Mitigating the Impact of Hardware Variability for GPGPUs Register File

Jingweijia Tan, *Student Member, IEEE*, Mingsong Chen, Yang Yi, and Xin Fu, *Member, IEEE*

Abstract—As technology keeps scaling down, hardware variability, such as process variations (PV) and negative bias temperature instability (NBTI), emerges as a growing challenge in the modern GPGPUs (general-purpose computing on graphics processing units). PV induces significant delay variations statically, while NBTI dynamically slows down the GPGPUs. Each computing core (i.e., streaming multiprocessor) in GPGPUs supports thousands of simultaneously active threads, and requires a large register file. Such a sizable register file is very sensitive to the hardware variability, and becomes one of the major units in determining the core frequency. In this study, we propose a set of techniques that mitigate both the PV and NBTI impacts on GPGPUs register file. In order to mitigate the susceptibility to PV, we first develop a novel mechanism that classifies registers into fast and slow categories in the highly-banked register architecture to maximize the frequency improvement. We then leverage the unique features in GPGPU applications to effectively tolerate the extra access delay to the slow registers. Moreover, we propose to dynamically balance the utilization across registers to further tolerate the NBTI degradation. Our experimental results show that our proposed techniques optimize GPGPUs performance by 22 percent on average under both PV and NBTI effects.

Index Terms—Hardware variability, process variations (PV), negative bias temperature instability (NBTI), aging, GPGPUs, register file

1 Introduction

Nowadays, GPUs possess strong computing power by executing thousands of threads in parallel, and have been widely adopted for the general-purpose computing, known as GPGPUs. As the process technology scales down in recent years, hardware variability, including both process variations (PV) and negative bias temperature instability (NBTI), becomes severer and is a growing threat to modern GPGPUs.

Process variation is the divergence of device parameters from their nominal values, which is caused by the challenging manufacture process at very small feature technologies. PV induces delay variations among critical paths and causes timing errors. To ensure that processors run as expected, the maximum clock frequency (FMAX) has to be limited by the worst critical path delay, leading to substantial performance loss. For example, the chip frequency degrades as much as 22 percent in 45 nm process technology due to PV [3]. And the frequency degradation becomes more significant with the continuous shrinking in feature size. The PV impact is exacerbated in modern GPGPUs which contain tremendous amount of parallel critical paths to deliver high computing throughput. Thus, the possibility that one path fails to meet the timing speculations increases substantially, which forces a severe decrease on FMAX in GPGPUs compared to that in CPUs [11], [12].

While PV causes delay variations statically, negative bias temperature instability gradually slows down the processor at run time. NBTI increases the threshold voltage in PMOS transistor when logic "0" is applied to the gate, this is also known as that the PMOS transistor is at the stress stage. The longer the PMOS is under the stress, the higher the threshold voltage becomes. As a result, the switching speed of the circuit decreases, leading to potential timing violations. The processor has to run at a lower frequency, known as the guardband, to absorb the NBTI impact. Previous work shows NBTI induces 25 percent performance degradation in 3 years under 45 nm process technology [37]. In GPGPUs, the high performance execution significantly stresses the critical paths which could suffer even more serious NBTI degradations [37], [38], [39].

The static variations due to the PV impact and the dynamic variations due to the NBTI impact can be aggregated, which substantially decrease the FMAX in GPGPUs. This motivates us to explore techniques to efficiently mitigate the hardware variability in GPGPUs, therefore, ensuring their high computational throughputs and performance.

GPGPUs support a great number of parallel threads and implement a zero overhead context switch among threads to hide the long latency operations. This requires an extremely large register file (RF) to keep the states and contents of all active threads. For instance, the register file size is 2 MB in NVIDIA Fermi [16] and 6 MB in AMD Cayman [17]. In recent GPUs product generations, the register file size is continuously increasing to afford a greater number of threads executing simultaneously in the streaming multiprocessor (SM) [16], [18]. Such a large register file includes numerous parallel critical paths, and is quite sensitive to

- *J. Tan and X. Fu are with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77004. E-mail: jtan12@uh.edu, xfu8@central.uh.edu.*
- *M. Chen is with the Software Engineering Institute at the East China Normal University, Shanghai, China. E-mail: mschen@sei.ecnu.edu.cn.*
- *Y. Yi is with the Department of Electrical Engineering and Computer Science at the University of Kansas, Lawrence, KS 66045. E-mail: yyi@ku.edu.*

Manuscript received 22 Sept. 2015; revised 8 Feb. 2016; accepted 14 Feb. 2016. Date of publication 18 Feb. 2016; date of current version 12 Oct. 2016. Recommended for acceptance by Z. Lan. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2016.2531668

process variations. Therefore, RF becomes one of the major units that affect the frequency and performance [33]. Meanwhile, the extensive use of RF to hold the state of each thread makes it greatly stressed and suffering heavy NBTI degradations [39], which further degrades the performance. As the first step to boost the GPUs performance under the impact of hardware variability, we mainly focus on tolerating the PV and NBTI impacts on the variability hotspot, i.e., register file, in this study.

The unique, i.e., highly-banked, register architecture design in GPGPUs provides a promising direction to efficiently tolerate the hardware variability. However, generic PV and NBTI mitigation techniques, such as adaptive body biasing (ABB) [19] and gate sizing [20], fail to exploit this unique feature and could cause considerable power and area overhead when directly applied to GPGPUs register file. In this paper, we propose to characterize and further leverage this highly-banked architecture feature to effectively mitigate the susceptibility of GPGPUs register file to variations. Note that we assume other PV- and NBTI-sensitive structures, such as streaming processors, are handled by conventional variability tolerance mechanisms (e.g., ABB).

