Predicting Critical Warps in Near-Threshold GPGPU Applications using a Dynamic Choke Point Analysis

Sourav Sanyal, Prabal Basu, Aatreyi Bal, Sanghamitra Roy, Koushik Chakraborty
USU BRIDGE LAB, Electrical and Computer Engineering, Utah State University
{sourav.sanyal, prabalb, aatreyi.bal}@aggiemail.usu.edu, {sanghamitra.roy, koushik.chakraborty}@usu.edu

ABSTRACT

General purpose graphics processing units (GP-GPU) can significantly improve the power consumption at the NTC operating region. However, process variation (PV) can drastically reduce its performance. In this paper, we examine choke points–a unique device-level characteristic of PV at NTC–that can exacerbate the warp criticality problem. We show that the modern warp schedulers cannot tackle the choke point induced critical warps in an NTC GPU. We propose *Warp Latency Booster*, a circuit-architectural solution to dynamically predict the critical warps and accelerate them in their respective execution units. Our best scheme achieves an average improvement of $\sim$32% and $\sim$41% in performance, and $\sim$21% and $\sim$19% in energy-efficiency, respectively, over two state-of-the-art warp schedulers.

1. INTRODUCTION

The growing adoption of general-purpose graphics processing units (GPGPU) over the last decade marks an important landmark in the era of parallel computing. However, the power envelope of GPGPUs has undergone a steady rise [15]. In order to limit power consumption, researchers have recently explored the benefits of Near-Threshold Computing (NTC) in the realm of GPGPUs [6]. Despite its impressive energy-efficiency, the NTC design paradigm is afflicted with a severe process variation (PV) induced delay variability problem. In this work, we demonstrate how choke points–a unique device-level characteristic at NTC–can redefine the critical warp(s). In the NTC domain, *choke points*–a small set of gates with PV induced delay variations–embody a unique challenge in designing micro-architectures [5, 7], substantially different from a large body of existing works on general PV [2, 8, 9, 20]. Performing a dynamic path sensitization analysis, we observe that an irregular formation of choke points in GPU SIMD lanes can aggravate the delays of the critical warp within a thread block. Moreover, owing to a heightened sensitivity to within-die PV at NTC [6], the delay profiles of the warps belonging to the same instruction, can exhibit a significant diversity, across different streaming multiprocessors. *Consequently, choke points at NTC often radically alter the architecturally induced critical warps, thus dwarfing the efficacy of architecturally tailored state-of-the-art warp schedulers* [13, 14, 19].

In order to reduce the performance imbalance among the parallel warps, we propose a novel adaptive technique, referred to as *Warp Latency Booster (WLB)*. The following are our specific contributions in this work:

- We explore the role of choke points in radically transforming the architecturally induced critical warps in GPGPUs (Section 2).
- We demonstrate that existing warp scheduling policies cannot effectively alleviate the delays of choke point induced critical warps (Section 5).
- We propose a low-overhead technique called *Warp Latency Booster* that identifies circuit-level delay variabilities and improves the performance of the parallel warps in GPGPUs (Section 3).

2. MOTIVATION

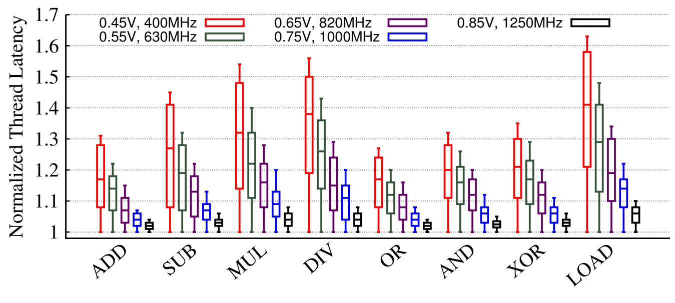
We explain the formation of critical warps (Section 2.1) and discuss the impact of choke points in further complicating the warp criticality problem at NTC (Section 2.2). Using a cross-layer experimental setup (Section 2.3), we present choke point induced performance imbalance in GPUs (Section 2.4). We conclude by establishing the need for a warp criticality prediction technique for NTC GPUs (Section 2.5).

