HVSM: Hardware-Variability Aware Streaming Processors' Management Policy in GPUs

Jingweijia Tan
College of Computer Science and Technology
Jilin University
Changchun, China, 130012
Email: jtan@jlu.edu.cn

Kaige Yan
College of Communication Engineering
Jilin University
Changchun, China, 130012
Email: yankaige@jlu.edu.cn

***Abstract*—GPUs are widely used in general-purpose high performance computing field due to their highly parallel architecture. In recent years, a new era with nanometer scale integrated circuit manufacture process has come, as a consequence, GPUs' computation capability gets even stronger. However, as process technology scales down, hardware variability, e.g., process variations (PVs) and negative bias temperature instability (NBTI), has a higher impact on the chip quality. The parallelism of GPU desires high consistency of hardware units on chip, otherwise, the worst unit will inevitably become the bottleneck. So the hardware variability becomes a pressing concern to further improve GPUs' performance and lifetime, not only in integrated circuit fabrication, but more in GPU architecture design.**

Streaming Processors (SPs) are the key units in GPUs, which perform most of parallel computing operations. Therefore, in this work, we focus on mitigating the impact of hardware variability in GPU SPs. We first model and analyze SPs' performance variations under hardware variability. Then, we observe that both PV and NBTI have large impact on SP's performance. We further observe unbalanced SP utilization, e.g., some SPs are idle when others are active, during program execution. Leveraging both observations, we propose a Hardware Variability-aware SPs' Management policy (HVSM), which dynamically prioritizes the fast SPs, regroups SPs in a two-level granularity and dispatches computation in appropriate SPs. Our experimental results show HVSM effectively reduces the impact of hardware variability, which can translate to 28% performance improvement or 14.4% lifetime extension for a GPU chip.

I. Introduction

Today, few words can attract more attention than "big data". Large volume of data is generated every day. How to efficiently process them becomes a great challenge. With thousands of cores on a chip, modern GPUs provide strong computational capability, and they are widely used platform for general-purpose high performance computing applications, e.g., deep learning, cloud computing, image processing, and so on. In the past, the improvement of multi-core and many-core processors, including GPUs, is largely driven by integrated circuit manufacture process scaling. However, as it scales down to nanometer level, the impact of hardware variability starts to become a major factor affecting chip quality [1][2][3][4][5].

This work is supported in part by Natural Science Foundation of Science and Technology Department of Jilin Province, and National Key Research and Development Program of China for High Performance Computing under Grant 2016YFB0201503.

Hardware variability includes static and dynamic effects. The static effect is primarily process variations (PVs) and the dynamic effect is mostly negative bias temperature instability (NBTI). PV is the variation of transistors' parameters (e.g., length, voltage, oxide thickness) on a chip introduced during fabrication phase, as it is very difficult to precisely control the nanometer scale manufacture process. PV can cause significant performance variation of transistors and some low-quality transistors have to run at lower frequency or higher voltage. NBTI is an aging effect mainly caused by negative gate voltages, which manifests a decrease in drain current and an increase in transistor's threshold voltage. It can significantly deteriorate the performance of PMOS transistors. If some PMOS transistors have negative gate voltage applied more frequently than others, they will suffer more deterioration because of NBTI. Overall, we use the terminology *condition of transistors*, to describe how fast they can run or how much they are affected by the hardware variability. Badly (well) conditioned transistors are low (high) quality ones due to PV or deteriorated (less deteriorated) ones due to NBTI. Obviously, the badly conditioned transistors have to run slower than the good ones. However, to avoid timing errors, all transistors on a chip must finish their switch operations on time, therefore, the overall chip performance is limited by the worst conditioned transistors. Recent research shows real operation frequency of GPUs can only achieve half of the theoretical value without any mitigation technique [6].