In order to mitigate the susceptibility to PV, we first develop a novel mechanism that classifies registers into fast and slow categories in the highly-banked register architecture to maximize the frequency improvement. We then leverage the unique features in GPGPU applications to effectively tolerate the extra access delay to the slow registers. Moreover, we propose to dynamically balance the utilization across registers to further tolerate the NBTI degradation. The contributions of this work are as follows:

- We observe that PV exhibits much stronger systematic effects in the vertical direction than that in the horizontal direction within each RF bank. We then propose a coarse-grain register classification mechanism by vertically dividing each RF bank into sub-banks, and applying the variable-latency technique at the sub-bank level (VL-SB). VL-SB is able to attain the same frequency improvement as the fine-grain classification at the register level.
- We further propose RF bank re-organization (RF-BRO) to virtually combine sub-banks with the same speed type (i.e., fast, and slow) at the chip test time. Thus, the same-named registers in a RF bank entry share the uniform access delay, and the newly formed RF banks can be classified into fast and slow categories.
- In order to mitigate the IPC (instructions per cycle) loss caused by the slow RF banks access under VL-SB+RF-BRO, we propose to grant warps that heavily use fast RF banks a higher issue priority (in GPGPUs, threads are executed in warps). It forces fast RF banks to serve more threads and minimizes the use of slow RF banks, and also appropriately enlarges the progress difference among warps to effectively hide stalls caused by the long-latency operations.
- We then propose to virtually build multiple hybrid RF banks: each is composed of both fast and slow sub-banks. And only their fast sub-bank portions are enabled to hold registers for active threads in partially active warps. Thus their unused slow sub-bank portions have no impact on the RF access delay.
- We observe that NBTI causes different latency increment to different RF banks due to the uneven utilization across all banks. We propose to dynamically rename the RF banks required by each block at its allocation time (in GPGPUs, threads are allocated to SM at the granularity of blocks) to the virtual RF banks explored in our RF-BRO technique. This mechanism considers the uneven bank utilizations in the NBTI tolerance and shrinks the required guardband under the aggregated PV and NBTI effects.
- Finally, to well capture the aggregated PV and NBTI effects at run time, we further propose to re-organize the virtual RF banks at the kernel level.

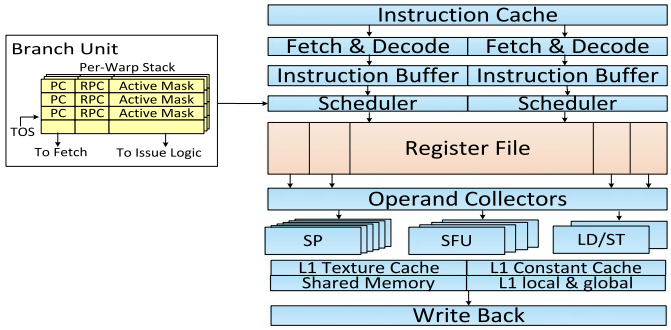


Fig. 1. An overview of streaming multiprocessor microarchitecture (SP: streaming processor; SFU: special functional units; LD/ST: load store units).

By combining all the explored techniques together, we achieve 22 percent performance improvement compared to the baseline case without any optimization under the impact of hardware variability (i.e., PV and NBTI).

2 Background

2.1 GPGPUs Architecture and Its Register File

The key component of a typical GPU is the in-order streaming multiprocessor [1]. Fig. 1 illustrates the SM microarchitecture [27]. In this paper, we study the NVIDIA CUDA programming model. In CUDA, the GPU executes highly-parallel kernel functions. The kernel is composed of a grid of light-weighted threads; a grid is divided into a set of blocks; each block is composed of hundreds of threads. Threads are distributed to SMs at the granularity of blocks.

Threads in the SM execute on the SPMD model. A number of individual threads (i.e., 32 threads in Nvidia GPUs) from the same block are grouped together, called warp. In the pipeline, threads within a warp execute the same instruction but with different data values. Each SM interleaves multiple warps on a cycle-by-cycle basis. At every cycle, an instruction warp that is ready for execution is selected and issued by a scheduler, and all threads belonging to that warp access the register file simultaneously. The execution of a branch instruction in the warp may cause warp divergence when some threads jump while others fall through at the branch. Threads in a diverged warp have to execute in serial fashion which causes multiple lanes to be idle in the SPMD pipeline. The load/store instruction may cause the off-chip memory access that can last hundreds of

cycles, and a long latency memory transaction from one thread would stall all threads within a warp.

In Nvidia PTX standard, an instruction can read up to 4 registers and write 1 register. Therefore, register file in the SM is heavily-banked (e.g., 16 or 32 banks) instead of multi-ported to provide high bandwidth, and multiple register operands required by one instruction can be read from different banks concurrently [1], [23], [24], [25], [26], [28], [29]. Each RF bank is equipped with dual ports to support 1 read and 1 write per cycle. Each entry in the bank is 128-byte wide to hold 32 same-named registers [23], [24]. During the register access, the RF bank ID is obtained based on the warp ID and register ID, and the port attached to that RF bank is activated to serve the access request.

Ideally, the register access for an instruction warp finishes in one cycle [23]. This is not the case when multiple register access requests map to the same bank and cause a bank conflict. In that case, requests have to be served sequentially, which extends the register access time to multiple cycles and hurts the performance. In order to reduce the possibility of bank conflicts, registers in a warp are distributed across the RF banks. Since multiple source operands for an instruction warp may not be read at the same cycle due to the bank conflicts, operand collectors (shown in Fig. 1) are applied to buffer the operands. One instruction warp will be allocated one operand collector once issued by the scheduler. When all required operands are ready in its assigned operand collector, the instruction warp proceeds into the execution stage and releases its operand collector resources.