2.1 What is Warp Criticality?

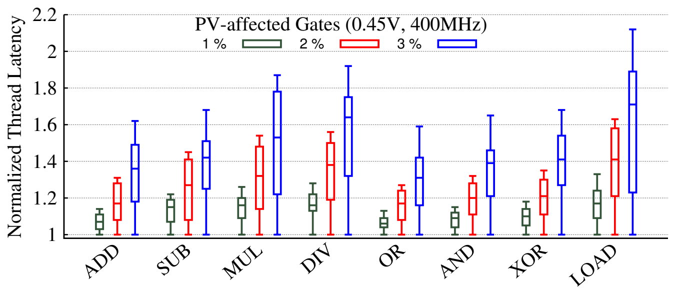
A large variation in the warp latencies within a thread block, can lead to a severe performance penalty. The slowest warp can keep other warps waiting in the GPU pipeline for a considerable time, even after their executions. Consequently, a new thread block cannot be swapped in, until the slowest warp in the current thread block finishes its execution. This phenomenon is known as the *warp criticality problem*, and the slowest warp is called the *critical warp* [14].

2.2 Impacts of Choke Points

We focus on the distribution of choke points–logic gates that alter non-critical paths into critical ones upon



(a) Relative performance slowdown at different operating VF points.



(b) Relative performance slowdown at different extents of PV.

Figure 1: *Figure 1(a) represents the variation in a warp's thread latencies at various operating VF points for different FUs. Figure 1(b) shows different thread latencies of the same warp at different fractions of PV-affected gates at (0.45V, 400MHz) across various FUs.*

sensitization, across fabricated design instances [5, 7]. Being a manifestation of the fabrication process, choke point induced delay variations cannot be estimated by pre-silicon techniques like static timing analysis [5]. A choke point in an FU can drastically increase the thread latency, which in turn, can degrade the corresponding warp execution time. As delay sensitivity to PV increases at lower operating voltages, NTC GPUs are more prone to choke point induced performance degradations compared to their super-threshold (STC) counterparts. Next, we briefly discuss our methodology to analyze the choke point induced warp criticality problem in NTC GPUs.

2.3 Methodology

We model a Southern Island AMD GPU [21] and run 15 GPGPU applications to obtain cycle-wise FU utilization metadata for several PTX instructions. To synthesize a GPU ALU [1], we use the 15-nm FinFET library from NanGate [17]. To model PV for FinFET technology at STC and NTC operating conditions, we use the VARIUS [20], VARIUS-NTV [9] and VARIUS-TC [10] models. Feeding our in-house Statistical Timing Analysis tool with the architectural metadata from Multi2Sim and synthesized ALU netlist, we obtain the sensitized delay characteristics of several FUs.

2.4 Results

Figure 1(a) illustrates the performance slowdown distributions at various operating voltage-frequency (VF) points for all the threads in a warp across different FUs.

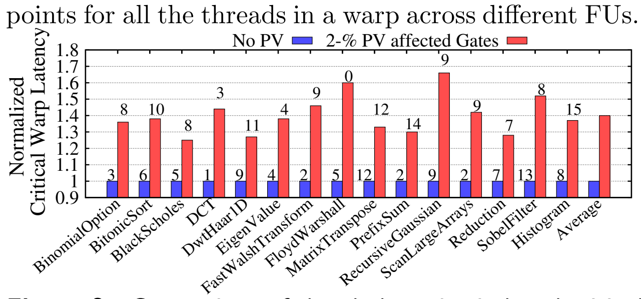


Figure 2: *Comparison of the choke point induced critical warp latencies (red bars) in 15 GPGPU applications at (0.45V, 400MHz), and considering 2% PV-affected gates.*

