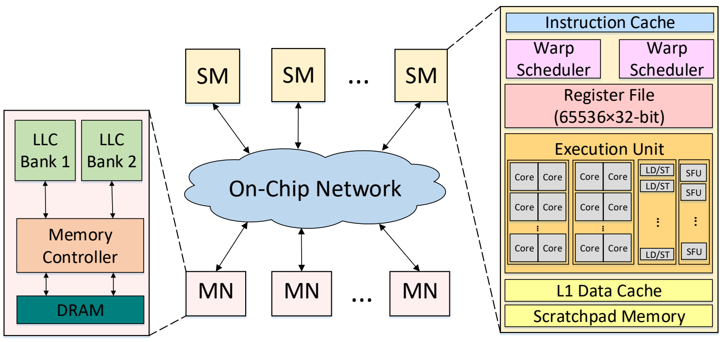

Fig. 1. Baseline GPU architecture.

2.2 Static Power Reduction Techniques

In the deep-sub-micron era, the contribution of the static power to total power consumption is significantly increasing [49, 87]. Therefore, several techniques for reducing static power consumption have been proposed over the past few years [1, 2, 7, 28, 33, 37, 41, 42, 46, 58, 60, 61, 73, 80, 91, 93, 95, 105, 107]. In this section, we briefly explain *Power-Gating* and *Voltage-Scaling*, two widely-used techniques shown to be effective at reducing static power. *Power-gating* (PG) [34, 102–104] cuts off the supply voltage entirely by use of a *sleep* transistor between the voltage supply line and the pull-up network, or the ground and the pull-down network. *Voltage-scaling* (VS) [2, 24, 55] uses voltage regulators to dynamically scale down the incoming supply voltage in order to reduce the energy consumption. In this work, we refer to both PG and VS as *sleep* modes. A unit that is in *sleep* mode does not function properly and needs to have its power supply switched back to full in order to regain functionality. We refer to this process as the *wake-up* process. This *wake-up* process demands both time (T_{wake_up}) and energy (E_{wake_up}), which, if not handled timely at the right voltage, imposes a significant overhead on performance and power [2, 34, 88, 103]. Peek-ahead techniques have been proposed in the past to alleviate the performance overhead of sleep modes, and issue the wake-up command of a unit in advance [2, 4]. Reducing the energy overhead depends on the ability to keep the unit in sleep mode for a long enough time (T_{break_even}), so that the energy savings from the sleep mode can break even with the energy overhead of the wake-up process [2, 4, 34].

The PG technique is usually used for memory-less units as PG destroys the value stored in memory cells. However, PG can be applied to memory units in some instances, such as virtual channels in network-on-chip, caches, and register file, when the units do not store important values or when the values can be recovered from other units [1, 4, 19, 20, 25, 34, 54, 66, 67, 79, 92, 102–104, 106]. In contrast, the VS technique can be easily used for blocks with memory cells, as it does not entirely cut off the supply voltage, and consequently the values on memory cells are not lost. To reduce the static power consumption of the execution units in GPUs, both PG and VS techniques are applicable. Based on our experiments and the literature [1, 2, 34, 102, 103, 105], T_{break_even} and T_{wake_up} in PG are significantly larger than that of VS, which makes PG more suitable for long idle periods. VS, on the other hand, is preferred for short idle periods as it has smaller T_{break_even} and T_{wake_up}, allowing additional energy saving with negligible performance overhead.

3 MOTIVATION

Previous studies [60, 89, 98, 105] demonstrate that power consumption in GPUs cannot be traced back to a single dominant component. GPUs contain many power-hungry elements, including execution units, register file, caches, on-chip network, and off-chip memory, all of which can be targeted separately to increase power efficiency [1–3, 12, 44, 50, 51, 56, 60, 68, 69, 81, 82, 88–90, 98, 99, 105] The focus of this work is to 1) study the power efficiency challenges of one of those components, execution units, one of the most power-hungry units inside the GPU [2, 3, 60, 105], and 2) propose an idle-time-aware solution to alleviate their power consumption. In this section, we first examine various sources behind execution unit idleness in Section 3.1. Then, we discuss the inefficiency of previous proposals in reducing the static energy of execution units in Section 3.2. Based on our observations, we then lay out our goals in Section 3.3.

3.1 Execution Unit Idleness

We first provide a detailed analysis of GPU execution unit utilization and idleness patterns. Figure 2 shows the fraction of run-time during which SIMD lanes are inactive. This figure shows that, on average, SIMD lanes are idle for over 53% of the entire run-time. In the subsections to come, we elaborate upon the two prevailing sources of such under-utilization of GPU resources: *partial-lane idleness* and *full-lane idleness*.

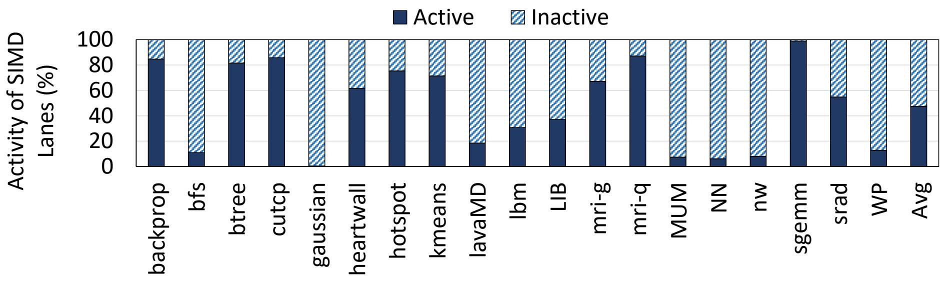

Fig. 2. SIMD execution lane activity during runtime.

While methods such as reducing the SIMD lane size have been introduced to mitigate the SIMD lane idleness [78, 97], reducing the number of SIMD lanes can lead to a significant performance loss. Figure 3 shows that reducing the SIMD lane size causes a significant performance loss across a wide range of applications. We conclude that reducing SIMD lane size cannot be used as a technique to alleviate execution unit power consumption.

We find that there are two major sources of execution units idleness in GPUs:

3.1.1 Partial-Lane Idleness. This occurs as a result of branch divergence [1, 26, 27, 71, 85, 105]. Upon executing a conditional branch, the threads inside a warp might have to follow different

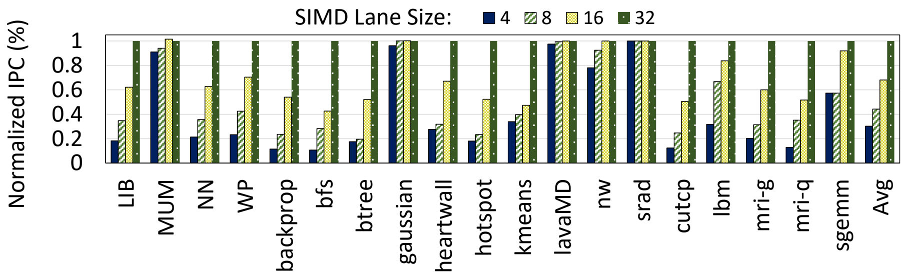

Fig. 3. Effect of SIMD lane size on performance.

control-flow paths, based on their unique operands. Consequently, the executing warp is executed using two separate warps, one for each path (taken and not taken). Each of these two warps have *partially* active lanes, and the warps have to be executed one after another. Upon completion of the execution of both paths, the warps rejoin to continue normal execution as a single warp [52, 53].

The activity of the execution units inside SIMD lanes is managed by keeping an *active-mask* for every warp inside the SM. The *active-mask* is a binary array with the same size as the number of threads in a warp. During the execution of a warp, this *active-mask* stores a value of 1 for every active lane and a value of 0 for every inactive lane. For example, the active mask of 11001010 for a warp size of eight shows that threads at lanes #7, #6, #3, and #1 are active, leading to 50% SIMD utilization. By logging the active masks of executing warps during run-time, we can quantify the idleness of execution units, and examine the role of branch divergence on idleness. Warps can execute with the least number of SIMD active threads (1), or the most (8 in our example; 32 in our evaluations; i.e., when all execution lanes are active).

Figure 4 shows the distribution of the fraction of active threads for multiple GPGPU applications (See Section 5 for the description of our evaluated benchmarks). A warp consists of 32 threads.

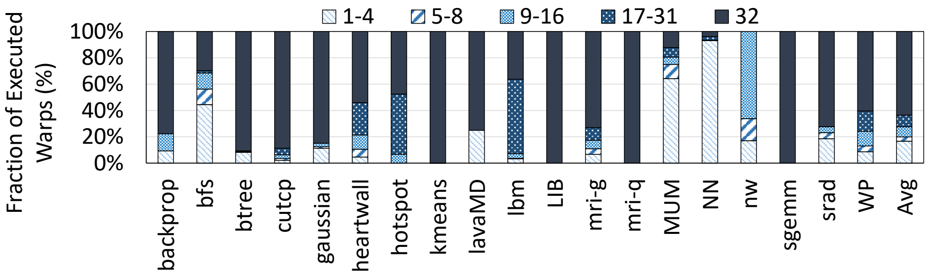

Fig. 4. Distribution of the number of active threads across all executed warps.

We observe that only four GPGPU applications: `LIB`, `mri-q`, `sgemm`, and `kmeans` achieve 100% SIMD utilization for nearly all warps. In contrast, `nw` and `NN` rarely execute with 100% utilized warps. `NN` and `MUM` spend a significant portion of their execution time, 93% and 64%, respectively, executing warps with less than four active threads. On average, across all our workloads, 35% of warps are under-utilized during their execution due to branch divergence. Previous approaches, such as *thread block compaction* [26] or the large warp micro-architecture [71], attempt to address this problem by using active threads of other warps to fill the idle lanes of an under-utilized warp. The effectiveness of such methods, however, is limited for three main reasons: (1) active threads selected for compaction should belong to the same CTA in order to meet the programming model constraints [26], such as data sharing and synchronization capabilities in a CTA, (2) active threads selected for compaction should have the same program counter value due to the SIMT execution model [26], and (3) active threads selected for compaction should *not* belong to the same SIMD lane; otherwise, fine-grained relocation of threads execution lanes is required, which requires significant hardware overhead [26, 27, 85]. Our experimental results attest to the limited potential of thread block compaction [26], and show that its average improvement in SIMD utilization is around 9%.

3.1.2 Full-Lane Idleness. This happens when there is no active warp to be scheduled for execution, or to replace the currently stalled warp. Executing warps can get deactivated for several reasons. Memory divergence [8–13, 72, 100], as one of the main causes, happens when warps execute a memory instruction. Since threads inside a warp could request to different blocks in memory, each access can hit in a different level of the memory hierarchy, such as the L1 cache, the L2 cache, or the off-chip memory, resulting in memory access latency variation inside a warp. Ultimately, a warp is stalled until *all* of its requests, no matter how long each takes, are serviced [9–11, 38]. In addition to memory divergence, inter-warp synchronization and resource contention are two other main causes of warp deactivation [39, 101]. Figure 5 shows the fraction of run-time in which the entire set of execution units is inactive. As can be seen in this figure, GPU execution units experience full-lane idleness for more than 42% of application run-time, on average.