2.2 Process Variations

Process variations are a combination of random effects (e.g., due to random dopant fluctuations) and systematic effects (e.g., due to lithographic lens aberrations) that occur during transistor manufacturing. Random variations refer to random fluctuations in parameters from die to die and device to device. Systematic variations refer to layout-dependent variations which cause nearby devices to share similar parameters. Among the design parameters, effective channel length (L_{eff}) and threshold voltage (V_{th}) are two key parameters subject to large variations [13]. The high V_{th} and L_{eff} variations cause high variations in transistor switching speed.

2.3 Negative Bias Temperature Instability

Negative Bias Temperature Instability is caused by the generation of interface trap in the silicon/oxide interface of PMOS transistors. When a negative voltage ($V_{gs} = -V_{dd}$) is applied to the gate of the PMOS transistor, the silicon-hydrogen (Si-H) bonds at the silicon/oxide interface are broken and generate the interface traps. The interface traps capture electrons flowing from the source to the drain. As a result, V_{th} increases and the transistor becomes slower. This process is called *stress*. When the transistor's delay exceeds the timing specifications, the timing error occurs. The increase in V_{th} can be modeled as [39]:

$$\Delta V_{th-stress} = \left(K_v\sqrt{t_{stress}} + \sqrt[2n]{\Delta V_{th-t0}}\right)^{2n}, \quad (1)$$

where t_{stress} is the PMOS stress time, K_v is determined by the electrical field, temperature, and supply voltage, n is the time exponent parameter which is 1/6 in this case, ΔV_{th-t0} is the initial V_{th} variation due to process variations.

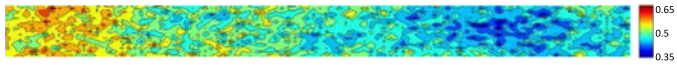


Fig. 2. V_{th} variation map of GPGPUs register file.

NBTI degradation can also be partially recovered by removing the stress voltage, which helps to heal the interface traps and partially recovers V_{th}. This process is called *recovery*. The overall increase of V_{th} considering both stress and recovery phases can be described as [40]:

$$\Delta V_{th} = \Delta V_{th-stress}\left(1 - \sqrt{\frac{\eta t_{rec}}{t_{stress} + t_{rec}}}\right), \quad (2)$$

where t_{rec} is the recovery time and η is equal to 0.35.

3 Mitigating the PV Impact on GPGPUs Register File

3.1 Modeling the PV Impact on GPGPUs Register File

In this study, we leverage the SRAM timing error model in VARIUS [13], and modify it to model the PV effects on GPGPUs register file. We focus on the 32 nm process technology that is generally used in the state-of-the-art GPUs [26]. We set the WID correlation distance coefficient ϕ as 0.5, and assume V_{th}'s σ/μ = 12%, L_{eff}'s σ/μ = 6% [9], [13], the random and systematic components have equal variances for both V_{th} and L_{eff} [4], [9], [13]. Fig. 2 shows an example of V_{th} variation map for GPGPUs register file. This figure indicates V_{th} has large variation across the whole register file. Note that we also perform the sensitivity analysis by varying the ratio between the random and systematic components when evaluating our proposed techniques in Section 6.4. We generate 100 chips for statistical analysis, and present the averaged result.

Note that each SM in GPGPUs exhibits different FMAX under PV. We model SM-to-SM variations as the ratio of frequencies of the fastest and the slowest SM in a GPGPUs chip, and model the within-SM variations as the ratio of frequencies of the fastest and slowest critical path. Based on our experimental results, within-SM variations are 1.7 which is larger than SM-to-SM variations that are around 1.3. This is because each SM has numerous parallel critical paths; while there are only tens of SMs. SM-to-SM variations have been smoothed out as FMAX of each SM is determined by the slowest critical path in it. Moreover, there have been techniques letting each SM in GPGPUs run at their own FMAX for PV mitigation [11]. We thus perform the variability analysis within the SM. As mentioned in Section 1, register file is one of the major structures that limit the SM frequency, and we assume other PV-sensitive structures are handled by conventional PV tolerance mechanisms. Thus, the SM frequency is determined by the register file frequency which is the reciprocal of the slowest register access time. Fig. 3a demonstrates the register file frequency distribution over 100 chips. There are 15 SMs in the Fermi architecture, and we use the averaged register file frequency across SMs to represent the frequency for one chip. Therefore, Fig. 3a mainly shows the impact of within-SM

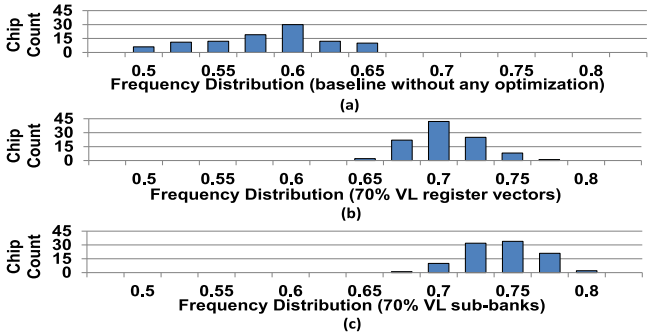


Fig. 3. Register file frequency distribution of 100 chips for (a) baseline, (b) 70 percent VL-RV, and (c) 70 percent VL-SB under process variations. The frequency of each chip is normalized to the chip without PV.

variations on frequency. As it shows, the mean frequency degradation is 40 percent compared to GPUs without PV. This is because numerous critical paths in GPGPUs register file lead to the large within-SM variation which significantly decreases the frequency.

3.2 Frequency Optimization for GPGPUs Register File under Process Variations

3.2.1 Variable-Latency Sub-Banks (VL-SB) in GPGPUs Register File

1) Extending VL-RF to GPGPUs RF

There have been several PV tolerant techniques explored to optimize the multi-ported register file in CPUs [21], [22] For instance, Liang et al. [21] proposed n% variable-latency RF (n% VL-RF) to partition all registers read ports into fast and slow categories. The slowest (100-n)% ports are marked as slow and accessed in two cycles. They are not considered in determining the frequency so that the chip frequency increases in the presence of PV. When a slow port is assigned to read a register, port switching technique is triggered to switch to a fast port attached to the same register and avoid the extra cycle delay.