We notice that: (a) for a given FU, a lower operating VF point results in a greater intra-warp performance variation. This is due to a significantly higher delay sensitivity to PV at lower operating voltages. Hence, the warp criticality problem is specifically detrimental to the performance of the NTC GPUs. (b) At a given operating point, different FUs are differently affected by choke points. This indicates the dependence of choke point induced delay degradation on the sensitized circuit paths. Figure 1(b) displays the slowdown for different percentages of PV affected logic gates across various FUs at (0.45V, 400MHz). We observe that the slowdowns increase with the fraction of the PV affected gates for all the FUs. Figure 2 illustrates the normalized latencies of the choke point induced (red bars), and the corresponding architecturally induced (blue bars) critical warps for 15 GPGPU applications. We observe that (a) the latencies of the choke point induced critical warps are significantly greater than the corresponding architecturally induced critical warps. The presence of choke points at NTC exacerbates the critical warp latencies from 25% to 66% with an average increase of 40%. (b) The formation of choke points engenders new critical warps–different from the respective architecturally induced critical warps—in 11 out of 15 GPGPU applications. This fact is reflected by the warp IDs on top of each bar in Figure 2. As architecturally tailored warp schedulers [13, 14] are agnostic of circuit-level dynamic delay variabilities, they cannot efficiently tackle the choke point induced critical warps at NTC.

2.5 Insight to WLB

Our motivational study uncovers the multi-modal nature of the warp criticality conundrum. While near-threshold operation of a GPU severely affects the latencies of the critical warps, sensitization of different circuit topologies brings about an appreciable spatial performance heterogeneity. However, boosting the execution speed of a choke point induced critical warp can reduce the idle time within a thread block. *Hence, it is imperative to dynamically predict the timing behaviors of the warps, and accordingly, speed up the execution of the critical warp(s) in a thread block.*

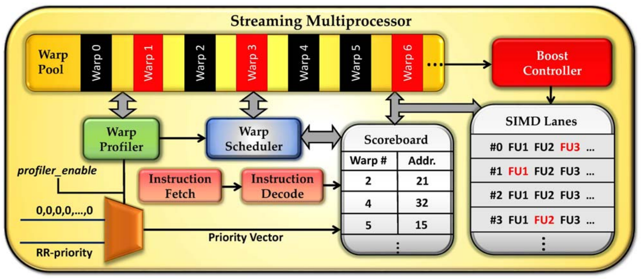


Figure 3: *A conceptual block diagram of WLB. The warp profiler predicts critical warps (Warp 1, Warp 3 and Warp 6) which are executed with boosted latencies (FU3 in lane 0, FU1 in lane 1, and FU2 in lane 3).*

3. WARP LATENCY BOOSTER

We present *Warp Latency Booster*—an adaptive technique that dynamically predicts and boosts the executions of the choke point induced critical warps (Section 3.1). We outline the design challenges (Section 3.2) and discuss the two phases of WLB (Sections 3.3 and 3.4).

3.1 System Overview

Figure 3 represents a conceptual block diagram of the WLB. The primary components of WLB are the *warp profiler* and the *boost controller*. We assume a baseline round-robin (RR) thread block scheduler to schedule thread blocks to available SMs [12]. We augment an RR warp scheduler to work in tandem with the proposed warp profiler. Given a barrier interval, the warp profiler ascertains critical warp(s) in the profiling phase (Section 3.3). The critical warps are subsequently accelerated by the boost controller in their respective FUs for the rest of their barrier executions (Section 3.4). We operate our baseline NTC GPU at a better-than-worst case VF point, and use an existing error detection and correction mechanism to tackle intermittent timing violations [11].

3.2 Design Challenges

The design of WLB is associated with several crucial challenges, as listed below.