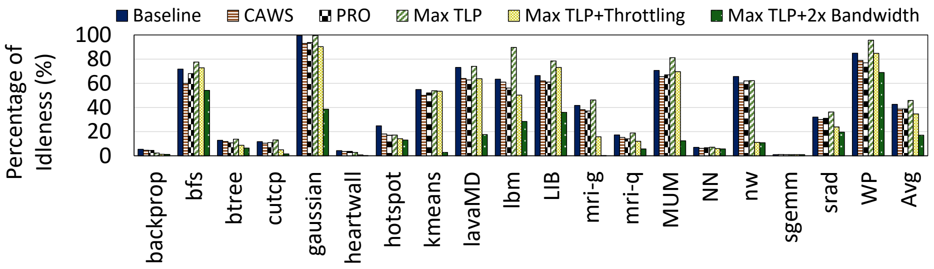

Fig. 5. Fraction of run-time with no active execution lane using different baselines.

Improving the TLP cannot completely eliminate the execution units' stalls. This is due to the fact that more TLP results in higher contention over different GPU resources, such as the data cache (a.k.a., cache thrashing), on-chip network, and memory controllers, leading to increased average memory access latency and warp stall time [9, 45, 47, 95]. To evaluate how increasing the TLP affects this phenomena, we increase the register-file size, maximum allowed number of CTAs per SM, and shared-memory size to enable the execution of the maximum number of threads in an SM (i.e. 2048 threads in our baseline). We also enable the maximum size of register file and shared memory required for our benchmark suites (2MB and 256KB, respectively), without altering their access latencies. To eliminate any effect that CTA count might have on TLP, we also increase the maximum number of allowed CTAs from 32 to 128 per SM. We then measure the percentage of runtime in which all the execution lanes are idle, referred to as percentage of full-lane idleness under six configurations: (1) baseline architecture, (2) criticality-aware warp scheduler (CAWS) [59] that attempts to prioritize slower warps, (3) progress-aware warp scheduling (PRO) [5] that prioritizes the warps based on their progress, (4) a baseline with maximum TLP (`Max TLP`), (5) a baseline with maximum TLP that is equipped with a state-of-the-art throttling mechanism [45] (`Max TLP+Throttling`) in order to mitigate the contention caused by higher TLP, and (6) a baseline with maximum TLP in which we double the on/off-chip bandwidth (`Max TLP+2x Bandwidth`). Figure 5 illustrates the results.

We make four key observations. First, we find that maximum TLP increases the percentage of full-lane idleness in runtime since it causes the on/off-chip bandwidth to become saturated for most of the workloads. Second, although the state-of-the-art TLP management technique [45] can mitigate the side-effects of higher TLP, there is still over 35% full-lane idleness during runtime, on average. Third, there is still over 38% full-lane idleness during runtime, on average, using CAWS [59] and PRO [5] techniques. Fourth, at the maximum TLP, we cannot fully eliminate full-lane idleness during runtime even if we double on/off-chip bandwidth. We conclude that the issue of having no active execution lane in runtime cannot be simply eliminated by allowing for more resources, and demands a different approach.

3.2 Inefficiency of Previous Techniques

To evaluate the opportunity of power-gating (PG), we analyze the length of idle periods. We measure the length of idle periods for each lane and report their distribution in Figure 6. As shown in this figure, the length of idle periods in most of the workloads is shorter than T_{break_even} of the PG technique (i.e., 14 cycles in our experiments), except for `gaussian` and `bfs` where 70% and 25% of idle periods are longer than T_{break_even}, respectively.

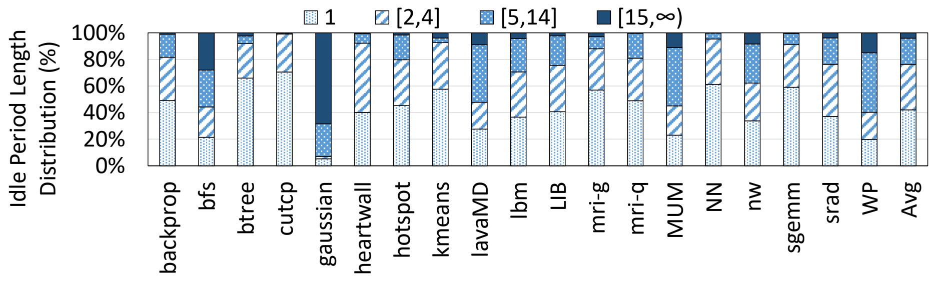

Fig. 6. Distribution of idle period lengths.

We conclude that although GPU execution units experience a significant amount of stalls during runtime, the idle period lengths are usually shorter than T_{break_even}. Hence, PG on its own is *not* effective in reducing static power consumption of execution units in GPUs. In the remainder of this section, we explain different approaches for reducing the corresponding performance and power overheads of PG.

Thread permutation has traditionally been used for improving the efficiency of techniques tackling the *branch divergence* problem, such as thread block compaction or the large warp micro-architecture [26, 27, 71, 85]. As another application, thread permutation can also be employed to opportunistically keep the active mask similar across multiple warps. For example, the active masks of 11001010 and 00010111 can both be changed to 11110000 under ideal thread permutation. This can potentially reduce the number of transitions between active and idle for each lane, decreasing both power and performance overheads of PG. To measure the effectiveness of this approach, we implement ideal fine-grained thread permutation. In ideal permutation, active masks with an equal number of 1s are considered to be exactly the same. Note that we assume ideal permutation is implemented with no overhead. Figure 7 compares the idle period distribution of GPU execution units, averaged across several workloads, with and without the ideal fine-grained permutation method in Figure 7. As can be seen in this figure, the portion of idle periods longer than T_{break_even} increases by a mere 1.04% with thread permutation. This small increase in idle period length using ideal permutation is mainly due to the fact that consecutive active masks usually do *not* have an equal number of active threads, as a warp executing a conditional branch instruction is usually split into two warps with different numbers of active threads. Therefore, we conclude that ideal permutation is *not* effective in mitigating PG overheads.

Warp scheduling is another solution that aims to modify the warp scheduler to give priority to warps with active masks that are similar to the one that is currently in execution. This method can potentially defragment the idle periods, and decrease the power and performance overheads of

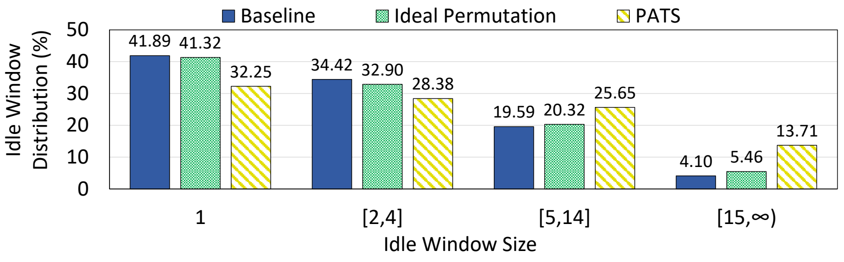

Fig. 7. Effect of ideal permutation and *PATS* [105] on the distribution of the idle period lengths.

PG. To this end, Pattern-Aware Two-level Scheduler (*PATS*) [105] proposes a novel scheduler to select warps with similar branch divergence patterns. *PATS* assumes that there are five dominant branch divergence patterns among different applications, which can be recognized during execution and used to schedule warps with similar active masks. However, we observe that the assumption does not hold for a broad range of GPGPU applications. Additionally, the active mask patterns of workloads with data-dependent branch conditions such as `bfs` [18] are fundamentally hard to predict. Figure 7 shows the average distribution of idle period lengths using the *PATS* technique [105]. *PATS* outperforms the ideal thread permutation with respect to defragmenting idle periods, but still falls short of enabling long idle periods, with more than 86% of idle periods still smaller than T_{break_even}.

We find that the main reason for sub-optimality in static power reduction of GPU execution units in prior proposals [34, 105], is because each past proposal uses *only one* power management technique, such as power-gating. Employing different static power reduction techniques based on the idle period length can significantly lower static power consumption.

3.3 Goals and Summary

In this paper, we aim to improve static power consumption in GPU execution units by addressing three key issues: First, GPU execution units experience a significant amount of idle time. Second, the total idle time is fragmented into frequent but short periods. Third, even with the help of the state-of-the-art idle period defragmentation technique [105], PG is still not effective for >85% of idle periods.

To this end, we propose *ITAP*, a novel approach to reduce the static power consumption of GPU execution units. *ITAP* combines several levels of VS as well as PG in order to cover 100% of idle periods. *ITAP* can dynamically switch between different power reduction modes, based on its estimation of idle period length, and with careful attention to the varying overheads and savings of each mode.

4 IDLE-TIME-AWARE POWER MANAGEMENT

ITAP is designed based on two key contributions. First, we analyze the efficiency of different static power reduction techniques for various lengths of idle periods. We use this analysis to select the static power reduction modes employed in *ITAP*. Second, we estimate the idle period length using our prediction and peek-ahead techniques. Based on these estimations, we apply the most suitable power management mode according to the runtime behavior of applications. We first provide an analysis of the opportunities for different power management policies (Section 4.1). We then describe the mechanism to determine the length of idle period in execution units (Section 4.2). We next discuss how *ITAP* can be used for different goals (Section 4.3). Finally, we explain how *ITAP* can be implemented at different granularities (Section 4.4).

4.1 Multi-Mode Power Management

ITAP benefits from multiple power management techniques, including VS with various scaled voltages and PG, in order to cover 100% of idle period lengths in an efficient way. We estimate the static energy consumed when an execution unit is in power reduction mode with the following equation:

$$E = Cycles_{idle} \times Power_{static} + E_{wake-up} \tag{1}$$

where $Cycles_{idle}$, $Power_{static}$, and $E_{wake-up}$ denote the idle period length, the normalized static power consumption in the power reduction mode, and the energy consumed in order to return to the fully-functional mode, respectively. To measure $Power_{static}$ and $E_{wake-up}$ for VS and PG techniques, we apply the VS and PG techniques to 1) different cells of the NanGate 45nm Open Cell Library [57], 2) a fanout-of-4 (FO4) inverter,[1] and 3) several important digital circuits (e.g., carry look-ahead adder, ripple-carry adder, array multiplier, multiplexer, and decoder) built using the NanGate 45nm Open Cell Library. This setup allows us to observe the trends in energy savings by applying VS and PG techniques to real cores [31]. We simulate the SPICE netlist using HSPICE. *VDD* is set to 1v in this study. As the input data can affect the static power consumption of each logic structure significantly, we calculate the static energy consumption using up to 1024 random input data vectors for each logic structure.[2] Figure 8 compares the reduction in static energy consumption averaged among NanGate cells, a chain of FO4 inverters, and important digital circuits in different static power reduction modes, while varying the idle period length. The results are normalized to the average static energy consumption without a power management technique.