In our baseline RF design, each SM is equipped with 128KB RF that is composed of 32K 32-bit registers. To reduce the impact of the extremely slow registers and boost the frequency under PV, one can apply the n% VL-RF technique to divide those 32K registers in the SM into fast and slow categories depending on their access delay. Based on our sensitivity analysis, setting n% as 70 percent delivers the optimal trade-off between frequency and the amount of slow registers. We implement the 70 percent variable-latency register file (70 percent VL-RF): the slowest 30 percent registers are classified into the slow category and will take two cycles to finish the read/write operation; frequency is determined by the slowest register of the remaining 70 percent registers in the fast category.

However, the variable-latency RF causes serious IPC loss. Recall that one operand access in an instruction warp involves the parallel accesses to 32 same-named registers from all threads within the warp. As long as there is one slow register among those 32 registers, the operand access latency is two cycles. In our baseline RF design [23], [24], although the 32 same-named registers are implemented close to each other and included in a single entry of the RF bank, the PV exhibits weak systematic effects for such a 1,024-bit wide entry. As a result, most 32 same-named registers contain at least one slow register, and the operand access latency is generally extended. We observe 23 percent IPC degradation under 70 percent VL-RF compared to the baseline case without any optimization under PV. Moreover, the variable-latency technique requires an extra bit per register to record the speed information as fast or slow, leading to large power and area overhead.

An alternative design to avoid the large IPC degradation is to consider each 32 same-named registers as a group, called register vector, and apply the 70 percent variable-latency technique at the register vector level, namely, 70 percent VL-RV. Similar to VL-RF, VL-RV requires an extra bit per register vector and causes considerable power and area overhead. More importantly, there is large delay variability among registers within a register vector but the slowest one determines its access delay. In other words, the variations at register level are significantly smoothed out at the register vector level. Thus, dividing register vectors into fast and slow categories has limited effect on frequency optimization.

2) Variable-Latency Sub-Banks in RF

In our baseline RF design [23], [24], each RF bank holds 64 1,024-bit wide entries. Therefore, PV exhibits much stronger systematic effects in the vertical direction than that in the horizontal direction within each RF bank. In this study, we focus on this wide-entry RF architecture containing 16 banks, and the explored techniques perfectly fit to other RF architecture which is discussed in Section 3.2.4.

We propose n% variable-latency sub-banks in GPGPUs register file (named as VL-SB) that vertically divides each RF bank into several sub-banks, and registers within each sub-bank share the same access speed that is constrained by the slowest one. Sub-banks exhibit distinct access delay, and the slowest (100-n)% ones are marked as slow. There is small delay variability among registers contained in a sub-bank due to the systematic effects, therefore, the large variations at register level is well maintained at the sub-bank level in VL-SB, which maximizes the frequency improvement under the VL technique.

Note that both read and write delay is considered in VL-SB. We observe that sub-banks with long (short) read delay are highly likely to exhibit long (short) write delay under the impact of systematic variations. This makes the sub-banks classification quite straightforward, and leads to only two categories that are fast read + fast write (i.e., fast sub-bank) and slow read + slow write (i.e., slow sub-bank). We choose to divide each RF bank into two sub-banks because further aggressively performing the finer-grained partition (i.e., four sub-banks or more) does not lead to an obvious frequency increase based on our sensitivity analysis. As can be seen, only 32 bits are required to keep the speed information for the 32 sub-banks in VL-SB, which causes negligible area overhead.

Since PV also causes delay variability among SMs in the same GPGPUs chip, one can change the value of n% during the sub-banks partition in different SMs to ensure the uniform frequency across SMs. In that case, each SM has distinct number of fast sub-banks which may affect the IPC. It is encouraged to employ the per-SM clocking as discussed in [11], we thus keep the uniform partition criterion (i.e., 70 percent) for each SM, and our techniques are orthogonal to

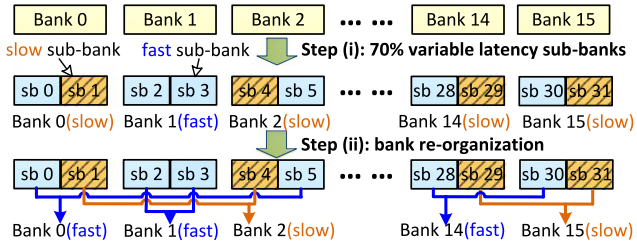


Fig. 4. An example of applying 70 percent VL-SB (step i) and RF-BRO (step ii)(sb:sub-bank).

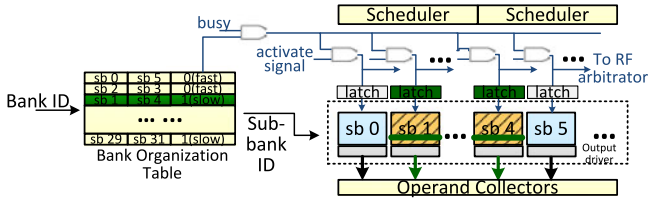


Fig. 5. Hardware implementation of variable-latency sub-banks and register file bank re-organization (RF-BFO).

the previously explored inter-SM level PV mitigation mechanisms [11].

Figs. 3b, 3c justify the effectiveness of 70 percent VL register vectors and 70 percent VL sub-banks by showing the RF frequency distribution when the two techniques are enabled, respectively. Every 32 nearby register vectors in VL-RV are grouped into an array to keep the same area overhead as VL-SB for a fair comparison. This makes 32 arrays in total, and the slowest one in the fastest 70 percent arrays decides the frequency. As Fig. 3b demonstrates, the mean frequency in 70 percent VL-RV increases by 10 percent compared to the baseline case presented in Fig. 3a, while 70 percent VL-SB in Fig. 3c is able to boost the mean frequency by 15 percent.