- **Choke Point Awareness:** It is imperative to correctly identify the warps that are most affected by choke point induced delay variabilities in the SIMD lanes. As the critical warps are selectively sped up, a misidentification can potentially lead to an even greater performance heterogeneity within an SM.
- **Harmonious Execution:** The baseline RR warp scheduler offers an impressive resource utilization by *fairly* scheduling the ready warps in the pool. It is critical to ensure that the warp profiler works in harmony with the RR warp scheduler, while analyzing the timing profiles of the warps.
- **Pipeline Synchronization:** It is essential to maintain synchronization among the GPU pipe stages when some warps are executed with boosted speeds. For example, any warp having an architectural dependency on a boosted warp needs to be issued earlier in order to prevent a queueing delay.

3.3 Profiling Phase

3.3.1 Principle of Warp Profiling

In view of choke point awareness, one of the most important questions in the design of WLB is: *how can we guarantee that a warp, identified as a critical one in the profiling phase, continues to be critical for the rest of the barrier interval?* The answer to this question lies in the inherent nature of the warp execution.

Say, n CUDA threads, comprising a warp, are scheduled to be executed on m SIMD lanes in an SM, where $n = k * m$. As each warp corresponds to only one instruction, each of the n CUDA threads utilizes the same FU. Assuming the FU latency to be 1 cycle, the total execution time of the warp is k cycles in the absence of pipeline stalls. As the warp utilizes the same FU in each of the k cycles, the choke point induced delays of the sensitized paths are very likely to be same across all the k cycles Thus, the warp's timing behavior in the first p cycles ($p < k$) of its execution, is a good proxy for its overall timing characteristic.

3.3.2 Warp Profiler

Algorithm 1 shows the working principle of the profiler. To compare the timing profiles of the warps, the warp scheduler needs to issue all the warps simultaneously. This is a deviation from the default RR behavior, where only one warp is scheduled to execute in one cycle. We make several strides to ensure a harmonious execution of the warp profiler and the warp scheduler.

Before the beginning of the profiling phase, the warp profiler communicates with the RR warp scheduler to assign equal priorities to all the warps in a thread block. Consequently, the warps get scheduled simultaneously. Once scheduled, the priorities of the warps are set to a high value to obviate rescheduling of warps during the profiling phase. For the purpose of profiling, each warp in the thread block needs to execute for at least one iteration. Hence, we set the duration of the profiling phase to the maximum of the execution times of all the instructions pertaining to the scheduled warps (line 3 in Algorithm 1). To account for PV, the instruction execution times are ascertained and stored for each SM.

We adopt a low-complexity heuristic (line 6 to 13 in Algorithm 1) to ascertain the critical warp(s). The progress of a warp is tracked using the existing dynamic load execution (DLE) counter in modern GPUs [18]. The DLE count gets incremented only if the warp has finished executing a load. The critical warps, being tardy, have smaller DLE counts compared to the faster warps in the same thread block. When a thread block executes a load, the warp profiler logs the DLE counts for all the warps in that thread block. If the DLE count

Algorithm 1 Warp Profiling

```
1:  $w_i \in W \quad W \leftarrow Available\ Warp\ Pool$
2:  $w_i\_priority \leftarrow 0 \quad \forall w_i \in W$
3:  $t_{profile} = t_{clock} * Max(CPI_j) \quad \forall j \in I_{w_i} \quad \forall w_i \in W$
4:  $Schedule\_Warps()$
5:  $w_i\_priority \leftarrow \infty \quad \forall w_i \in W$
6:  $C_{w_\mu} \leftarrow 0$
7:  while $t_{profile} > 0$ do
8:      $C_{w_\mu} \leftarrow C_{w_\mu} + C_{w_i} \forall C_{w_i} \leftarrow Warp's Execution Count$
9:  end while
10: $C_{w_\mu} \leftarrow C_{w_\mu}/n \quad n \leftarrow Total\ No.\ of\ Warps$
11: if $C_{w_i} < C_{w_{\mu - r*\sigma}}$ then
12:     $CriticalWarp[\,] \leftarrow w_i \quad \forall w_i \in W$
13: end if
14: $w_i\_priority \leftarrow RR - Priority \quad \forall w_i \in W$
```