The impact of hardware variability is especially detrimental to GPUs, since GPUs feature high parallelism. A GPU program can contain thousands of parallel threads and streaming processors (SPs) are the key hardware components of GPUs for executing these parallel threads. Usually, there are thousands of SPs in a GPU chip. If just one SP is in bad condition and it runs slow, other SPs will be forced to run at the same low speed and the overall performance will be bound by the slowest one. Although integrated circuit fabrication improvement can partially alleviate hardware variability, to guarantee the consistency of thousands of SPs in GPU is very difficult, and almost impractical.

In this work, we explore mitigating the impact of hardware variability of GPU SPs at architecture design level. We first model and analyze SPs' performance variations under the

978-3-9819263-0-9/DATE18/©2018 EDAA

impact of PV and NBTI. We observe that the *speed variation ratio*, defined as the execution time of the slowest SP over the fastest in one GPU chip, is 3.4 with PVs only. Further, considering NBTI, the speed variation ratio can be up to 7.9 after 3 years. It suggests that both PVs and NBTI have great impact on SPs' performance. We then observe unbalanced SP utilization during program execution, e.g., some SPs can be idle for 23% of the execution time while other SPs are active, which gives us the opportunity to re-distribute the workloads of SPs based on their conditions to alleviate the impact of hardware variability. If they are mis-matched, a badly conditioned SP can take heavy workload and becomes even worse.

Based on our observations, we propose a Hardware-Variability aware SPs' Management policy (HVSM) to combat the impact of hardware variability for GPU SPs. HVSM dynamically prioritizes the SPs in good condition, regroups the SPs into two levels and dispatch the computation to appropriate SPs, during program execution. Our experimental results show HVSM can effectively reduce SPs' speed variation ratio, the metric to quantify the impact of hardware variability effects as defined before, to 2.6 on average from 7.9 in worst case. It can improve the performance by 28% or extend the GPU lifetime by 14.4%.

The contributions of this work are summarized as follows:

- We model and analyze the impact of hardware variability in GPU SPs, by considering the aggregated effect of process variations and negative bias temperature instability.
- We analyze SPs' utilization by considering GPU characteristics at program and architecture levels.
- We propose a Hardware-Variability aware SPs' Management policy (HVSM) to reduce the impact of hardware variability. It can match the utilizations and conditions of SPs, by dynamically prioritizing the well conditioned SPs during program execution.

The rest of the paper is organized as follows. Section II introduces the background of GPU architecture and hardware variability. We present the proposed HVSM technique in Section III. Section IV is the experimental setup and we discuss experiment results in Section V. Section VI is related work. Finally, we conclude this work in Section VII.

II. Background

A. GPU Architecture and Streaming Processors

Figure 1 shows the architecture of a modern GPU. A GPU is typically composed of several in-order streaming multiprocessors (SMs)[1] [7]. SMs connect to L2 cache and off-chip memory via an interconnection network. Usually, a GPU program is composed of thousands of parallel threads. Threads are distributed to SMs at the granularity of blocks, each of which may contain hundreds of threads. When executing a GPU program, SM will group a certain number of threads

[1]All GPUs share similar architecture, although terminologies may be different. In this work, we use NVIDIA terminologies as illustration, but our technique is applicable to all GPUs.

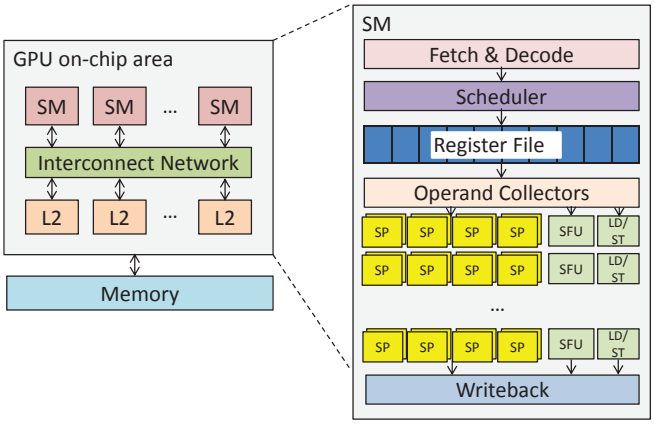

Fig. 1. GPU architecture (SM: streaming multiprocessor; SP: streaming processor; SFU: special function unit; LD/ST: load store unit).