Note that the structural redundancy technique [12], which adds redundant structures to the processor as spares, is not applicable to eliminate the slowest x% critical paths in GPGPUs RF for frequency boosting. If just applying redundancy to replace the slowest x% register vectors which may distribute across all the RF banks, the register mapping becomes extremely complicated and impractical. If applying redundancy to replace the slowest x% RF banks, it will cause considerable area overhead. Moreover, selective word-line voltage boosting [2], which was applied to reduce the cache line access latency under PV effect, is not applicable to GPGPUs RF. Since RF is accessed more frequently than caches, selective word-line voltage boosting will cause considerable energy overhead to GPGPUs RF.

3.2.2 Register File Bank Re-Organization (RF-BRO)

The VL-SB technique faces the same challenge as VL-RF since it distributes registers belonging to the same register entry (i.e., register vector) into two sub-banks, which may be classified into different categories. We further propose RF bank re-organization (RF-BRO) on top of VL-SB that virtually combines two sub-banks from the same category to form a new RF bank. RF-BRO ensures the uniform access latency during the parallel accesses to 32 same-named registers. An operand access from an instruction warp is able to finish in one cycle as long as it is mapped to the newly formed RF bank that is composed of two fast sub-banks.

Fig. 4 shows an example of applying 70 percent VL-SB with RF-BRO on GPGPUs RF. When VL-SB is applied (step i in Fig. 4), the 16 RF banks are vertically divided into 32 sub-banks, and 22 of them are fast based on the 70 percent partition criterion (only five RF banks are shown in Fig. 4 for the illustration purpose). However, 10 RF banks still need two cycles to finish the read/write operation because they all contain slow sub-banks. With the help of RF-BRO (step ii in Fig. 4), sub-banks with the same type are virtually grouped to re-build the RF banks. For example, the new RF bank 0 is composed of sub-bank 0 and 5, and its access delay decreases to one cycle as it gets rid of the slow sub-bank 1. As a result, there are only five RF banks exhibiting two-cycle access delay. In other words, the percentage of slow RF banks reduces from 62.5 to 31.25 percent under RF-BRO.

3.2.3 The Implementation of VL-SB and RF-BRO

We divide the word-line in each RF bank into two segments to obtain the sub-banks. This is a widely used method in SRAM-based structures to reduce the delay [30] or save dynamic power when only a single word is accessed in a large cache [31]. Each word-line segment in the sub-bank is equipped with a local decoder in our VL-SB.

The implementation of VL-SB is similar to that of the VL-RF technique [21]: the speed information for each RF sub-bank will be collected by using BSIT [32] at chip test time. This information is used to mark each sub-bank as fast or slow, and also set an appropriate SM frequency. Note that the bank re-organization does not physically move any sub-bank during the chip fabrication. It *virtually* re-builds the RF banks by introducing a 16-entry bank organization table: each entry in the table records the IDs of two sub-banks that are assigned to the newly formed RF bank. In order to implement the bank re-organization technique, the IDs and the type (fast or slow) of every two same-type sub-banks are configured into a ROM at the chip test time. They will be loaded from the ROM and written into the SRAM-based bank organization table once GPUs are powered on.

Fig. 5 depicts the implementation of the two proposed techniques. During an operand access to the 32 same-named registers, the RF bank ID obtained from the warp and register IDs is used to index the bank organization table, and retrieve IDs and the speed type of corresponding sub-banks. The same entry in those two sub-banks will be activated simultaneously for operand access. For example, when RF bank 2 is accessed, sub-bank 1 and 4 are enabled as shown in Fig. 5. Meanwhile, the speed type obtained from the table will be ANDed with a busy signal, and a slow type leads to a distribution of the signal to all sub-banks. Only sub-banks that are activated to fulfill this operand access will receive the busy signal and save it into the attached latch. This is used to prevent the pre-charge at next cycle to ensure the register read lasting for two cycles and finishing correctly. In GPGPUs RF, an arbitrator is applied to select a group of non-conflicting accesses and send to the RF banks at every cycle [23], [24], [26]. Therefore, the busy signal is also sent to the RF arbitrator to stall the following register read/write to the same RF bank.

3.2.4 Feasibility in Alternative Register File Architecture

Alternative register file architectures are used in contemporary GPGPUs. One example is to group four SIMD lanes in an SM into a cluster, and eight clusters form a complete 32-wide SM [28], [29]. In this case, each cluster contains four register banks, and each entry in a bank is only 16-byte wide that contains the register values for four threads in a warp (i.e., four same-named registers). Thus 32 same-named registers are distributed into eight banks (one bank per cluster), and the same entries from eight RF banks are accessed simultaneously for one operand access. As can be seen, the systematic effects for those 32 same-named registers are weak since they are evenly distributed to different clusters. Neither VL-RF nor VL-RV techniques could deliver good frequency improvement. We can consider this narrow-width style RF architecture as vertically dividing the 1,024-bit wide register vectors (i.e., 32 same-named registers) into eight sub-banks. Thus the proposed RF-BRO technique can be directly applied for the performance optimization under PV: we will not sub-divide each register bank, instead we adopt the VL technique to those 16-byte wide banks, and virtually re-organize eight banks with the same speed to form the 32 same-named registers. In summary, our VL-SB RF design and the RF-BRO mechanism built upon it are applicable to other GPGPUs RF design.