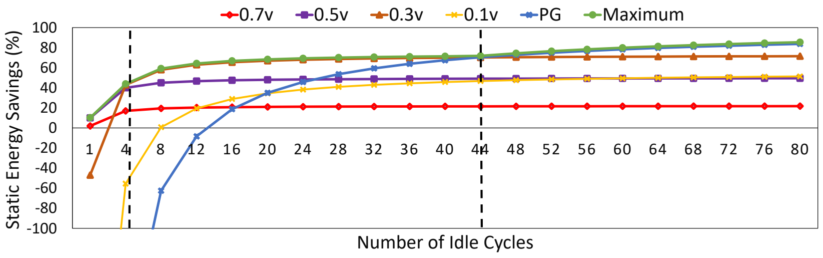

Fig. 8. Normalized energy consumption based on the power reduction mode and the idle period length.

We make five observations based on data in Figure 8. First, $\mathbf{VS_{0.5}}$ and $\mathbf{VS_{0.7}}$ modes (i.e., VDD is set to 0.5v and 0.7v, respectively) are useful for *all* range of idle period lengths because T_{break_even} for $\mathbf{VS_{0.5}}$ and $\mathbf{VS_{0.7}}$ modes are only one clock cycle. However, to reduce the complexity of our design, *ITAP* does not employ $\mathbf{VS_{0.7}}$ and only employs a more effective $\mathbf{VS_{0.5}}$ for idle periods smaller than four clock cycles, which we refer to as the *short* idle periods. Second, when the idle period length is between 4 and 44 cycles, which we refer to as the *medium* idle period, $\mathbf{VS_{0.3}}$ (i.e., VDD is set to 0.3v) is effective at further reducing the energy consumption. Third, we observe that $\mathbf{VS_{0.1}}$ (i.e., VDD is set to 0.1v) is inefficient compared to $\mathbf{VS_{0.3}}$ for any idle period length for two reasons: the execution units in the $\mathbf{VS_{0.1}}$ mode consume more static power compared to the $\mathbf{VS_{0.3}}$ mode (as leakage current increases exponentially [32, 43, 48]) and the $E_{overhead}$ for the $\mathbf{VS_{0.1}}$ mode is larger than that of the $\mathbf{VS_{0.3}}$ mode. Fourth, **PG** is the best choice when the idle period length is greater than 44 cycles, i.e., for *long* idle periods. Finally, although the T_{break_even} of the PG mode is approximately 14 cycles, PG mode starts to outperform the $\mathbf{VS_{0.5}}$ and $\mathbf{VS_{0.3}}$ modes only when

[1]A FO4 inverter is an inverter that drives 4 similar inverters.
[2]The number of random input data vectors is the same as the circuit's input data vectors for circuits with less than 1024 input vectors. For example, a 4-input *AND* cell has 16 different input vectors. In such cases, we evaluate all the input vectors using 16 input data vectors of the circuit.

the idle period length is about 1.8× and 3.1× of PG's T_{break_even}, respectively. Altogether, *ITAP* benefits from three static power reduction modes including $\mathbf{VS_{0.5}}$, $\mathbf{VS_{0.3}}$, and **PG** for *short*, *medium*, and *long* idle periods, respectively.

4.2 Estimating the Idle Period Length

Because predicting the exact value for idle period length is not only difficult, but also expensive, *ITAP* relies on a coarser grain classification of idle periods into three categories: *short*, *medium* and *long*. To this end, *ITAP* employs a prediction method in order to estimate the idle period type. We use a parameter, called $Activity_{cur}$, that shows whether or not an execution lane is currently active. When an execution lane becomes idle (i.e., $\neg Activity_{cur}$), *ITAP* predicts a short idle period and immediately applies $\mathbf{VS_{0.5}}$ mode to the idle lane. We use a parameter, called $Length_{idle}$, that shows idle time of a lane in terms of number of cycles. If the lane remains idle for four clock cycles (i.e., $Length_{idle} == 4$), *ITAP* decides whether to keep $\mathbf{VS_{0.5}}$ mode or change it to either $\mathbf{VS_{0.3}}$ or **PG** mode. It is only beneficial to change the state *only if* the lane will remain idle for at least *four* more clock cycles. Otherwise, *ITAP* should keep the current mode. To decide between keeping and changing the static power reduction mode, *ITAP* employs a *mode-change* saturating counter ($Counter_{mode}$). If the value of $Counter_{mode}$ is larger than a defined threshold (Thr_{switch}), *ITAP* switches the mode to either $\mathbf{VS_{0.3}}$ or **PG** mode. *ITAP* updates $Counter_{mode}$ when the lane becomes active again, in order to maintain the history of the correct/incorrect decisions. We increment $Counter_{mode}$ if the lane remains idle for at least 8 (i.e., 4+4) cycles to improve the chance of selecting either $\mathbf{VS_{0.3}}$ or **PG** mode. On the other hand, we decrement $Counter_{mode}$ if the idle time of the lane is less than 8 cycles, to improve the likelihood of choosing the $\mathbf{VS_{0.5}}$ mode. The next design challenge is how to select the power reduction mode between $\mathbf{VS_{0.3}}$ and **PG** modes when *ITAP* decides to change the power reduction mode from $\mathbf{VS_{0.5}}$.

ITAP employs a *confidence* saturating counter ($Counter_{conf}$) to differentiate between *medium* and *long* idle periods. If the value of $Counter_{conf}$ is lower than a defined threshold (Thr_{long}), *ITAP* applies the $\mathbf{VS_{0.3}}$ mode to the idle execution lane. If the value of $Counter_{conf}$ is greater than a defined threshold (Thr_{long}), *ITAP* applies the **PG** mode to the idle execution lane. *ITAP* decrements $Counter_{conf}$ if the idle time is less than 48 (i.e., 4+44) cycles, to improve the chance of selecting the $\mathbf{VS_{0.3}}$ mode. On the other hand, *ITAP* increments $Counter_{conf}$ if the idle time is larger than 48 cycles, to increase the chance of using the **PG** mode. Figure 9 depicts the states and transitions used in *ITAP*'s finite state machine (FSM). Table 1 (the second column) describes the detail of each transition. For *ITAP* without peek-ahead, the state transition is triggered by changes in four parameters: $Activity_{cur}$, $Length_{idle}$, $Counter_{mode}$, and $Counter_{conf}$.

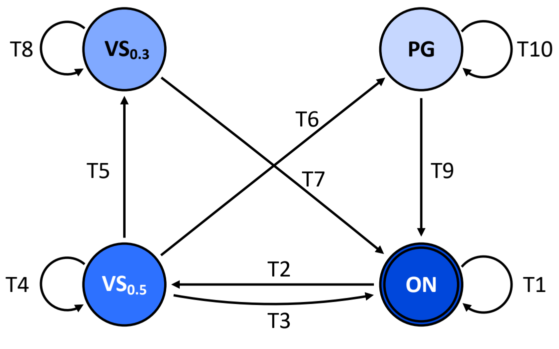

Fig. 9. *ITAP* algorithm FSM.

Table 1. Detail of each transition in *ITAP* with and without peek-ahead.

Transition	*ITAP* without peek-ahead	*ITAP* with peek-ahead
T1	$Activity_{cur}$	$Activity_{cur}$
T2	$\sim Activity_{cur}$	$\sim Activity_{cur}$
T3	$Activity_{cur}$	$Activity_{ahead}[1]$
T4	$\sim Activity_{cur}$ $\& \; [\,(Length_{idle} < 4) \mid (Counter_{mode} < Thr_{switch})\,]$	$\sim Activity_{ahead}[1] \; \& \; Activity_{ahead}[2or3]$
T5	$\sim Activity_{cur} \; \& \; (Length_{idle} \geqslant 4) \; \& \; (Counter_{mode} \geqslant Thr_{switch})$ $\& \; (Counter_{conf} < Thr_{long})$	$\sim Activity_{ahead}[1] \; \& \sim Activity_{ahead}[2or3]$ $\& \; (Counter_{conf} < Thr_{long})$
T6	$\sim Activity_{cur} \; \& \; (Length_{idle} \geqslant 4) \; \& \; (Counter_{mode} \geqslant Thr_{switch})$ $\& \; (Counter_{conf} \geqslant Thr_{long})$	$\sim Activity_{ahead}[1] \; \& \sim Activity_{ahead}[2or3]$ $\& \; (Counter_{conf} \geqslant Thr_{long})$
T7	$Activity_{cur}$	$Activity_{ahead}[1or2]$
T8	$\sim Activity_{cur}$	$\sim Activity_{ahead}[2or1]$
T9	$Activity_{cur}$	$Activity_{ahead}[1or2or3]$
T10	$\sim Activity_{cur}$	$\sim Activity_{ahead}[1or2or3]$

In the aforementioned mechanism, *ITAP* updates the *mode-change* and *confidence* counters using simple increment/decrement operations. However, there are other possibilities to update these counters when the correct idle time is known. For example, as the overhead of **PG** mode is much larger than the overhead of $\mathbf{VS_{0.3}}$ mode, it may make sense to reset the *confidence* counter to zero when **PG** prediction is incorrect. We analyze the impact of the size of *mode-change* and *confidence* counters, Thr_{switch}, Thr_{long}, and two other counter update approaches on the prediction accuracy of the prediction unit in Section 6.4.1.

Peek-Ahead Window. the aforementioned prediction mechanism has two main issues. First, if the idle period length is larger than four clock cycles, it first selects the $\mathbf{VS_{0.5}}$ mode and then may change the mode to either $\mathbf{VS_{0.3}}$ or **PG** mode based on the value of *mode-change* and *confidence* counters after four clock cycles elapse. However, when the idle period is longer than four clock cycles, *ITAP* needs to apply either $\mathbf{VS_{0.3}}$ or **PG** mode as soon as possible. Second, *ITAP* cannot figure out when an execution lane should be woken up ahead of time, and hence, *ITAP* imposes performance overhead due to the T_{wake_up} of static power reduction mode. To efficiently address these issues, *ITAP* combines our prediction technique with a peek-ahead technique.

Our peek-ahead technique employs a short peek-ahead window in order to figure out the state of the execution lane activity in the near future. Because this peek-ahead window requires additional overhead, we limit the peek-ahead window to the next three cycles in order to compensate for the maximum wake-up latency of three cycles required by *ITAP*. We implement the peek-ahead technique by modifying the round-robin warp scheduler employed for scheduling warps to the execution units. The baseline warp scheduler selects a warp from a warp pool that has ready operands in the operand collectors to be issued for the execution in a round-robin manner. In order to detect the lane idleness behavior that occurs three cycles later, we modify the baseline warp scheduler such that it determines two future warps to schedule in addition to the currently-selected warp. As a result, our peek-ahead technique can potentially detect the active masks within three cycles. *ITAP* with peek-ahead employs states and transitions that are the same as those of *ITAP* without peek-ahead, as shown in Figure 9. However, state transition conditions and actions are different for *ITAP* with peek-ahead, as shown in the third column of Table 1. *ITAP* with peek-ahead adds an additional parameter compared to *ITAP* without peek-ahead, $Activity_{ahead}$, to the condition for state transition. $Activity_{ahead}$ is a 3-entry array used for each execution lane, where $Activity_{ahead}[i]$ shows whether or not the lane is active i cycle(s) later.