(e.g., 32) to form a warp and execute them together. Threads in a warp execute the Same Instruction, but with Multiple Data (SIMD). Several warps are executed in an interleaved fashion in an SM. Every cycle, one ready warp is issued to an SM pipeline. Then, this pipeline schedules the instruction for execution after fetching and decoding it from the instruction cache. Operands for the ready warp can be collected from register file and temporarily stored in the operand collector. When all operands are available, the warp is selected for execution. In the execution stage, the warp will be issued to streaming processors (SPs), special function units (SFUs) or load/store units (LD/ST), based on the instruction type.

SPs are the units where most of the normal instructions are executed and we observe that SPs may execute up to 100% of the computing operations in some programs. There are multiple groups of SPs in an SM. Each SP group, also referred as SPG, usually contains 16 SPs. They first perform the SIMD execution of half a warp and then another half [8], [9]. All the SPs in one group execute the same instruction, but each SP only access the operands of its corresponding thread. Execution of branch instructions in a warp will cause warp divergence, with some threads taking the branch while others fall through. In this case, two branch directions will have to execute in serial fashion, i.e., threads with branch taken will execute first then others with branch not taken execute next, or vice versa. When some SPs are executing one branch direction, other SPs will be idle in the SIMD pipeline. Besides branch divergence, SPs can also become idle when executing some long latency operations. All SPs remain idle until at least one warp is ready.

B. Process Variations

Process variations (PVs) are the variation of transistor parameters on a chip, which are combination of two main effects during manufacture process: random effect (e.g., random dopant fluctuations) and systematic effect caused by lithographic lens aberrations. Both random and systematic effects have high contribution to PV. There are two key

parameters, that determine the transistor switch latency, and are highly influenced by process variation: effective channel length (L_{eff}) and threshold voltage (V_{th}) [10]. They can be calculated as

$$\Delta V_{th} = \Delta V_{th,rand} + \Delta V_{th,sys}, \quad (1)$$
$$\Delta L_{eff} = \Delta L_{eff,rand} + \Delta L_{eff,sys}, \quad (2)$$

where $\Delta V_{th,rand}$, $\Delta V_{th,sys}$ are random and systematic parts of V_{th} respectively, and $\Delta L_{eff,rand}$, $\Delta L_{eff,sys}$ are random and systematic parts of L_{eff} respectively. The latency T required to switch the transistor can be modeled as

$$T \propto \frac{V_{dd} \times L_{eff}}{\mu \times (V_{dd} - V_{th})^{\alpha}}, \quad (3)$$

where α is a constant factor of 1.3 and μ is a function of temperature indicating the mobility of carriers [10].

C. Negative Bias Temperature Instability (NBTI)

Negative Bias Temperature Instability (NBTI) is an aging effect, which causes significant deterioration especially for PMOS, when negative voltage is applied. Applying negative voltage breaks the silicon-hydrogen (Si-H) bonds at the silicon/oxide interface and then generates interface traps. When electrons flow from source to the drain, they are captured by those traps. As a consequence, V_{th} will increase and this is called the stress process. When negative voltage is removed, the interface trap is partially healed and V_{th} is slightly recovered, which is called recovery process. Considering both processes, the threshold voltage increase can be modeled as

$$\Delta V_{th} = (K\sqrt{t_s} + \sqrt[2n]{\Delta V_{th-t0}})^{2n} \times (1 - \sqrt{\frac{\eta t_r}{t_s + t_r}}), \quad (4)$$

where t_s is stress time, t_r is recovery time, K is a factor determined by the electrical field, temperature, and supply voltage, n is a constant factor of 1/6 [1], V_{th-t0} is the initial V_{th} variation due to PVs and η is a constant factor of 0.35 [11]. A transistor with increased V_{th} will have longer switch latency according to Equation 3, and we call it deteriorated. The higher ΔV_{th}, the more deteriorated it is. In general, if a transistor is used more often, there is higher likelihood that negative voltage is applied and it becomes more deteriorated.