3.3 Migigating the IPC Degradation under VL-SB and RF-BRO

Although the VL-SB+RF-BRO technique largely optimizes the GPGPUs RF frequency under PV impacts, there are still about 30 percent slow banks among the virtually re-organized RF banks, leading to around 9 percent IPC degradation based on our experimental results shown in Section 6.1. We further propose to harness the unique characteristics in GPGPUs applications to minimize the IPC loss. Note that the slow (fast) RF banks mentioned in this subsection are RF banks that are virtually composed of two slow (fast) sub-banks under RF-BRO.

3.3.1 Register Mapping under VL-SB and RF-BRO

The SM in Fermi style architecture is armed with two warp schedulers. Warps with odd and even IDs are dispatched into those two schedulers, respectively. At every cycle, two instruction warps are issued based on the round-robin scheduling policy, and they are likely to have identical PC since all threads in a kernel execute the same code. Mapping the same-ID registers from different warps into the same bank seriously exacerbates the bank conflicts, because different entries within a bank may be requested by the two simultaneously issued instruction warps. Therefore, the register-to-bank mapping mechanism follows the Eq. (3)

$$(warp\_ID + register\_ID)\%(the\ number\ of\ banks) \quad (3)$$

to ensure that different banks hold the same-ID registers across the warps. As Eq. (3) shows, consecutive warps tend to map their same-ID registers into nearby RF banks. For instance, R1 from warp0 and warp1 are mapped to bank 1 and bank 2, respectively. Generally, consecutive warps exhibit strong data locality [8], their same-ID registers should be allocated to RF banks with same speed type to ensure they execute at similar progress. We propose to save IDs of same-type sub-banks into consecutive entries in the ROM at the chip test time, therefore, bank0-bank10 are fast while bank11-bank15 are slow in the bank organization table under VL-SB and RF-BRO techniques. More specifically, the virtual banks are sorted from the fastest to the slowest in the bank organization table. By using Eq. (3), there are a number of registers per warp mapping to the slow RF banks. And the slow bank keeps registers with different IDs at the warp level. Fig. 6 demonstrates an example of register mapping: R11-R15 from warp0 while R0-R4 from warp11 are assigned to the slow RF bank11-bank15.

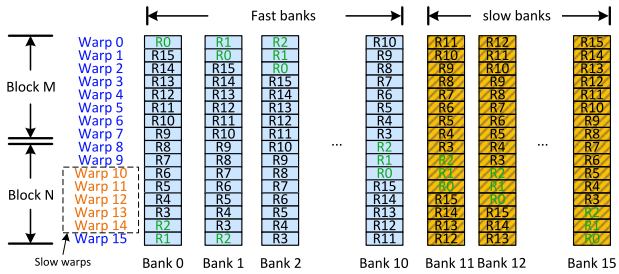


Fig. 6. The idea of FWAS. R0-R2 are frequently accessed small-ID registers. Warp 10-14 in block N are slow warps.

3.3.2 Fast-Bank Aware Register Mapping

It has been observed that around 50 percent of registers are not even allocated by the compiler for the application execution [25]. This unique feature can be leveraged to minimize the use of slow RF banks by mapping registers to fast banks to the maximum degree during the kernel launch time. For the benchmarks that have high RF utilization, slow banks are used to ensure high level TLP. We find that a small set of registers have much higher access frequency than other registers allocated to the same warp, and they are usually the registers with small ID. For instance, each warp in benchmark *BN* (detailed experiment methodologies are in Section 5) is assigned 14 register vectors, and R0-R2 are used 250 percent more frequently than R3-R13. This is because the compiler tends to re-use the small-ID registers. We explore a novel fast-bank aware register mapping mechanism (named as FBA-RM) that consists of two steps: (1) obtaining the register resource requirements at the kernel launch time, and mapping registers only to fast banks if they are large enough to hold all registers needed by the parallel threads (i.e., reducing the number of RF banks to 11 in Eq. (3) during the mapping); (2) allocating the large-ID registers to slow RF banks when they have to be used, therefore, the slow banks are rarely accessed. In that case, Eq. (3) is used for small-ID registers to fast banks mapping and large-ID registers to slow banks mapping, respectively.

The major disadvantage of the FBA-RM technique is the increased bank conflicts as most RF accesses are limited to fast banks, and it fails to effectively mitigate the IPC degradation. In Section 6.1, we perform the detailed evaluation about FBA-RM by comparing it with other proposed techniques in the following sections.

3.3.3 Fast-Warp Aware Scheduling (FWAS) Policy

Considering that modifying the register mapping mechanism to minimize the use of slow banks has little impact on

performance optimization, we thus adopt the default mapping mechanism in Fermi and propose a set of methods to hide the extra access delay on slow banks.

As Fig. 6 shows, the frequently accessed small-ID registers in a number of warps (e.g., warp 11) are mapped to slow RF banks, which seriously delay their execution progress. And we define this kind of warp as slow warp. We further define warps whose frequent register accesses in fast banks as fast warps (e.g., warp 0). The execution progress for fast warps is delayed somehow when the default warp scheduling policy (i.e., round-robin) is applied. This is because round-robin policy gives each warp the same issue priority, and when the slow warps occupy the pipeline resources (e.g., the issue and write slot), the ready fast warps cannot leverage those resources for execution. As a result, there is a small progress difference between fast and slow warps within the same SM. We propose the fast-warp aware scheduling policy (named as FWAS) that assigns fast warps higher issue priority than slow warps to maximize the progress difference between them. This explores the unique opportunities to mitigate the IPC loss as follows:

(1) The fast warps have shorter execution time, thus the RF resources allocated to these warps are able to serve more warps during execution. This is effective for kernels including a large number of blocks that cannot be fully distributed to SMs at one time;
(2) The fast warps are able to start their off-chip memory accesses earlier, which alleviates the memory contention under the round-robin policy and reduce the pipeline stall time. This is effective for memory-intensive benchmarks.