of a warp is found to be r ($r > 0$) standard deviations less than the mean DLE count of all the warps, that warp is inferred to be critical. The value of r can be found empirically. Note that this mechanism can identify both choke point induced, as well as, architecturally induced critical warps. At the end of the profiling phase, all the warps are updated with the default RR priorities. As we do not employ any additional delay sensing circuit, the performance and hardware overheads of the profiler is marginal.

3.4 Boosting Phase

The top few critical warps, obtained from the profiling phase, are executed with higher execution speeds in the boosting phase (i.e., the remaining barrier execution time). A low-overhead boost controller communicates with the profiler and manages the execution speeds of the warps. The speed enhancement is realized by dynamically boosting the latencies of the corresponding FUs in the SIMD lanes.

3.4.1 Circuit-Architectural Considerations for Boost

- In pursuit of sustaining an error-free and optimal pipeline activity, we alter the dispatch logic. By monitoring the progress of boosted warps [18], subsequent dependent warps are accordingly dispatched. This, in turn, alleviates scoreboarding and results in a better thread block performance.
- To maintain low implementation overheads, we consider two boost levels, realized using additional voltage rails [6]. The runtime transition among different voltages is achieved using a setup similar to the one proposed in [16].
- In order to avoid complex synchronization circuitry, we consider only integral boost levels in this work, specifically, $2\times$ and $3\times$ the nominal FU latencies. Out of the n critical warps, the boost controller applies $3\times$ boost to the top *n/2* warps, and $2\times$ boost, to the rest *n/2* warps.

3.4.2 Improving Energy-Efficiency

- The latency improvement of a warp plateaus at very high boost levels (viz., $4\times$ and more) when the FUs consume significantly high energy. Consequently, we stipulate the maximum boost level at $3\times$ the corresponding nominal execution speed of an FU.
- If a boosted warp is stalled due to any architectural hazard, it can incur a tremendous power overhead. To avoid this situation, we keep track of a warp's execution time [18], and if the execution time crosses a threshold, we undo the boost, i.e., reduce the execution speed to the corresponding nominal value.
- When the performance heterogeneity among the parallel warps is insignificant, boosting the critical warp's execution can cause a non-critical warp to become critical. This phenomenon is known as *criticality inversion* [14] which reduces the power-performance benefit of WLB. In Section 5.2.1, we discuss how one form of WLB can reduce criticality inversion.

4. METHODOLOGY

4.1 Device Layer

We estimate the NTC energy consumptions by performing HSPICE simulations on the basic logic gates (viz., NAND, NOR and Inverter). We use the 31-stage FO4 inverter-chain as a representative of various combinational logics in a GPU [6]. The simulation parameters are obtained from the 16-nm Predictive Technology Model [22]. The delays of the basic gates are used in the circuit layer (Section 4.2) to ascertain the latencies of the warps sensitizing different FUs in the SIMD lanes.

4.2 Circuit Layer

We implement WLB by augmenting an open-source reference GPU RTL [1]. The synthesized GPU netlist, along with an encoded PTX instruction vector obtained from the architecture layer (Section 4.3), are used as inputs to our in-house statistical timing analysis (STA) tool. The warp timing information is exported to the architecture layer to obtain the application performance.

4.3 Architecture Layer

We instrument the codebase of the Multi2Sim architectural simulator [21] to implement the WLB. The specific architectural parameters for our evaluation are outlined in Table 1. We use 15 GPGPU applications from AMD's APP SDK suite [3]. To obtain the critical warps of an application, we collect the executed instruction metadata and use it as input vectors to the STA tool (Section 4.2). The boost phase of the WLB is implemented by dynamically altering the FU latencies. in Multi2Sim for some warps based on the warp criticality information obtained from the STA tool.