III. Hardware-Variability Aware SPs' Management Policy

A. Observation: Performance Variations of SPs

The performance of an SP is determined by its transistors. After a GPU chip is fabricated, different SPs may contain transistors with different levels of conditions due to process variation effect. Furthermore, SPs are generally not used uniformly when executing a GPU program: first, only one of many streaming processors groups, i.e., SPGs, will be selected when a warp is scheduled into the pipeline in an SM; second, when branch divergence occurs, not all SPs within the same group are active at the same time; last, SPs can execute different kinds of operations and they have various execution times. Therefore, after GPUs are used for a while, transistors in SPs can have different level of deterioration by NBTI, based on the characteristics of executed programs. Overall, depending on the condition of its transistors, SPs will have different execution latencies and we call it the condition of SPs. Obviously, a well conditioned SP can run faster than a bad one.

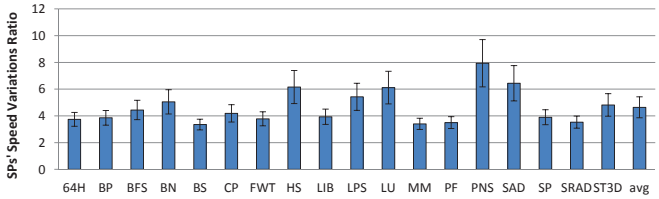

Fig. 2. Average and standard deviation of SPs' speed variation ratio under hardware variability.

In this subsection, we investigate the performance variation of SPs under both hardware variability effects. We first model the impact of process variation using the timing error model of logics in VARIUS [10] under 28nm process technology (detailed experimental methodology can be found in Section IV). We then integrate it with NBTI effect as in Equation 4, to further evaluate GPU SPs' performance after three years' execution. To quantify SPs' performance variation, we also define a metric called *speed variation ratio*, as the execution time of worst conditioned SP over the best one in the same GPU chip. Small ratio indicates low hardware variability impact and guardband requirement.

Figure 2 shows the average normalized speed variation ratio for 100 chips, with +/-3 standard deviations (3σ) across all chips which comprises 99.7% of all values. As it shows, SPs' speed variation ratio is 4.6 on average for all benchmarks, which suggests the conditions of SPs can vary significantly within the same GPU chip due to process variation and NBTI effects. Meanwhile, we can also see the speed variation ratio for different benchmarks ranges from around 3.7 to 7.9, which indicates this ratio is also affected by benchmark-specific characteristics.

In the current GPU architecture design, the frequency guardband of all SPs are limited by the worst conditioned one due to process variation, then an aging guardband is applied on top of it assuming the worst case NBTI impact on all SPs. Well conditioned SPs have the capability to run faster, but may be forced to run at lower frequency due to these two guardbands, which causes significant performance loss.

B. Observation: Unbalanced Utilization of SPs

In this subsection, we further investigate SPs' utilization by characterizing their cycle distribution during program execution. The baseline case is non-stall cycle, defined as the number of cycles with at least one SP active in an SM. Figure 3 shows percentage of diverge cycles, defined as the number of cycles when some SPs are active while others are idle in an SPG; and idle cycles, defined as the number of cycles when all SPs are idle in an SPG, comparing against the baseline. As

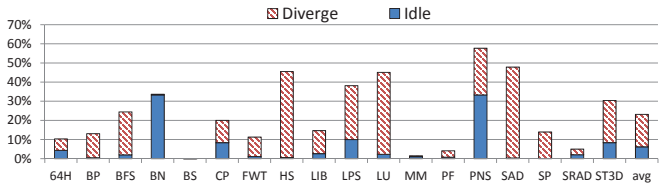

Fig. 3. Percentage of SPs' idle and diverge cycles comparing against the baseline non-stall cycle.