Fig. 6 explains the first opportunity in detail. Generally, there are multiple blocks executing concurrently in an SM. Fig. 6 shows an example SM with two blocks: block M and block N, each block contains eight warps. The frequently accessed registers (i.e., R0-R2) of all warps in block M (i.e., warp 0-7) are mapped to fast banks, so all warps belonging to block M are fast warps. On the contrary, in block N, most of their frequently accessed registers in warp 10-14 are mapped to slow banks, thus these warps are considered as slow warps. During program execution, FWAS prioritizes fast warps and allows them to finish earlier. As a result, warp 10-14 will left behind and become the bottleneck for block N. When all warps in block M finish execution, the new coming block within the same kernel will be assigned to take the resources (e.g., warp slots, registers) just released by block M. It also contains more fast warps (i.e., use those fast banks) because its warps will be assigned the same warp IDs as those warps in block M. On the other hand, block N will not release its resource until all its slow warps finish execution. As a result, the number of blocks that contain a larger amount of fast warps increases (in other words, fast banks are able to serve more warps) during the entire kernel execution, leading to the IPC improvement.

In order to implement the FWAS policy, the fast warps have to be identified at the issue stage. However, there is no clear boundary between fast and slow warps, because the frequently accessed registers in a few warps are allocated to both fast and slow banks under the default mapping mechanism, e.g., the heavily used R0-R2 in warp9-warp10 in Fig. 6. Although slow warps also access fast banks, it is highly possible that an instruction with fast bank access belongs to a fast warp. Instead of performing the accurate fast and slow warp identification, we choose to simply give the instruction warp that requires fast bank accesses higher issue priority. At the decode stage, an instruction warp is marked as fast if it has fast RF read/write. Note that only when all its operands reads are mapped to fast RF banks, an instruction warp is considered as possessing fast RF read. The one-bit fast/slow information combined with the ready bit is sent to the selection logic during the issue stage to perform the FWAS policy.

3.3.4 Hybrid RF Bank for Partially Active Warps

1) Observations on Partially Active Warps

In some GPGPUs benchmarks (e.g., *NN*), the block size is even smaller than the warp width of 32 threads, and warps do not contain enough active threads through the entire execution. Moreover, in benchmarks with heavy branch divergence, warps usually include several inactive threads. Both the two cases result in partially active warps, which will not use all the 32 registers contained in a register vector. We further find that a few register vectors usually have the fixed registers utilized in a branch induced divergent warp. This is because a divergent branch that is depending on the programmatic values (e.g., block ID, thread ID) always causes the fixed threads in a warp active, and such programmatic-value dependent branches occur quite frequently in GPGPUs applications [34]. The partially utilized register vector does not need to map to the pure fast RF bank that is composed of fast sub-banks only. As long as its utilized registers are allocated to fast sub-banks, the register access operation latency can be constrained into one cycle.

2) The Idea of Hybrid RF Bank

In this study, we propose to build multiple hybrid RF banks and harness the above unique characteristics in GPGPUs benchmarks to further improve the performance. Different from the RF-BRO technique that always combines same-type sub-banks to organize a RF bank, each hybrid RF bank consists of one fast and one slow sub-bank. It is particularly used to hold register vectors with fixed registers active in partially active warps. Since only the fast sub-bank portion is required to conduct the read/write operations, the hybrid RF banks can be treated as fast banks, which increase the total number of fast banks without sacrificing the RF frequency and meanwhile, mitigating the IPC loss. The new GPGPUs register architecture contains three types of RF banks: pure fast, pure slow, and hybrid categories.

Note that aggressively increasing the hybrid RF size reduces the pure fast RF size since the overall amount of sub-banks remains the same. This may force some fully occupied register vectors that are supposed to be mapped to pure fast RF banks being re-directed to the hybrid banks and using their slow-bank portion, thus, leading to the performance degradation. Ideally, the size of hybrid RF should match the total number of partially used register vectors. However, that number varies significantly across benchmarks.

The programmatic-value dependent branches can be detected at the compilation time using taint analysis [34]. We adopt this method to mark the special register vectors that prefer hybrid RF. The register namespace is partitioned into two sets of architectural register names representing

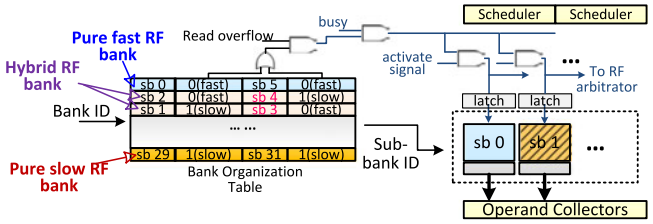


Fig. 7. The implementation of virtually building hybrid RF banks based on RF-BRO technique.

the pure fast/slow and hybrid RF, respectively. The amount of those two types of registers per thread will be sent to the GPU during the kernel launch time, and meanwhile, the RF banks are re-configured to match the resource requirements on hybrid RF banks. To avoid the excessive hybrid RF resources, the number of hybrid RF banks is equal to the integer part of a product of the RF banks quantity (i.e., 16) and the ratio of requested hybrid RF size to the overall requested RF size. For benchmarks that have insufficient threads to fill up a warp (this can be identified at the kernel launch time), all slow sub-banks will be combined with the fast ones when warps contain no more than 16 active threads, therefore, all RF banks become fast.

3) The Implementation of Hybrid RF Bank

To implement the dynamic hybrid RF re-organization under RF-BRO, we switch a certain number of sub-banks from pure slow RF banks with those from pure fast RF banks in the bank organization table during the kernel launch time. For example, sub-bank 3 and 4 are exchanged in Fig. 7 compared to that in Fig. 4, and build two hybrid RF banks. Two bits are used in each table entry to record every sub-bank's speed information. In a branch induced divergent warp, its register access time on the hybrid RF bank is fast only when all its active threads use the first half registers in a register vector. We compact the active threads of programmatic-value dependent branches into the left-most portion during kernel launch time, so that they are mapped to the fast sub-bank portion of the hybrid RF bank.