There are some situations in which peek-ahead technique fails. For example, when there is no active warp to schedule, the peek-ahead technique in *ITAP* cannot specify the activity of the next three cycles. Another example is that when there is only one active warp with a memory/branch instruction, predicting the state of the execution unit after executing this warp instruction is difficult. Therefore, *ITAP* uses the peek-ahead technique only when the peek-ahead provides valid information. Otherwise, *ITAP* makes a decision based on only its simple prediction logic.

We implement both algorithms (*ITAP* with and without the peek-ahead technique) using finite state machines. We synthesize the hardware-description language (HDL) model of both finite state machines for 45nm NanGate open cell library using the Synopsys Design Compiler. We observe that both finite state machines can operate properly at 2.5 GHz clock frequency. Therefore, selecting a power reduction mode among $\mathbf{VS_{0.5}}$, $\mathbf{VS_{0.3}}$, or **PG**, which is translated to update the current state in the finite state machine, can be performed in one GPU clock cycle (i.e., 1.4 GHz).

4.3 Different Optimization Goals

ITAP allows the GPU to optimize for two different optimization goals: *performance-aggressive* and *power-aggressive*. When the goal is to be performance-aggressive, *ITAP* should *not* impose performance overhead to the system. We make two decisions for the *performance-aggressive* use of *ITAP*. First, *ITAP* attempts to issue the wake-up command earlier based on T_{wake_up} of the current power-management mode. As an example, if the idle lane is in $\mathbf{VS_{0.3}}$ mode, *ITAP* issues the wake-up command two cycles earlier using the peek-ahead technique. Second, *ITAP* significantly reduces the likelihood of using the **PG** mode by resetting the confidence counter to zero when the **PG** mode is wrongly used. For *power-aggressive* optimization, on the other hand, the main goal is to aggressively reduce the power consumption at the expense of incurring some performance overhead. For this goal, we switch off early wake-up in *ITAP* in order to keep the idle lanes in the power-reduction mode as much as possible. Moreover, we update the confidence counters via simple increment/decrement operations, to increase the likelihood of the **PG** mode. We compare these two design approaches in terms of static energy savings and performance in Section 6.3.

4.4 *ITAP* Granularities

ITAP can be implemented in different granularities. The most fine-grained implementation is to apply a power reduction mode for each lane individually based on the lane's idle period length. As a result, it is possible to have some lanes active, some lanes in $\mathbf{VS_{0.5}}$, some lanes in $\mathbf{VS_{0.3}}$, and some lanes in **PG** modes. This implementation maximizes the opportunity for static power reduction at the price of higher design cost due to the fine-grained on-chip power management circuits, such as on-chip voltage regulators and PG sleep transistors. The most coarse-grained implementation, on the other hand, is to apply the same power reduction mode for *all* of idle lanes. In this implementation, *ITAP* first selects the best power reduction mode for each idle lane based on the aforementioned mechanism. *ITAP* then unites the power reduction mode for all idle lanes based on their individual modes, as described in Table 2.

Table 2. *ITAP* power management mode selection policy in its most coarse-grained design.

Condition	**Power Mode Selection**
Any idle execution lane is selected to be in $\mathbf{VS_{0.5}}$ mode	$\mathbf{VS_{0.5}}$
No idle execution lane in $\mathbf{VS_{0.5}}$ & at least one idle execution lane in $\mathbf{VS_{0.3}}$	$\mathbf{VS_{0.3}}$
All idle execution lanes are selected to be in **PG** mode	**PG**

The policy provided in Table 2 ensures that the power reduction mode does not impose performance and energy overheads in the coarse-grained implementation. The coarse-grained implementation reduces the hardware cost at the price of decreasing the opportunity of static power reduction. Other granularities between the most fine-grained and course-grained implementations are also possible in *ITAP*. For example, *ITAP* can cluster four lanes and apply the same power reduction mode for each cluster. We analyze the effect of implementation granularity of *ITAP* on the static energy savings and performance in Section 6.4.2.

5 EVALUATION METHODOLOGY

We describe the methodology used for our experimental evaluation and analysis.

Simulator. We evaluate the performance of *ITAP* using GPGPU-Sim 3.2.2 [14]. We evaluate the power consumption of *ITAP* using GPU-Wattch [60]. Table 3 shows the simulation parameters modeling the NVIDIA Pascal architecture [76], which are consistent with prior work [1–3, 24, 105]. We use HSPICE to measure the amount of static power reduction (P_{static}-reduction), T_{break_even}, and T_{wake_up} for each static power reduction mode. We report these numbers in Table 4.

Table 3. Baseline simulated GPU configuration.

Parameters	**Value**
SMs	16, 1400MHz, SIMT Width = 32
Resources per SM	max 2048 Threads, 65536 Registers, max 32 CTAs, 64KB Shared Memory
Warp Schedulers	2 per SM, two-level round-robin [71]
Cache	32KB/SM 4-way L1 cache, 256KB/Memory Channel 8-way L2 cache
Memory Model	GDDR5 1674 MHz [94], 8 channels, 8 banks per rank, 1 rank, FR-FCFS scheduler [86, 108]
GDDR5 Timing	t_{CL}=12, t_{RP}=12, t_{RC}=40, t_{RAS}=28, t_{RCD}=12, t_{RRD}=6 [14]

Table 4. Parameters of the *ITAP* power reduction modes.

Mode	**Specifications**
$VS_{0.5}$	T_{break_even} =1 cycle, Idle length=1-3 cycles T_{wake_up}=1 cycle, E_{wake_up}=40%, P_{static}-reduction=50%
$VS_{0.3}$	T_{break_even} =2 cycles, Idle length=4-43 cycles T_{wake_up}=2 cycles, E_{wake_up}=120%, P_{static}-reduction=73%
PG	T_{break_even} =14 cycles, Idle length >=44 cycles T_{wake_up}=3 cycles, E_{wake_up}=1300%, P_{static}-reduction=100%

Workloads. We evaluate the effectiveness of *ITAP* on Rodinia [18], Parboil [96], and ISPASS [14] benchmark suites. Each workload is either simulated entirely, or until it reaches 2 billion executed instructions, whichever comes first.

Comparison Points. We compare the most coarse-grained implementation of *ITAP* to conventional PG (*CPG*) and a state-of-the-art scheduler-aware PG technique [105] (*PATS*). *CPG* power-gates lanes that are idle for more than T_{idle_detect} (e.g., 5 cycles). *PATS* attempts to defragment idle periods in order to improve the likelihood and opportunity of using the PG technique. In addition, we evaluate the combination of *ITAP* and *PATS* (*ITAP+PATS*) with other techniques, to quantitatively show how much the efficiency of *ITAP* is improved when it is combined with a state-of-the-art idle period defragmentation technique. We also implement the ideal version of *ITAP*, called *ITAP-ideal*, to evaluate the accuracy of *ITAP* decisions. To this end, we record a trace of all the warps' active masks along with their issue cycles. We use this trace to analyze the *future* lane activity at each decision point for changing power modes, and deciding the best mode based on the future information in the trace. Moreover, in order to quantitatively show the effect of the peek-ahead technique on the efficiency of *ITAP*, we implement *ITAP* without the peek-ahead technique (called *ITAP-WO-PeekAhead*), in our simulation environment. Finally, *ITAP* can target two different optimization goals, power-aggressive and performance-aggressive. We implement both $ITAP_{pow}$ and $ITAP_{perf}$ for power-aggressive and performance-aggressive goals, respectively.

***ITAP* Parameters.** We empirically set the size of *mode-change* and *confidence* counters to 8 bits (see Section 6.4.1 for more details). We set Thr_{switch} and Thr_{long} at 50% of the maximum values of the corresponding counters, to simplify our design.[3]

[3]We observe that the threshold values have negligible effect on prediction accuracy in Section 6.4.1.

Hardware Overhead. There are six components that contribute to the hardware overhead of *ITAP*. First, *ITAP* employs two 8-bit counters for *mode-change* and *confidence* counters for each execution lane. Second, *ITAP* needs to maintain the current idle period length for each execution lane in order to improve the accuracy of the prediction unit. A 6-bit saturating counter is large enough to maintain the idle period length for each execution lane. Third, we need to evaluate the idle period length to update the *mode-change* and *confidence* counters. To this end, we use a 6-bit comparator for each execution lane. Fourth, we check the threshold values by evaluating the most-significant-bit of each counter. Fifth, we use 2-bit register for each execution lane to track the lane's current power mode. Sixth, we modify the two-level round-robin warp scheduler [71] and add a buffer with three 32-bit slots to implement the peek-ahead technique.

To measure the area and power overheads of *ITAP*, we implement the HDL model of prediction/peek-ahead techniques and synthesize them for the 45nm NanGate open cell library [57] using Synopsys Design Compiler [21]. Our hardware implementation of VS and PG techniques follows the optimizations in previous work [1–3, 6, 19, 88, 105], minimizing the hardware overhead. To implement **PG**, we add a sleep transistor to each execution lane. To implement **VS**, we use on-chip voltage regulators to generate 0.5v and 0.3v input voltages. Depending on the granularity of our design, one voltage regulator is added to each SM (most coarse-grained), or each execution lane (most fine-grained).

As a result, the granularity of *ITAP* presents a trade-off between static energy savings and hardware implementation overheads. While the most fine-grained approach may yield higher static energy savings, it also leads to larger power and area overheads, i.e., one voltage regulator for each execution lane. Implementing *ITAP* in its most coarse-grained form allows us to minimize such overheads by amortizing the cost of voltage regulators and sleep transistors across the entire set of SIMD lanes. We estimate the overhead of PG (i.e., adding sleep transistors) and VS (i.e., voltage regulators) techniques using prior work [6, 66, 70].

Table 5 shows the summary of required resources and power/area overheads for different *ITAP* designs. We make two key observations. First, the coarse-grained design has significantly lower power and area overheads compared to the fine-grained design. This is mainly due to the fact that adding a voltage regulator per execution lane leads to a significant overhead, which is not scalable with current technologies [6]. Nevertheless, we implement the most fine-grained approach to measure the impact of granularity on our method, and the energy efficiency gap between the coarse-grained and fine-grained approaches.[4] Second, adding the peek-ahead technique on top of both *ITAP* designs (i.e., coarse-grained and fine-grained designs) has negligible area and power overheads.

6 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the effectiveness of *ITAP*. Section 6.1 evaluates the energy consumption of *ITAP* relative to other state-of-the-art baselines. Section 6.2 shows that *ITAP* has minimal effect on GPU performance. Section 6.3 shows the power and performance tradeoff between our *power-aggressive* and our *performance-aggressive* designs. Section 6.4 provides the sensitivity analysis to the design parameters of *ITAP*.