Parameters	Configurations
No. of SMs	128
Thread Block Size	16
FU Latency	4-64 / 2-32 / 1-21 cycles
Local Memory	32 KB, latency: 2 cycles
L2 Cache	8×768 KB, latency: 20 ns
Global Memory	B/W: 264 GB/s, latency: ~300 ns

Table 1: *NTC GPU architectural configurations.*

5. EXPERIMENTAL RESULTS

We present a comparative analysis of various schemes (Section 5.1) in terms of performance (Section 5.2) and energy-efficiency (Section 5.3). Additionally, we discuss the implementation overheads of WLB (Section 5.4).

5.1 Comparative Schemes

- **Round-Robin (RR)**: The scheduling is done by issuing a single warp in every clock cycle, after assigning equal priority to each warp.
- **Greedy-Then-Oldest (GTO)**: This scheme executes a single warp till it stalls and then selects the oldest ready warp for the execution [19].
- **Criticality Aware Warp Scheduler (CAWA)**: It consists of a a *warp scheduler* that allocates more SIMD resources to the predicted critical warps, and a *cache prioritizer* that favors the critical warps while accessing the cache [13].
- **WLB - Streaming Multiprocessor (WLB-SM)**: In this variant of WLB, implemented at the granularity of an SM, the warps across an entire SM are subjected to profiling in a given barrier execution.
- **WLB - Thread Block (WLB-TB)**: This is another variant of WLB, implemented at the granularity of a thread block. Unlike WLB-SM, the profiling for identifying choke point induced critical warps, is done for every thread block.

5.2 Performance Results

Figure 4 depicts the performances of 15 GPGPU applications under different comparative schemes (Section 5.1). The results are normalized to the respective performances under the baseline RR warp scheduler. On average, WLB-TB and WLB-SM outperform RR (by ~53% and ~44%), GTO (by ~50% and ~41%), and

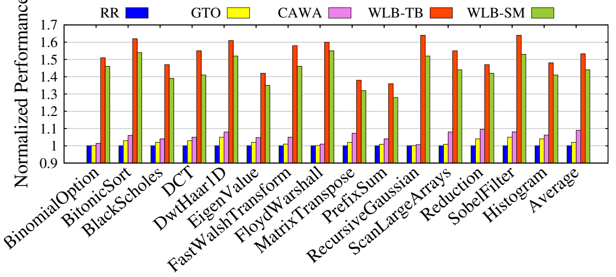


Figure 4: *Performance comparison (Higher is better).*

WLB Variants	Criticality Inversion	Performance Improvements
WLB-SM	~4.8%	~44%
WLB-TB	~2.3%	~53%

Table 2: *Criticality inversion and the performance improvements of the WLB variants over RR.*

CAWA (by ~41% and ~32%), respectively. The warp criticality predictor in CAWA can identify only the architecturally induced critical warps [13]. As choke points often radically alter the architecturally induced critical warps (Section 2.4), CAWA frequently allocates sub-optimal SIMD resources for the choke point induced critical warps, leading to minor performance improvements over RR. On the other hand, as GTO executes a single warp till it stalls and then selects the oldest one, GTO often stalls repeatedly due to the choke point induced critical warps, incurring a severe performance penalty. For all the applications, WLB-TB offers the best performance. Next, we elaborate on the performance difference between WLB-TB and WLB-SM.

5.2.1 Criticality Inversion

Table 2 enumerates the frequency of the occurrences of the criticality inversion (Section 3.4.2) in the WLB variants, along with their respective average performance gains, compared to the baseline RR warp scheduler. A lower value of criticality inversion is associated with a higher performance improvement. WLB-SM is agnostic of the specific performance heterogeneity of the warps inside each thread block. Instead, it collectively profiles all the warps in an SM, to ascertain the critical warp(s) for that SM. Consequently, the inferred boost level(s) can potentially overspeed some warps in their respective thread blocks, resulting in several criticality inversions. On the other hand, WLB-TB, tailored to reduce the performance imbalance at the granularity of a thread block, leads to fewer criticality inversions, and hence a better performance than WLB-SM.