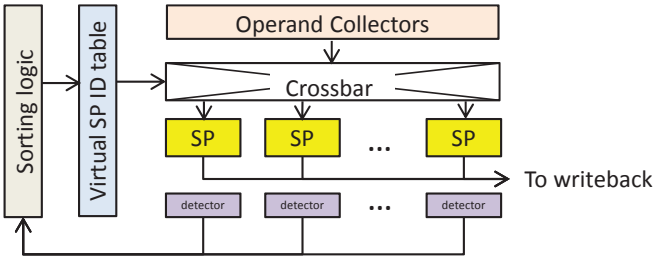

Fig. 4. The architecture of HVSM.

it shows, on average idle and diverge cycles account for 6% and 17% of non-stall cycles respectively. It provides us the opportunity of re-distributing the workloads of SPs to match with their conditions, i.e., SPs in good condition will take more workload while bad ones will take less, in order to alleviate the impact of hardware variability. For benchmarks like *PNS* and *SAD*, idle and diverge cycles take around 50% of the non-stall cycle, which means the workloads of some SPs are only half of others. In these cases, if the workload of SPs are properly re-distributed, the impact of hardware variability can be greatly reduced. However, for benchmark *BS*, *MM* and *PF*, and *SRAD*, SP idle cycles are rare ($<$10%), so there is little room to re-distribute the workload since all SPs are heavily loaded all the time.

C. Matching SPs' Conditions and Utilizations

Leveraging both observations, we propose a Hardware-Variability aware SPs' Management policy (HVSM) to match the SPs' conditions and utilizations, in order to reduce the speed variation ratio for a GPU chip. As mentioned in Section III-A, the unbalanced utilization of SPs is due to branch divergence, long latency operations, and so on. However, to precisely predict branch direction and long latency operations for each SP, can be difficult and expensive. Therefore, we adopt a more heuristic and efficient idea, i.e., prioritizing the SPs in good conditions, which makes sure that well conditioned SPs will be stressed more than the bad ones. To do this, we need to sort all SPs based on their conditions. Then, we design a two-level prioritizing scheme for SM, since an SM is the minimal granularity of warp scheduling and SIMD execution.

The architecture of HVSM is shown in Figure 4. Each SP is supplied with a detector to detect its condition. It will save the initial condition of SP and also model the dynamic deterioration of NBTI as in Equation 4. Note that these detectors are not in the critical path and do not affect SPs' normal execution, since normal execution does not require access to these detectors. The initial condition of each SP is determined right after the chip is fabricated. By gradually increasing the supply voltage and measuring corresponding execution latency of SP, we can easily infer L_{eff} and V_{th} as in Equation 3, which will be saved in the detector as the initial condition for this SP. Each detector contains two 64-bit timers to record the stress and recovery time respectively, at cycle granularity. The maximal duration of time these timers can record is much longer than the life span of most GPU chips, therefore, the timers can not be saturated. A sorting logic is used to sort SPs according to their conditions. We also introduce a virtual SP ID table to store the virtual to physical mapping of SP IDs based on sorted condition information. After sorting, physical SP IDs are written to the virtual SP ID table in descending order based on their conditions. The physical SP in the best condition becomes the first virtual SP, the second best conditioned physical SP becomes the second virtual SP, and so on. A crossbar is added in between operand collectors and SPs, which will re-direct the virtual SP to corresponding physical one based on the information in the virtual SP ID table. Note that unlike register file, re-directing the execution in SPs does not require any data migration, which can be much more expensive than our crossbar implementation.

When a warp is issued to the execution stage and the operation type is for SPs, HVSM performs the appropriate SP assignment based on a two-level prioritizing policy. First, we form SPGs based on SPs' conditions, e.g., if the SPG size is 16, the best conditioned 16 SPs will be grouped together, then next 16 SPs with slightly worse conditions are grouped, and so on so forth. When selecting an SPG, we always choose the available SPG with best condition. Second, in the selected SPG, SPs with best conditions are always chosen first to execute the active threads in cases like branch divergence.