[4]Note that we use the power overhead of the most coarse-grained design in our energy analysis. However, for the most fine-grained design, we assume that the power overhead is equal to that of the most coarse-grained design, in order to solely evaluate the effect of granularity in *ITAP* without penalizing the fine-grained design for power consumption.

Table 5. Summary of required resources and the overall area and power overheads for different *ITAP* designs.

Resource	coarse-grained	coarse-grained+peek-ahead	fine-grained	fine-grained+peek-ahead
Mode-change	lane-size × 8-bit counters			
Confidence	lane-size × 8-bit counters			
Idle cycles	lane-size × 6-bit counters			
Comparing #idle-cycles	lane-size × 6-bit comparators			
Current mode	lane-size × 2-bit registers			
Modified warp scheduler	No	Yes (3 × 32-bit registers)	No	Yes (3 × 32-bit registers)
Global decision maker	Yes	Yes	No	No
PG switch	1	1	lane-size	lane-size
VS regulator	1	1	lane-size	lane-size
Overall power overhead (lane-size = 32)	2.208%	2.21%	70.408%	70.41%
Overall area overhead (lane-size = 32)	0.325%	0.33%	9.725%	9.73%

6.1 Energy Analysis

To show the efficiency of *ITAP*, we measure the static energy savings of execution units using different techniques. Figure 10 shows the static energy savings of execution units using *CPG* [34], *PATS* [105], *ITAP*$_{\text{pow}}$-WO-PeekAhead (i.e., power-aggressive *ITAP* w/o peek-ahead), *ITAP*$_{\text{pow}}$, *ITAP*$_{\text{pow}}$-ideal (i.e., power-aggressive *ITAP* that uses perfect, i.e., 100%-accurate, idle time prediction), and *ITAP*$_{\text{pow}}$+*PATS* techniques. Note that we compare different design approaches of *ITAP*, *ITAP*$_{\text{pow}}$ and *ITAP*$_{\text{perf}}$, in Section 6.3.

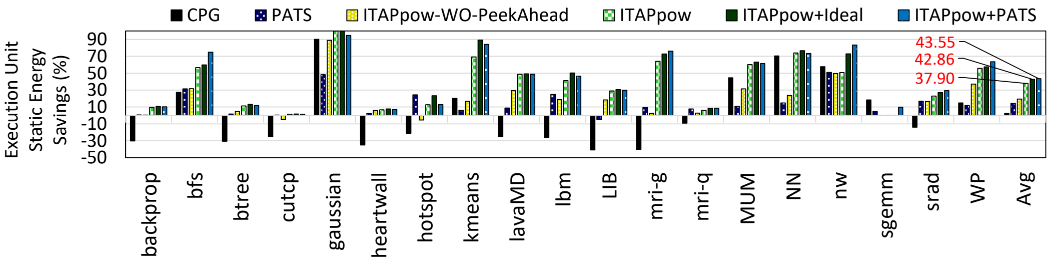

Fig. 10. Static energy savings using different power reduction techniques.

We make four key observations. First, *ITAP*$_{\text{pow}}$ outperforms *CPG* and *PATS* in terms of static energy savings by an average of 36.3% and 27.6%, respectively. Second, the contribution of the peek-ahead technique on the efficiency of *ITAP* is significant. Comparing the static energy savings of *ITAP*$_{\text{pow}}$ with and without the peek-ahead technique shows that peek-ahead improves the static energy savings by 24.3%, on average. Second, implementing *ITAP* on top of a state-of-the-art idle period defragmentation technique, *PATS*, improves the static energy saving by an average of 9.1%. This is due to the fact that *PATS* defragments the idle periods and improves the opportunity of applying more powerful static power reduction modes in *ITAP*. Finally, the improvement of *ITAP*$_{\text{pow}}$+ideal (i.e., power-aggressive *ITAP* that uses a 100%-accurate idle time prediction) over *ITAP*$_{\text{pow}}$ is less than 8%, which clearly shows the high accuracy of our technique in estimating idle period length. We conclude that *ITAP* is a highly effective approach to reducing the static energy consumption in GPU execution units.

6.2 Performance Analysis

We measure performance on a broad set of workloads using various power management techniques. Figure 11 shows the Instructions per Cycle (IPC) for *CPG*, *PATS*, *ITAP$_{pow}$-WO-PeekAhead*, *ITAP$_{pow}$*, *ITAP$_{pow}$-ideal*, and *ITAP$_{pow}$+PATS* normalized to the baseline architecture with no static power reduction technique. We make four key observations. First, *ITAP$_{pow}$* has smaller performance overhead compared to *CPG* and *PATS*. *ITAP$_{pow}$* reduces performance by up to 2% while *CPG* and *PATS* reduce performance by up to 41% and 12%, respectively. Second, our peek-ahead technique in *ITAP$_{pow}$* improves performance by 17%, on average, compared to *ITAP$_{pow}$-WO-PeekAhead*. Third, although combining *ITAP* and *PATS* effectively reduces the static energy (see Figure 10), it has almost no negative effect on performance. Finally, the ideal implementation of *ITAP* reduces performance by about 0.5%, on average. We conclude that the negative effect of *ITAP* on GPU performance is small or negligible.

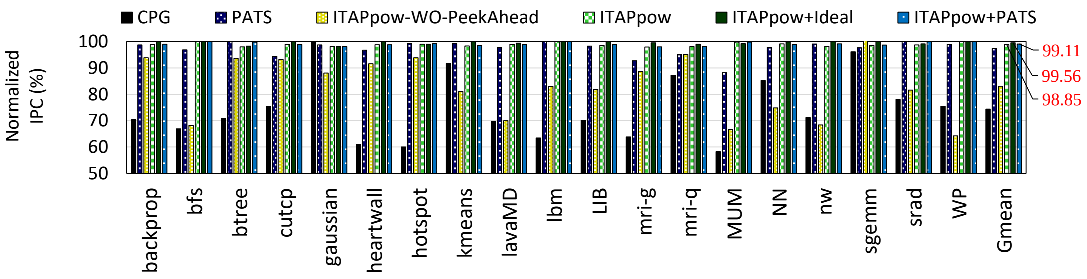

Fig. 11. Normalized IPC using different power reduction techniques.

6.3 *ITAP$_{pow}$* vs. *ITAP$_{perf}$*

In this section, we evaluate the *performance-aggressive* and *power-aggressive* designs of *ITAP*. To show the difference between *ITAP$_{pow}$* and *ITAP$_{perf}$* in applying the static reduction modes, we report the breakdown of the usage of different power reduction modes, averaged across various workloads in Figure 12.

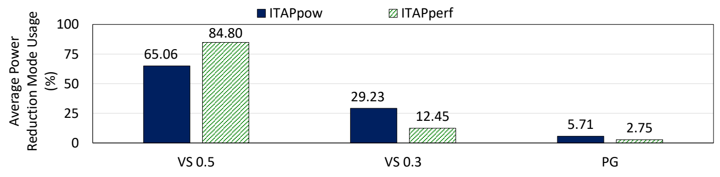

Fig. 12. Power reduction mode usage of *ITAP$_{pow}$* & *ITAP$_{perf}$*.

As Figure 12 shows, *ITAP$_{pow}$* employs **PG** and **VS$_{0.3}$** power reduction modes about 2.1× and 2.3× more frequently than *ITAP$_{perf}$*, respectively. These results clearly show that *ITAP$_{pow}$* attempts to reduce the static power consumption aggressively by applying modes with more static power reduction capability more often.

Figure 13a reports the static energy savings of *ITAP$_{pow}$* and *ITAP$_{perf}$* techniques for different workloads. We see that *ITAP$_{pow}$* improves the static energy savings by 13%, on average, compared to *ITAP$_{perf}$*. However, *ITAP$_{perf}$* is useful when the performance overhead needs to be minimal. To compare the performance overhead of these two techniques, Figure 13b reports GPU throughput (measured using IPC) of *ITAP$_{pow}$* and *ITAP$_{perf}$* normalized to the baseline GPU with no power reduction technique. The performance overhead of *ITAP$_{perf}$* is up to 0.4% (0.2%, on average), which clearly shows that $ITAP_{perf}$ is effective at reducing static energy with minimal performance overhead in every application examined. We conclude that $ITAP_{pow}$ is more effective than $ITAP_{perf}$ at reducing the static energy of the execution units. However, $ITAP_{perf}$ is able to reduce the static energy with almost no performance overhead.

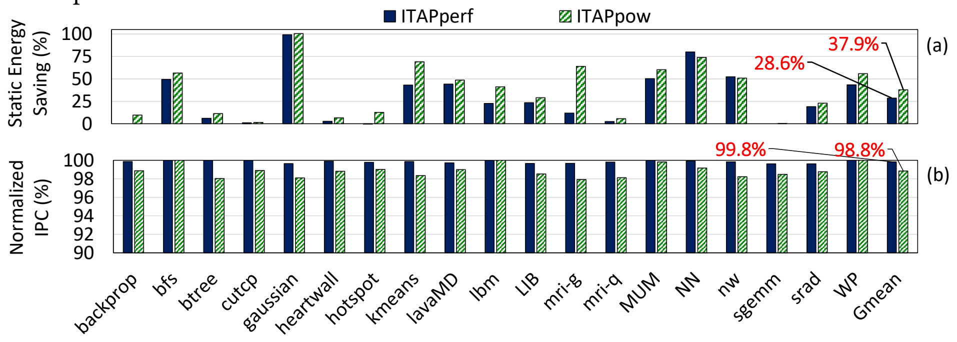

Fig. 13. (a) Static energy saving and (b) Normalized IPC using $ITAP_{pow}$ and $ITAP_{perf}$

6.4 Sensitivity Analysis

We provide the sensitivity analysis of prediction parameters, *ITAP* granularity, SIMD lane size, and PG/VS parameters to *ITAP*.

6.4.1 Effect of prediction parameters. Figure 14 shows the effect of different parameters of *mode-change* and *confidence* counters on idle time prediction accuracy, averaged across all workloads. We vary the size (4, 16, 64 and 256) as well as the threshold values (25%, 50% and 75% of the maximum value) of each counter. We also consider three update mechanisms for the counters: (1) the simple update mechanism we explained in Section 4.2, (2) the same update mechanism with one difference: after two continuous incorrect predictions on the values above the threshold, the counter is reset to zero, and (3) using the simple update mechanism only for counter values below the threshold; otherwise resetting the counter to zero after each wrong prediction.