5.3 Energy-efficiency Results

Figure 5 shows the energy consumptions for all the comparative schemes, normalized to baseline. WLB-TB and WLB-SM consume ~40% and ~26% more energy, respectively, compared to the baseline. The major proportion of these energies comes from the power overheads of the boosting phase. As WLB-TB boosts more warps in a barrier interval, on an average, with respect to WLB-SM, the former consumes significantly more energy than the latter, for all the applications. The additional hardware for critical warp prediction in CAWA, and additional control logic in GTO, make them more energy-hungry than RR. Figure 6 presents the energy efficiency benefits for all the comparative schemes in terms of energy-delay product (EDP), normalized to baseline. Our proposed schemes (WLB-SM

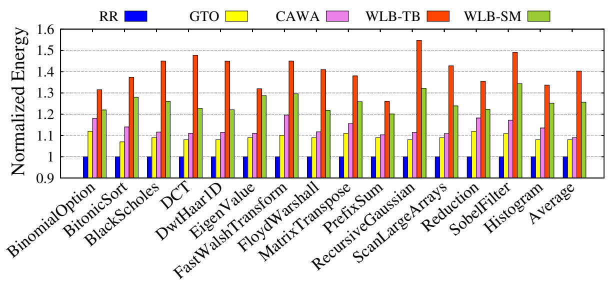


Figure 5: *Energy comparison (Lower is better).*

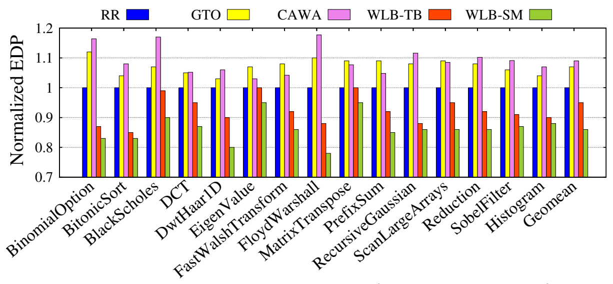


Figure 6: *EDP comparison (Lower is better).*

and WLB-TB) are more energy-efficient than RR, GTO and CAWA. This is because the large performance improvements of our schemes grossly amortize their relatively high energy footprints. Energy efficiency of GTO is inferior to the WLB variants, as GTO achieves only a minor performance improvement over RR, at the expense of a significantly high energy consumption. CAWA is the least energy-efficient scheme across all applications, on average, as its relatively high energy footprint eclipses its meager performance benefits. Among all the schemes, WLB-SM is the most energy-efficient surpassing RR, GTO and CAWA by ∼15%, ∼19%, and ∼21%, respectively, on an average.

5.4 Implementation Overheads

The hardware cost of WLB comes from the warp profiler, the boost controller, and the additional voltage rails. Due to identical implementations of the warp profiler and the voltage rails, and only a minor difference in the boost control logic, both WLB-TB and WLB-SM incur almost identical overheads. With our methodology, the area, wire-length and power overheads for WLB are 0.71%, 2.4%, and 4.1%, respectively, over an NTC GPU with RR warp scheduling.

6. RELATED WORK

Researchers have investigated the prospects of NTC systems for about a decade now. Dreslinski et al. explored the energy-efficiency benefits, and limitations of NTC systems [2]. Aguilera et al. have delved into recovering performance loss in PV affected GPUs [4]. However, PV affected NTC GPUs have garnered minimal attention from the researchers. Basu et al. have proposed a run-time technique to improve GPU performance at NTC [6]. Bal et al. have investigated choke points in a CPU and proposed a dynamic approach to tackle them [5]. Lee et al. proposed a coordinated warp acceleration technique by managing the compute and memory resources of the GPGPU workloads [13]. *However, to the best of our knowledge, our work is the first to thoroughly analyze the impact of choke points on the warp criticality problem in NTC GPUs.*

7. CONCLUSION

In this paper, we demonstrate that existing warp schedulers cannot effectively tackle choke point induced critical warps in NTC GPUs. To improve the GPU performance, we propose a novel design paradigm that identifies critical warps in GPGPU applications, and boosts their execution speeds at NTC. Our cross-layer simulation results demonstrate that our proposed scheme outperforms existing warp schedulers in terms of performance and energy-efficiency.

8. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants (CAREER-1253024, CCF-1318826, CNS-1421022, and CNS-1421068). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the National Science Foundation.

9. REFERENCES

[1] *MIAOW GPU.* http://miaowgpu.org.
[2] *Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits* (2010).
[3] AMD Accelerated Parallel Processing (APP) Software Development Kit , 2016.
[4] Aguilera, P. and others Process variation-aware workload partitioning algorithms for GPUs supporting spatial-multitasking. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014* (2014), pp. 1–6.
[5] Bal, A. and others Revamping Timing Error Resilience To Tackle Choke Points at NTC systems. In *Proc. of DATE* (2017), pp. 1020–1025.
[6] Basu, P. and others SwiftGPU: Fostering Energy Efficiency in a Near-Threshold GPU Through a Tactical Performance Boost. In *Proc. of DAC* (2016).
[7] De, V. Fine-Grain Power Management in Manycore Processor and System-on-Chip (SoC) Designs. In *Proc. of ICCAD* (2015), pp. 159–164.
[8] Karpuzcu, U. R. and others Coping with Parametric Variation at Near-Threshold Voltages. *IEEE Micro 33*, 4 (2013), 6–14.
[9] Karpuzcu, U. R. and others VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *DSN* (2012), pp. 1–11.
[10] Khatamifard, S. K. and others VARIUS-TC: A modular architecture-level model of parametric variation for thin-channel switches. In *ICCD* (2016), pp. 654–661.
[11] Krimer, E. and others Lane decoupling for improving the timing-error resiliency of wide-SIMD architectures. In *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA* (2012), pp. 237–248.
[12] Lee, M. and others Improving GPGPU resource utilization through alternative thread block scheduling. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014* (2014), pp. 260–271.
[13] Lee, S. and others CAWA: coordinated warp scheduling and cache prioritization for critical warp acceleration of GPGPU workloads. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015* (2015), pp. 515–527.
[14] Lee, S., and Wu, C. CAWS: criticality-aware warp scheduling for GPGPU workloads. In *PACT* (2014), pp. 175–186.
[15] Lucas, J. and others How a single chip causes massive power bills GPUSimPow: A GPGPU power simulator. In *ISPASS* (April 2013).
[16] Mohanty, S. P., and Pradhan, D. K. ULS: A dual-V_{th}/high-kappa nano-CMOS universal level shifter for system-level power management. *J Emerg. Tech. Comp. Sys. 6*, 2 (2010).
[17] NanGate. http://www.nangate.com/?page_id=2328.
[18] Oh, Y. and others WASP: Selective Data Prefetching with Monitoring Runtime Warp Progress on GPUs. *IEEE Transactions on Computers* (2018).
[19] Rogers, T. G. and others Cache-Conscious Wavefront Scheduling. In *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Vancouver, BC, Canada, December 1-5, 2012* (2012), pp. 72–83.
[20] Sarangi, S. and others VARIUS:A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Tran. on Semicond. Manufac. 21* (2008), 3 –13.
[21] Ubal, R. and others Multi2Sim: A Simulation Framework for CPU-GPU Computing . In *PACT* (Sep. 2012).
[22] Zhao, W., and Cao, Y. New Generation of Predictive Technology Model for sub-45nm Early Design Exploration. *T. Electron Devices 53*, 11 (2006), 2816 –2823.