We use HSPICE to model the implementation of the added structures. The sorting logic can be implemented as [12]. The condition calculation, sorting operation and update of virtual SP ID table is triggered when a GPU program is launched. Usually, before program execution, data needs to be copied to GPU at kernel launch time, and the latency overhead of HVSM can be hidden by the data movement. In cases when some GPU program runs very long and HVSM needs to perform condition calculation, sorting, and table update during program execution, the HVSM latency overhead is still negligible, since these operations take only 0.071ns together each time. For a GPU running at 1GHz, the cycle time is 1ns. Thus, the sorting can be finished within the same cycle of SP operation and the SP execution time is not increased. The introduced power overhead is negligible due to the small structure and infrequent operations of HVSM. The area overhead HVSM is $0.014mm^2$ per SM, they are 0.08% of a 28nm GPU chip.

Overall, HVSM can efficiently improve the speed variation ratio in a GPU chip, and this improvement can be translate to lifetime extension or performance improvement.

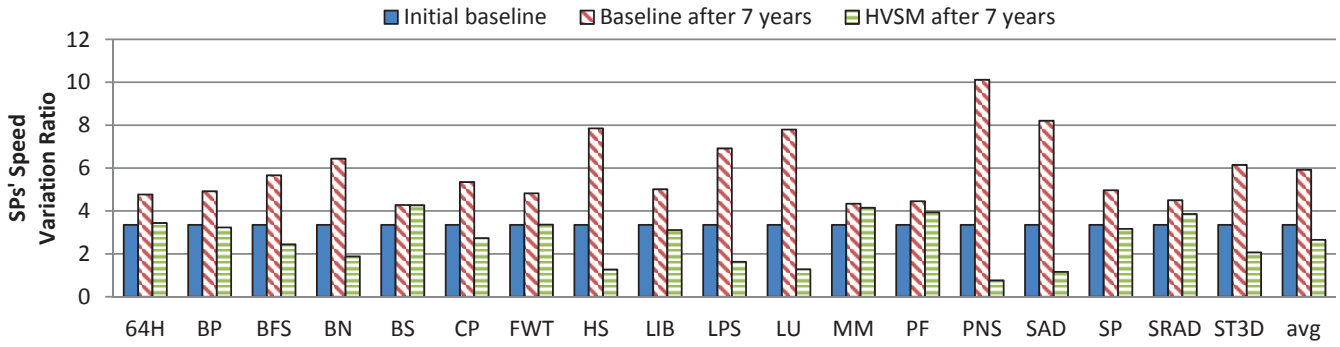

Fig. 5. SPs' speed variation ratio comparison: the initial baseline, baseline after seven years of execution and HVSM after seven years of execution.

TABLE I
CONFIGURATIONS OF THE BASELINE GPU.

Core (SM) frequency	700 MHz
Technology node	28nm
Number of SMs	15
Warp size	32
Number of Warp Schedulers	2
SPGs per SM	2
SIMT pipeline width	32
Warp scheduling policy	Greedy-Then-Oldest (GTO)
Register file size	128KB/SM
Shared memory size	48KB/SM
L1 data cache size	16KB/SM (4-way/128B)
L2 cache size	128KB/memory channel (8-way/128B)
Memory channels	6

IV. EXPERIMENTAL SETUP

We use a cycle-accurate simulator GPGPU-Sim (v3.1.0) [13] for our evaluation. The configuration of baseline GPU architecture is listed in Table I. We use the logic timing error model in VARIUS [10] to model SPs' performance under process variation effect. Then, we integrated it with the NBTI model as in Equation 4, to evaluate the combined effects of both process variation and NBTI. Our results are averaged with one hundred GPU chips. The benchmarks we use are a large set from NVIDIA CUDA SDK [14], Rodinia [15], and Parboil [16] benchmark suites, which have distinct characteristics and are able to represent real world GPU applications.