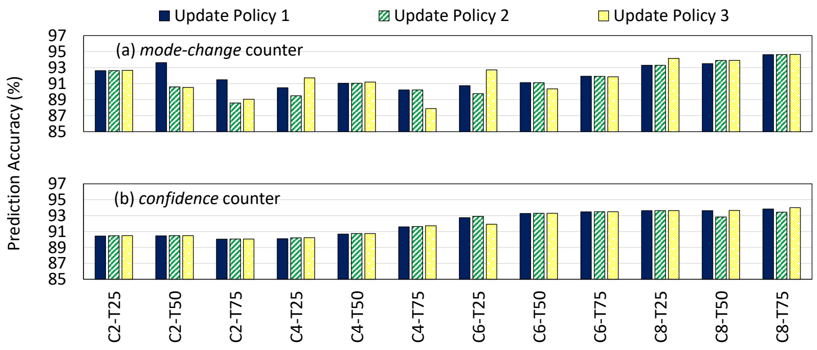

Fig. 14. Effect of counter size (C), threshold value (T), and update policy on prediction accuracy.

We make two key observations from Figure 14. First, using an 8-bit counter size with the threshold value of 50% of the maximum value of the counter represents a good trade-off between prediction accuracy and complexity (i.e., comparing the counter value to the threshold only requires checking the most significant bit of the counter value). Second, the third update policy slightly outperforms the other two for an 8-bit counter size. However, considering the negligible difference between all policies, we can use any one of them effectively.

6.4.2 Effect of ITAP *granularity.* We implement *ITAP* under different granularities (See Section 4.4) to show their effect on GPU performance and static energy. Figure 15 compares the average static energy savings and the average IPC normalized to the baseline GPU with no power reduction technique, for various *ITAP* granularities. The x-axis of Figure 15 shows different implementation granularities where '1' and '32' show the most fine-grained and coarse-grained implementations, respectively.

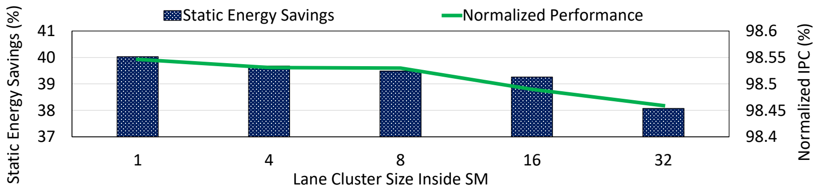

Fig. 15. Effect of *ITAP* granularity on static energy and performance.

We make two observations. First, the effect of *ITAP*'s granularity is negligible on GPU performance. Second, finer granularity *ITAP* designs lead to higher static energy savings because the accuracy of selecting the most proper static power reduction mode increases in fine-grained designs. However, our main evaluation in Sections 6.1 and 6.2 uses the most coarse-grained design because the overhead of required circuits for implementing *ITAP*, such as the PG sleep transistors and on-chip voltage regulators, is significantly lower in the most coarse-grained *ITAP* implementation.

6.4.3 Effect of SIMD lane size. While our main experiment enables all 32 SIMD lanes in the GPU core, it is important to analyze the sensitivity of *ITAP* to different lane sizes in order to evaluate the applicability of *ITAP* to various GPU architectures. SIMD lane size has two main effects on the efficiency of *ITAP*. First, decreasing the lane size increases the accuracy of the peek-ahead window because an instruction is executed over several cycles when the lane size is reduced. Hence, fewer SIMD lanes improve the efficiency of *ITAP*. Second, reducing the lane size has both positive and negative effects on the lane idle period lengths based on patterns of the active masks. To elaborate, assume that a warp with an active mask of 0X0000FFFF is issued for execution. A lane of size 32 causes one-cycle idleness for the lanes numbered 16 to 31. However, there is *no idleness* for the same group of threads on a 4-lane SIMD GPU because the GPU would execute four fully-active cycles for the active threads and then skip the remaining non-active threads. On the other hand, assume that the active mask is 0X88888888. In this example, the warp has only eight active threads. A lane size of 32 causes one-cycle idleness for 24 lanes. However, a lane size of four results in eight-cycle idleness for three lanes (i.e., eight active threads in the warp will be executed in eight consequent cycles in a lane size of 4), greatly improving the opportunity of more powerful static power reduction techniques.

Figure 16 compares the average static energy savings and average IPC normalized to the baseline GPU with no power reduction technique for various lane sizes (4, 8, 16, and 32 lanes), in order to quantitatively evaluate the effect of lane size on the accuracy of our peek-ahead technique and positive/negative changes in idle period lengths caused by reducing the lane size.

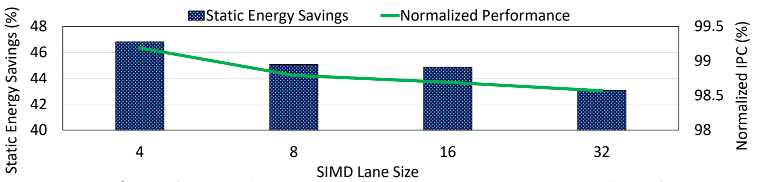

Fig. 16. Effect of SIMD lane size on static energy savings and performance.

We make two key observations. First, reducing the lane size greatly reduces the performance overhead by enabling a more accurate and powerful peek-ahead technique. For the smallest lane size, the performance overhead is less than 1%. Second, *ITAP* on a lane size of four has the highest static energy savings, on average, for our workloads. We conclude that *ITAP* is applicable for different SIMD lane sizes and the efficiency of *ITAP* increases as the lane size reduces.

6.4.4 Effect of T_{break_even} and T_{wake_up}. Because T_{break_even} and T_{wake_up} vary for different GPU architectures, we analyze the sensitivity of *ITAP* to T_{break_even} and T_{wake_up}. Figure 17(a) compares the average static energy savings of *ITAP*, *PATS*, and *ITAP+PATS* over multiple T_{break_even} values with up to a 200% increase in the default T_{break_even} value (i.e., 14 cycles in our experiments). We make three key observations. First, the static energy savings decreases for all techniques by increasing T_{break_even}. This is expected since, by increasing T_{break_even}, the opportunity of reducing the static energy consumption using PG and VS decreases. Second, the improvement of *ITAP* over *PATS* increases for larger T_{break_even} values, as *ITAP* benefits from two extra power reduction modes in addition to the PG technique. Third, the static energy savings of *ITAP+PATS* slightly decreases by increasing T_{break_even} because the defragmentation technique in *PATS* enlarges the idle period lengths and thus compensates for the overhead of larger T_{break_even} values.

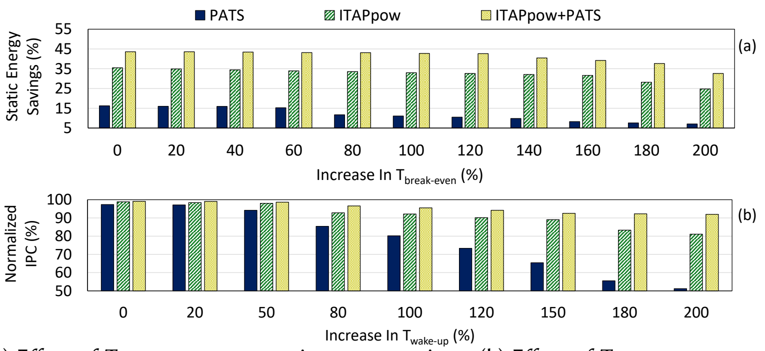

Fig. 17. (a) Effect of T_{break_even} on static energy savings; (b) Effect of T_{wake_up} on performance.

Figure 17(b) compares the average normalized IPC for *ITAP*, *PATS*, and *ITAP+PATS* with various T_{wake_up} values. We increase T_{wake_up} of all power reduction modes by up to 200% over the default values (which are 1, 2, and 3 cycles for the $VS_{0.5}$, the $VS_{0.3}$, and the PG, respectively, in our experiments). We make three observations. First, the performance overhead of all techniques increases with increasing T_{wake_up}. However, *ITAP* provides better performance compared to *PATS* as it uses two other static power reduction techniques with lower performance overhead in addition to the PG technique. Second, the peek-ahead technique embedded in *ITAP* keeps performance almost unchanged until up to a 50% increase in T_{wake_up}, but becomes less efficient as T_{wake_up} increases further, due to the limited peek-ahead window. Third, *ITAP+PATS* outperforms *ITAP* as T_{wake_up} increases because defragmenting idle periods reduces the frequency of waking up from the static power reduction modes.

7 RELATED WORK

To our knowledge, this is the first paper to propose a multi-mode static power reduction technique in order to cover 100% of idle periods in execution units, and thus efficiently reduce the static energy of GPU execution units, based on the idle period length. In the previous section, we provided a thorough comparison of our technique, *ITAP*, to *PATS* [105], a state-of-the-art idle period defragmentation technique. In this section, we describe other related works that leverage **power-gating** and **voltage/frequency scaling** to reduce the static energy consumption.

GPU power-gating. Several works attempt to power-gate idle execution lanes [7, 33, 41, 42, 46]. Warped Gates [2] aims to improve the energy efficiency of GPUs by leveraging the fact that integer and floating-point instructions cannot be executed simultaneously. Therefore, executing one type results in the idleness of the functional units of the other type. Warped Gates modifies the warp scheduler to schedule warps with instructions of the same type as much as possible in order to defragment the idle periods of integer and floating-point units. Aghilinasab et al. [3] propose static instruction reordering to improve the idleness opportunity of the Warped Gates scheduler. *ITAP* is orthogonal to these power-gating techniques as it targets the idleness of every component within a SIMD lane rather than focusing on functional units within the lane.

Several works show that there are some GPU applications with considerable scalar execution, in which, the same instruction is executed on the same operands, or almost the same operands [84], among threads in a warp [30, 63, 83, 84]. Based on this observation, these works modify the GPU execution pipeline to support scalar execution. Hence, other inactive lanes can be power-gated/clock-gated during scalar execution. However, scalar execution is not common enough among different GPU applications and thus the benefit of such techniques is limited to workloads with significant scalar execution. *ITAP* targets two main sources of execution unit idleness, partial- and full-lane idleness, which happen frequently across a wide variety of GPU workloads.

Kayiran et al. [46] propose µC-States, a power management method that applies power-gating/clock-gating techniques to different GPU datapath components. µC-States targets big-cores in modern GPUs with several SIMD pipelines in each SM. In particular, based on their utilization rate, µC-States power-gates some of the SIMD pipelines to improve the energy efficiency of modern GPUs with big cores. However, in the energy-efficient state, there is at least one functional SIMD pipeline to guarantee progress of application. In comparison to µC-States, *ITAP* is more general as it can be employed for static power reduction for SMs with any number of SIMD pipelines.

GPU voltage/frequency scaling. Commercial GPUs provide control to dynamically change the voltage and the frequency of SMs [73, 75, 77]. They scale frequency and voltage based on the total power budget and temperature restrictions of the chip [73, 75, 77]. *ITAP* has the ability to improve power efficiency in a more fine-grained manner, targeting execution units inside SIMD lanes. Additionally, *ITAP* can be more effective when SMs run under hard power constraints, as it can capture idle periods and further improve power efficiency of the hardware.