V. EVALUATION

A. Reduction of Speed Variation Ratio

We first evaluate SPs' speed variation ratio in Figure 5. The baseline is the architecture without HVSM. The initial baseline is right after the GPU chip is fabricated, which is only affected by process variation since it is not used yet. Later, when GPUs are used for a while, NBTI starts to affect the speed variation ratio. We choose seven years since most GPUs would be replaced beyond that. As the figure shows, SPs' speed variation ratio is 3.4 for initial baseline. After seven years of program execution, the ratio becomes 5.9 on average for baseline. Especially, the ratio for *PNS*, *SAD*, *HS* and *LU* are as high as 10.1, 8.2, 7.8, and 7.8 respectively, because their highly unbalanced SP utilization. Taking *PNS* as an example, we observe that the workloads of some SPs are only half of others. With HVSM, SPs' speed variation ratio is reduced to only 2.6 on average across all benchmarks, which is even lower than the initial baseline by 24%. This indicates our technique can effectively re-distribute the workloads based on the condition of SPs, especially for the benchmarks with high unbalanced SP utilization, like *PNS*, *SAD*. For benchmark *BS*, *MM* and *PF*, and *SRAD*, the improvement is small comparing with the baseline. The reason is that SPs for these benchmarks are almost fully loaded all the time, so it makes little difference no matter how to re-distribute the workloads.

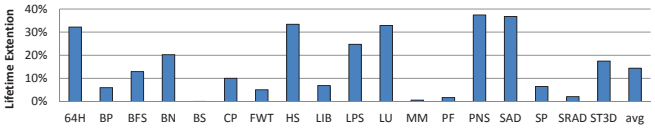

Fig. 6. Normalized lifetime extension with HVSM.

B. Lifetime Extension

We then analyze the lifetime extension with HVSM. We first define critical condition[2] as the worst condition of all SPs in a GPU chip with baseline architecture after seven years program execution. If any SP in a GPU chip reaches it, we consider the whole chip as in critical condition. The lifetime of a GPU chip can be defined as the time elapsed before it reaches the critical condition. Figure 6 shows the normalized lifetime extension is 14.4% with HVSM, which is about one year, comparing with the baseline. In general, benchmarks with higher reduction of speed variation ratio will have greater lifetime extension. As shown in the figure, GPU's lifetime can extend by 37% on average, which equals to 2.6 years, for benchmark *PNS* and *SAD*, and the improvement is small for benchmark *BS*, *MM* and *PF*, and *SRAD*.

C. Performance Improvement

Further, we evaluate the performance improvement with HVSM by increasing the frequency, under the same lifetime constraint, i.e., seven years. Since the hardware variability can be alleviated by HVSM, the guardband requirement will

[2]Critical condition is just a term, which doesn't mean GPUs cannot be used after reaching it.

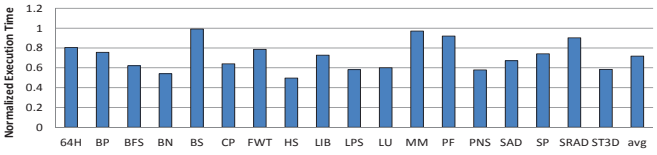

Fig. 7. Normalized execution time with HVSM.

be less strict, therefore, the frequency can be increased to improve the performance. Figure 7 shows the normalized GPU execution time with HVSM comparing with the baseline. On average, HVSM reduces the execution time by 28%. For benchmark *BS*, *MM* and *PF*, and *SRAD*, the performance improvement is small, with around 1%-10% execution time reduction. For benchmark *PNS* and *SAD*, the execution time is decreased by around 50%. Notes that the reduction of speed variation ratio can be flexibly translated to either lifetime, or performance, or both with each partially improved, depending on the needs.