Several proposals scale voltage/frequency at different granularities in order to decrease consumed power [15, 37, 58, 60, 61, 73, 80, 95]. All of these works scale the supply voltage and the working frequency [37, 58, 60, 73] at the same time or scale only the working frequency [61, 80, 95] in order to keep the units in the functional state. *ITAP*, on the other hand, employs voltage-scaling for *idle periods* of execution lanes. Hence, there is no need to scale the working frequency to keep the lanes in their functional state. Moreover, *ITAP* benefits from power-gating in addition to voltage-scaling for large idle periods.

CPU and DRAM power management. Several works apply power-gating and voltage/frequency scaling in the context of CPUs [17, 22, 24, 31, 34, 35, 62–65, 79] and DRAM [16, 22, 23]. Power management techniques in such other contexts cannot be simply applied to GPUs. To have an efficient power management technique for GPUs, it is essential to optimize the power reduction techniques by thoroughly considering the GPU context, such as micro-architecture, execution model, and the characteristics of idle periods.

8 CONCLUSION

We propose *ITAP* (Idle-Time-Aware Power Management) to efficiently reduce the static power consumption of GPU execution units, by exploiting their idleness. *ITAP* employs three static power reduction modes, each with different static power reduction abilities and overheads to reduce power consumption of idle periods in an efficient manner. *ITAP* estimates the idle period length using prediction and peek-ahead techniques, and applies the most suitable power reduction mode to each idle execution lane. Our experimental results show that the *power-aggressive* design of *ITAP* outperforms the state-of-the-art solution by an average of 27.6% in terms of static energy savings, with up to 2.1% performance overhead (1.2%, on average). On the other hand, the *performance-aggressive* design of *ITAP* improves the static energy savings by an average of 16.9%, while having negligible impact on GPU performance (i.e., up to 0.4% performance overhead), compared to the state-of-the-art static energy savings mechanism. We conclude that *ITAP* provides an effective framework for reducing static power consumption in modern GPUs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers, members of the HPCAN research group and the SAFARI research group for their valuable feedback. Special thanks to Tracy Ewen for proofreading.

REFERENCES

[1] Mohammad Abdel-Majeed and Murali Annavaram. 2013. Warped Register File: A power efficient register file for GPGPUs. In *HPCA*.

[2] Mohammad Abdel-Majeed, Daniel Wong, and Murali Annavaram. 2013. Gating aware scheduling and power gating for GPGPUs. In *MICRO*.

[3] Homa Aghilinasab, Mohammad Sadrosadati, Mohammad Hossein Samavatian, and Hamid Sarbazi-Azad. 2016. Reducing power consumption of GPGPUs through instruction reordering. In *ISLPED*.

[4] Ali Fakhrzadehgan Mehdi Modarressi Amirhossein Mirhosseini, Mohammad Sadrosadati and Hamid Sarbazi-Azad. 2015. An energy-efficient virtual channel power-gating mechanism for on-chip networks. In *DATE*.

[5] Jayvant Anantpur and R Govindarajan. 2015. PRO: Progress aware GPU warp scheduling algorithm. In *IPDPS*.

[6] Amin Ansari, Asit Mishra, Jianping Xu, and Josep Torrellas. 2014. Tangle: Route-oriented dynamic voltage minimization for variation-afflicted, energy-efficient on-chip networks. In *HPCA*.

[7] Manish Arora, Srilatha Manne, Indrani Paul, Nuwan Jayasena, and Dean M Tullsen. 2015. Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU integrated systems. In *HPCA*.

[8] Rachata Ausavarungnirun. 2017. *Techniques for shared resource management in systems with throughput processors.* Ph.D. Dissertation. Carnegie Mellon University.

[9] Rachata Ausavarungnirun, Kevin Kai-Wei Chang, Lavanya Subramanian, Gabriel H. Loh, and Onur Mutlu. 2012. Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems. In *ISCA*.

[10] Rachata Ausavarungnirun, Saugata Ghose, Onur Kayiran, Gabriel H Loh, Chita R Das, Mahmut T Kandemir, and Onur Mutlu. 2015. Exploiting inter-warp heterogeneity to improve GPGPU performance. In *PACT*.

[11] Rachata Ausavarungnirun, Saugata Ghose, Onur Kayıran, Gabriel H Loh, Chita R Das, Mahmut T Kandemir, and Onur Mutlu. 2018. Holistic management of the GPGPU memory hierarchy to manage warp-level latency tolerance. *arXiv preprint arXiv:1804.11038* (2018).

[12] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J Rossbach, and Onur Mutlu. 2017. Mosaic: A GPU memory manager with application-transparent support for multiple page sizes. In *MICRO*.

[13] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J Rossbach, and Onur Mutlu. 2018. MASK: Redesigning the GPU memory hierarchy to support multi-application concurrency. In *ASPLOS*.

[14] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*.

[15] Juan M. Cebrin, Gines D. Guerrero, and Jose M. Garcia. 2012. Energy efficiency analysis of GPUs. In *IPDPSW*.

[16] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. 2016. Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization. In *SIGMETRICS*.

[17] Kevin K Chang, A Giray Yağlıkçı, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu. 2017. Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and mechanisms. In *SIGMETRICS*.

[18] 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 *IISWC*.

[19] Lizhong Chen and Timothy M Pinkston. 2012. NoRD: Node-router decoupling for effective power-gating of on-chip routers. In *MICRO*.

[20] Lizhong Chen, Lihang Zhao, Ruisheng Wang, and Timothy M Pinkston. 2014. MP3: Minimizing performance penalty for power-gating of clos network-on-chip. In *HPCA*.

[21] Design Compiler. 2000. Synopsys inc. (2000).

[22] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. 2011. Memory power management via dynamic voltage/frequency scaling. In *ICAC*.

[23] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F Wenisch, and Ricardo Bianchini. 2011. Memscale: active low-power modes for main memory. In *ASPLOS*.

[24] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. 2002. Drowsy caches: simple techniques for reducing leakage power. In *ISCA*.

[25] Denis Foley, Pankaj Bansal, Don Cherepacha, Robert Wasmuth, Aswin Gunasekar, Srinivasa Gutta, and Ajay Naini. 2011. A low-power integrated x86-64 and graphics processor for mobile computing devices. In *ISSCC*.

[26] Wilson WL Fung and Tor M Aamodt. 2011. Thread block compaction for efficient SIMT control flow. In *HPCA*.

[27] Wilson WL Fung, Ivan Sham, George Yuan, and Tor M Aamodt. 2007. Dynamic warp formation and scheduling for efficient GPU control flow. In *MICRO*.

[28] Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, William J. Dally, Erik Lindholm, and Kevin Skadron. 2011. Energy-efficient mechanisms for managing thread context in throughput processors. In *ISCA*.

[29] Syed Zohaib Gilani, Nam Sung Kim, and Michael J Schulte. 2013. Exploiting GPU peak-power and performance tradeoffs through reduced effective pipeline latency. In *MICRO*.

[30] Syed Zohaib Gilani, Nam Sung Kim, and Michael J Schulte. 2013. Power-efficient computing for compute-intensive GPGPU applications. In *HPCA*.

[31] Bhargava Gopireddy, Choungki Song, Josep Torrellas, Nam Sung kim, Aditya Agrawal, and Asit Mishra. 2016. ScalCore: Designing a core for voltage scalability. In *HPCA*.

[32] David Hodges, Horace Jackson, and Resve Saleh. 2004. *Analysis and design of digital integrated circuits in deep submicron technology*. McGraw-Hill.

[33] Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. In *ISCA*.

[34] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. 2004. Microarchitectural techniques for power gating of execution units. In *ISLPED*.

[35] Canturk Isci, Alper Buyuktosunoglu, and Margaret Martonosi. 2005. Long-term workload phases: Duration predictions and applications to DVFS. *IEEE micro* (2005).

[36] Hyeran Jeon and Murali Annavaram. 2012. Warped-DMR: Light-weight error detection for GPGPUs. In *MICRO*.

[37] Qing Jiao, Mian Lu, Huynh Phung Huynh, and Tulika Mitra. 2015. Improving GPGPU energy-efficiency through concurrent kernel execution and DVFS. In *CGO*.

[38] Naifeng Jing, Jianfei Wang, Fengfeng Fan, Wenkang Yu, Li Jiang, Chao Li, and Xiaoyao Liang. 2016. Cache-emulated register file: An integrated on-chip memory architecture for high performance GPGPUs. In *MICRO*.

[39] Adwait Jog, Onur Kayiran, Nachiappan Chidambaram Nachiappan, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, Ravishankar Iyer, and Chita R. Das. 2013. OWL: Cooperative thread array aware scheduling techniques for improving GPGPU performance. In *ASPLOS*.

[40] Adwait Jog, Onur Kayiran, Ashutosh Pattnaik, Mahmut T Kandemir, Onur Mutlu, Ravishankar Iyer, and Chita R Das. 2016. Exploiting core criticality for enhanced GPU performance. In *ACM SIGMETRICS*.

[41] Ali Jooya and Amirali Baniasadi. 2013. Using synchronization stalls in power-aware accelerators. In *DATE*.

[42] David Kadjo, Hyungjun Kim, Paul Gratz, Jiang Hu, and Raid Ayoub. 2013. Power gating with block migration in chip-multiprocessor last-level caches. In *ICCD*.

[43] Himanshu Kaul, Mark Anders, Steven Hsu, Amit Agarwal, Ram Krishnamurthy, and Shekhar Borkar. 2012. Near-threshold voltage (NTV) design: Opportunities and challenges. In *DAC*.

[44] Mehmet Kayaalp, Khaled N Khasawneh, Hodjat Asghari Esfeden, Jesse Elwell, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. 2017. RIC: Relaxed inclusion caches for mitigating LLC side-channel attacks. In *DAC*.

[45] Onur Kayıran, Adwait Jog, Mahmut Taylan Kandemir, and Chita Ranjan Das. 2013. Neither more nor less: Optimizing thread-level parallelism for GPGPUs. In *PACT*.

[46] Onur Kayıran, Adwait Jog, Ashutosh Pattnaik, Rachata Ausavarungnirun, Xulong Tang, Mahmut T. Kandemir, Gabriel H. Loh, Onur Mutlu, and Chita R. Das. 2016. μC-States: Fine-grained GPU datapath power management. In *PACT*.

[47] Onur Kayiran, Nachiappan Chidambaram Nachiappan, Adwait Jog, Rachata Ausavarungnirun, Mahmut T Kandemir, Gabriel H Loh, Onur Mutlu, and Chita R Das. 2014. Managing GPU concurrency in heterogeneous architectures. In *MICRO*.

[48] Michael Keating and Pierre Bricaud. 1998. *Reuse methodology manual, Keating and Bricaud*. Kluwer Academic Publishers.

[49] Ali Keshavarzi, Kaushik Roy, and Charles F Hawkins. 1997. Intrinsic leakage in low power deep submicron CMOS ICs. In *ITC*.

[50] Farzad Khorasani, Hodjat Asghari Esfeden, Nael Abu-Ghazaleh, and Vivek Sarkar. 2018. In-register parameter caching for dynamic neural nets with virtual persistent processor specialization. In *MICRO*.

[51] Farzad Khorasani, Hodjat Asghari Esfeden, Amin Farmahini-Farahani, Nuwan Jayasena, and Vivek Sarkar. 2018. RegMutex: Inter-Warp GPU register time-sharing. In *ISCA*.

[52] Farzad Khorasani, Rajiv Gupta, and Laxmi N Bhuyan. 2015. Efficient warp execution in presence of divergence with collaborative context collection. In *MICRO*.

[53] Farzad Khorasani, Bryan Rowe, Rajiv Gupta, and Laxmi N Bhuyan. 2016. Eliminating intra-warp load imbalance in irregular nested patterns via collaborative task engagement. (2016).

[54] Gwangsun Kim, John Kim, and Sungjoo Yoo. 2011. Flexibuffer: Reducing leakage power in on-chip network routers. In *DAC*.

[55] Nam Sung Kim, Krisztián Flautner, David Blaauw, and Trevor Mudge. 2002. Drowsy instruction caches: Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *MICRO*.

[56] John Kloosterman, Jonathan Beaumont, D Anoushe Jamshidi, Jonathan Bailey, Trevor Mudge, and Scott Mahlke. 2017. Regless: Just-in-time operand staging for GPUs. In *MICRO*.

[57] Jesper Knudsen. 2008. Nangate 45nm open cell library. *CDNLive, EMEA* (2008).

[58] Oshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. 2013. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *ICCD*.

[59] Shin-Ying Lee and Carole-Jean Wu. 2014. CAWS: Criticality-aware warp scheduling for GPGPU workloads. In *PACT*.

[60] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. In *ISCA*.

[61] Dong Li, Surendra Byna, and Srimat Chakradhar. 2011. Energy-aware workload consolidation on GPU. In *ICPPW*.

[62] Hai Lia, Swarup Bhunia, Yiran Chen, TN Vijaykumar, and Kaushik Roy. 2003. Deterministic clock gating for microprocessor power reduction. In *HPCA*.

[63] Zhenhong Liu, Syed Gilani, Murali Annavaram, and Nam Sung Kim. 2017. G-Scalar: Cost-effective generalized scalar execution architecture for power-efficient GPUs. In *HPCA*.

[64] Anita Lungu, Pradip Bose, Alper Buyuktosunoglu, and Daniel J Sorin. 2009. Dynamic power gating with quality guarantees. In *ISLPED*.

[65] Srilatha Manne, Artur Klauser, and Dirk Grunwald. 1998. Pipeline gating: Speculation control for energy reduction. In *ISCA*.

[66] Hiroki Matsutani, Michihiro Koibuchi, Daisuke Ikebuchi, Kimiyoshi Usami, Hiroshi Nakamura, and Hideharu Amano. 2010. Ultra fine-grained run-time power gating of on-chip routers for CMPs. In *NOCS*.

[67] Hiroki Matsutani, Michihiro Koibuchi, Daihan Wang, and Hideharu Amano. 2008. Adding slow-silent virtual channels for low-power on-chip networks. In *NOCS*.

[68] Amirhossein Mirhosseini, Mohammad Sadrosadati, Sara Aghamohammadi, Mehdi Modarressi, and Hamid Sarbazi-Azad. 2018. BARAN: Bimodal adaptive reconfigurable-allocator network-on-chip. *ACM TOPC* (2018).

[69] Amirhossein Mirhosseini, Mohammad Sadrosadati, Behnaz Soltani, Hamid Sarbazi-Azad, and Thomas F. Wenisch. 2017. BiNoCHS: Bimodal network-on-chip for CPU-GPU heterogeneous systems. In *NOCS*.

[70] Asit K Mishra, Reetuparna Das, Soumya Eachempati, Ravi Iyer, Narayanan Vijaykrishnan, and Chita R Das. 2009. A case for dynamic frequency tuning in on-chip networks. In *MICRO*.

[71] 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 *MICRO*.

[72] Negin Nematollahi, Mohammad Sadrosadati, Hajar Falahati, Marzieh Barkhordar, and Hamid Sarbazi-Azad. 2018. Neda: Supporting direct inter-core neighbor data exchange in GPUs. *IEEE CAL* (2018).

[73] NVIDIA. 2008. NVIDIA management library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml. (2008).

[74] NVIDIA. 2009. *Whitepaper: NVIDIA's next generation CUDA^{TM} compute architecture: Fermi^{TM}*. Technical Report. NVIDIA.

[75] NVIDIA. 2016. How to tune GPU performance using Radeon WattMan and Radeon Chill. https://support.amd.com/en-us/kb-articles/Pages/DH-020.aspx. (2016).

[76] NVIDIA. 2016. *White paper: NVIDIA Tesla P100*. Technical Report.

[77] NVIDIA. 2018. Dynamic Clocking. https://www.geforce.com/hardware/technology/gpu-boost/technology. (2018).

[78] NVIDIA. 2018. GTX480. https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/architecture. (2018).

[79] Xiang Pan and Radu Teodorescu. 2014. NVSleep: Using non-volatile memory to enable fast sleep/wakeup of idle cores. In *ICCD*.

[80] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. 2014. Integrated CPU-GPU power management for 3D mobile games. In *DAC*.

[81] Gennady Pekhimenko, Evgeny Bolotin, Mike O'Connor, Onur Mutlu, Todd C Mowry, and Stephen W Keckler. 2015. Toggle-aware compression for GPUs. In *IEEE CAL*.

[82] Gennady Pekhimenko, Evgeny Bolotin, Nandita Vijaykumar, Onur Mutlu, Todd C Mowry, and Stephen W Keckler. 2016. A case for toggle-aware compression for GPU systems. In *HPCA*.

[83] Abbas Rahimi, Luca Benini, and Rajesh K Gupta. 2016. CIRCA-GPUs: Increasing instruction reuse through inexact computing in GPGPUs. *IEEE Design and Test* (2016).

[84] Abbas Rahimi, Amirali Ghofrani, Kwang-Ting Cheng, Luca Benini, and Rajesh K. Gupta. 2015. Approximate associative memristive memory for energy-efficient GPUs. In *DATE*.

[85] Minsoo Rhu and Mattan Erez. 2013. Maximizing SIMD resource utilization in GPGPUs with SIMD lane permutation. In *ISCA*.

[86] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. 2000. Memory access scheduling. In *ISCA*.

[87] Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. 2003. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proc. IEEE* (2003).

[88] Mohammad Sadrosadati, Amirhossein Mirhosseini, Homa Aghilinasab, and Hamid Sarbazi-Azad. 2015. An efficient DVS scheme for on-chip networks using reconfigurable virtual channel allocators. In *ISLPED*.

[89] Mohammad Sadrosadati, Amirhossein Mirhosseini, Seyed Borna Ehsani, Hamid Sarbazi-Azad, Mario Drumond, Babak Falsafi, Rachata Ausavarungnirun, and Onur Mutlu. 2018. LTRF: Enabling high-capacity register files for GPUs via hardware/software cooperative register prefetching. In *ASPLOS*.

[90] Mohammad Sadrosadati, Amirhossein Mirhosseini, Shahin Roozkhosh, Hazhir Bakhishi, and Hamid Sarbazi-Azad. 2017. Effective cache bank placement for GPUs. In *DATE*.

[91] Mohammad Hossein Samavatian, Hamed Abbasitabar, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2014. An efficient STT-RAM last level cache architecture for GPUs. In *DAC*.

[92] Ahmad Samih, Ren Wang, Anil Krishna, Christian Maciocco, Charlie Tai, and Yan Solihin. 2013. Energy-efficient interconnect via router parking. In *HPCA*.

[93] Ankit Seething, Ganesh Dasika, Mehrzad Samadi, and Scott Mahlke. 2010. APOGEE: Adaptive prefetching on GPUs for energy efficiency. In *PACT*.

[94] Hynix Semiconductor. 2009. Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0. (2009).

[95] Ankit Sethia and Scott Mahlke. 2014. Equalizer: Dynamic tuning of GPU resources for efficient execution. In *MICRO*.

[96] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniek Liu, and Wen Mei Hwu. 2012. *Parboil: A revised benchmark suite for scientific and commercial throughput computing*. Technical Report. Center for Reliable and High-Performance Computing, UIUC.

[97] Aniruddha S Vaidya, Anahita Shayesteh, Dong Hyuk Woo, Roy Saharoy, and Mani Azimi. 2013. SIMD divergence optimization through intra-warp compaction. In *ISCA*.

[98] Rangharajan Venkatesan, Shankar Ganesh Ramasubramanian, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2014. STAG: Spintronic-tape architecture for GPGPU cache hierarchies. In *ISCA*.

[99] Nandita Vijaykumar, Eiman Ebrahimi, Kevin Hsieh, Phillip B Gibbons, and Onur Mutlu. 2018. The locality descriptor: A holistic cross-layer abstraction to express data locality in GPUs. ISCA.

[100] Nandita Vijaykumar, Kevin Hsieh, Gennady Pekhimenko, Samira Khan, Ashish Shrestha, Saugata Ghose, Adwait Jog, Phillip B Gibbons, and Onur Mutlu. 2016. Zorua: A holistic approach to resource virtualization in GPUs. In *MICRO*.

[101] Nandita Vijaykumar, Gennady Pekhimenko, Adwait Jog, Abhishek Bhowmick, Rachata Ausavarungnirun, Chita Das, Mahmut Kandemir, Todd C Mowry, and Onur Mutlu. 2015. A case for core-assisted bottleneck acceleration in GPUs: Enabling flexible data compression with assist warps. In *ISCA*.

[102] Po-Han Wang, Yen-Ming Chen, Chia-Lin Yang, and Yu-Jung Cheng. 2009. A predictive shutdown technique for GPU shader processors. *IEEE CAL* (2009).

[103] Po-Han Wang, Chia-Lin Yang, Yen-Ming Chen, and Yu-Jung Cheng. 2011. Power gating strategies on GPUs. *ACM TACO* (2011).

[104] Yu Wang, Soumyaroop Roy, and Nagarajan Ranganathan. 2012. Run-time power-gating in caches of GPUs for leakage energy savings. In *DATE*.

[105] Qiumin Xu and Murali Annavaram. 2014. PATS: Pattern aware scheduling and power gating for GPGPUs. In *PACT*.

[106] Jieming Yin, Pingqiang Zhou, Sachin S. Sapatnekar, and Antonia Zhai. 2014. Energy-efficient time-division multiplexed hybrid-switched NOC for heterogeneous multicore systems. In *IPDPS*.

[107] Wing-kei S Yu, Ruirui Huang, Sarah Q Xu, Sung-En Wang, Edwin Kan, and G Edward Suh. 2011. SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading. In *ISCA*.

[108] William K Zuravleff and Timothy Robinson. 1997. Controller for a synchronous DRAM that maximizes throughput by allowing memory requests and commands to be issued out of order. (May 13 1997). US Patent 5,630,096.