VI. Related Work

Recently, many work have been proposed to mitigate the impact of hardware variability in GPUs. Several techniques only consider mitigating a single source of hardware variability [17][18][5][2][3][1]. For example, Aguilera et al. [19] proposed to run SMs at different frequencies in the presence of process variations and designed a task allocation algorithm to optimize GPU performance. Leng et al. [3] analyzed the safe-limit of supply voltage reduction under voltage noise. There are also works considering the combined impact of different sources of variations [20][6][21]. For instance, Thomas et al. [20] proposed a dynamic voltage reduction technique to mitigate the combined effect of process variation and voltage noise in GPUs. Tan et al [6] proposed to mitigate the impact of process variation and aging in GPU register file for performance improvement. These techniques either mitigated hardware variability at coarse granularity (i.e., GPU SM level) or focused on the storage structure such as register file. In contrast, ours is to mitigate hardware variability impact on the execution units, which is a key component for the basic computation of GPUs. Thus, our technique is orthogonal and novel comparing to all the above techniques.

VII. Conclusion

In this work, we introduce a microarchitecture-level technique to mitigate the combined effects of PV and NBTI in streaming processors of GPUs. We first model and analyze SPs' speed variation ratio under PV and NBTI, and find both effects contribute greatly to it. Further, we observe SPs' utilizations are unbalanced. We then propose a Hardware-Variability Aware SPs' Management policy (HVSM) that leverages the opportunity to match the utilizations and conditions of SPs. Our results show that HVSM can extend GPUs' lifetime by 14.4% or improve the performance by 28%.

References

[1] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, and P. Gupta, "Argo: Aging-aware gpgpu register file allocation," in *International Conference on Hardware/software Co-design and System Synthesis (CODES+ISSS)*, 2013, pp. 1–9.

[2] J. Leng, Y. Zu, and V. Reddi, "Gpu voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in gpu architectures," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 161–173.

[3] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe limits on voltage reduction efficiency in gpus: a direct measurement approach," in *Proceedings of the IEEE International Symposium On Microarchitecture (MICRO)*, 2015, pp. 294–307.

[4] S. Seo, R. Dreslinski, M. Woh, Y. Park, C. Charkrabari, S. Mahlke, D. Blaauw, and T. Mudge, "Process variation in near-threshold wide simd architectures," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2012, pp. 980–987.

[5] J. Tan and X. Fu, "Mitigating the susceptibility of gpgpus register file to process variations," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015, pp. 969–978.

[6] J. Tan, M. Chen, Y. Yi, and X. Fu, "Mitigating the impact of hardware variability for gpgpus register file," in *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 11, 2016, pp. 3283–3297.

[7] NVIDIA, "CUDA C Programming Guide," http://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[8] "NVIDIA's Next Generation CUDA Computer Architecture: Fermi," http://www.nvidia.com/content/pdf/fermi-white-papers/nvidia-fermi-compute-architecture-whitepaper.pdf.

[9] "NVIDIA's Next Generation CUDA Computer Architecture: Kepler," http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[10] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," in *IEEE Transactions on Semiconductor Manufacturing (TSM)*, vol. 21, no. 1, 2008, pp. 3–13.

[11] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 129–140.

[12] G. M. Blair, "Low cost sorting circuit for vlsi," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, no. 6, pp. 515–516, 1996.

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

[14] "Nvidia cuda sdk," https://developer.nvidia.com/cuda-downloads.

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

[16] "Parboil benchmark suite," https://github.com/abduld/Parboil.

[17] J. Lee, P. P. Ajgaonkar, and N. S. Kim, "Analyzing throughput of gpgpus exploiting within-die core-to-core frequency variation," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2011, pp. 237–246.

[18] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 237–248.

[19] P. Aguilera, J. Lee, A. Farmahini-Farahani, K. Morrow, M. Schulte, and N. S. Kim, "Process variation-aware workload partitioning algorithms for gpus supporting spatial-multitasking," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2014, pp. 1–6.

[20] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu, "Core tunneling: Variation-aware voltage noise mitigation in gpus," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 151–162.

[21] A. Rahimi, L. Benini, and R. K. Gupta, "Hierarchically focused guardbanding:an adaptive approach to mitigate pvt variations and aging," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2013, pp. 1695–1700.