During the register access, Eq. (3) is applied separately for pure RF and hybrid RF mapping. In the case that there are insufficient hybrid RF banks, some hybrid RF IDs are redirected to pure RF banks, vice versa. A "read overflow" signal is enabled if the instruction's source pure RF ID is mapped to a hybrid RF bank at the register read stage. It is also asserted when the active threads in divergent warps fail to compact into the left-most portion. This means that the slow portion of the hybrid RF bank has to be enabled. During a RF bank read, the "read overflow" signal is ANDed with the output of an OR operation on the speed information bits for that bank, determining if a "busy" signal should be generated (shown in Fig. 7). Moreover, the overflow signal is always asserted when pure slow RF banks are accessed to ensure they have enough time to finish the access operation. Similarly, a "write overflow" signal is used for the write operation.

3.3.5 Putting All PV Mitigation Techniques Together

The idea of building hybrid RF banks to hold special register vectors is orthogonal to the warp scheduling policy, we thus propose to combine the two techniques in Sections 3.3.3 and 3.3.4 together based on VL-SB and RF-BRO, and maximize both IPC and frequency optimization. Since hybrid RF bank access just takes one cycle in majority of the time, instruction warps contain hybrid RF access is simply treated as fast warp when integrated with the FWAS technique.

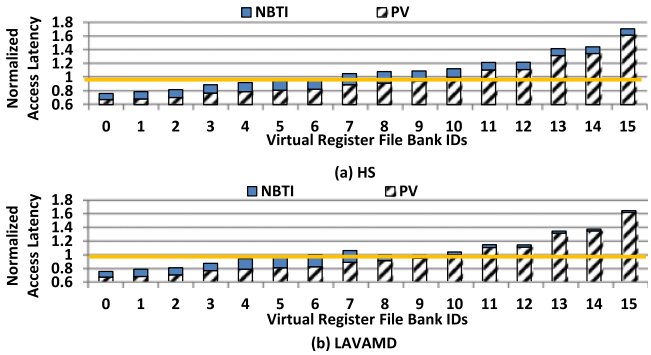


Fig. 8. Normalized access latency of each *virtual* register file bank under PV and NBTI for (a) HS and (b) LAVAMD. The yellow line indicates the normalized latency under PV impact only, called boundary.

The combined PV mitigation technique induces a 16-entry bank organization table, latches, and some simple logic gates to control busy and overflow signals. Based on our HSPICE simulation, these hardware extensions increase the access latency by 0.00019 ns and area by 0.000691 mm^2 under 32 nm technology, which are negligible compared with the sizable RF.

4 Mitigating the NBTI Impact on GPGPUs Register File

As introduced in Section 1, PV is a static impact on GPGPUs register file access latency, while NBTI will gradually wear out the transistors at runtime and cause a dynamic impact on the access latency. It is critical to consider the aggregated impact of PV and NBTI. In this section, we characterize and mitigate the NBTI impact on GPGPUs register file based on our PV mitigation techniques explored in Section 3.

4.1 The Impact of NBTI on GPGPUs Register File under PV

We first model the NBTI degradation in V_{th} by calculating the ΔV_{th} as described in Eq. (2). We obtain the PMOS stress cycles (t_{stress}) and recovery cycles (t_{rec}) via the microachitecture simulator (details about the simulator can be found in Section 5), and insert them into Eqs. (1) and (2) to determine the increase in V_{th} due to NBTI. According to the alpha power law [37], transistor switching delay increases as V_{th} increases. Thus we can model the NBTI impact as the increase in the access delay.

Fig. 8 shows the aggregated effect of PV and NBTI on the access latency of register file banks when running benchmarks *HS* and *LAVAMD*, respectively, for several years (e.g., 7 years). To well characterize the NBTI effect, the two benchmarks' execution are simulated on the same chip with the same PV impact. Note that the bank ID in this figure is the *virtual* ID under our PV mitigation techniques proposed in Section 3, and banks are sorted based on their access speed under PV only in descending order. The access latency of each bank is normalized to the slowest bank in the fast bank category under PV impact only (i.e., bank 10 in Fig. 8), which also determines the chip frequency. The yellow line in the figure emphasizes this normalized latency,

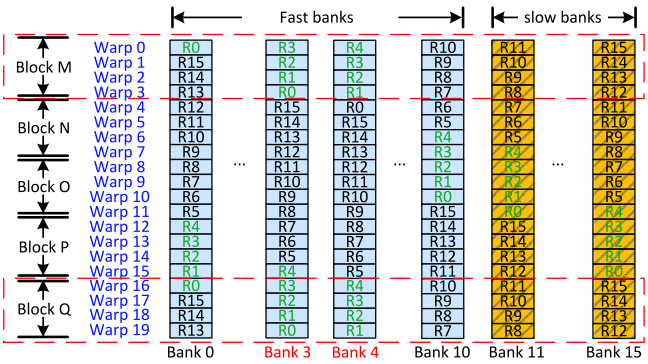


Fig. 9. The unbalanced utilization across RF banks. R0-R4 are frequently accessed registers. Bank 3 and 4 are most utilized in this example.

and we refer it as the *boundary*. Banks whose latency is smaller than the boundary are classified as fast banks, and those whose latency is larger are classified as slow banks.

As we expected, NBTI exacerbates the access latency of each bank in Fig. 8. After 7 years, the latency of some banks (e.g., banks 0, 1, 2, and 3) in both benchmarks is still far below the boundary. However, the access latency of some fast banks exceeds the boundary, such as the banks 7, 8, 9 and 10 in *HS*, and banks 7 and 10 in *LAVAMD*. They downgrade to the slow category if the same boundary applies. To ensure sufficient fast banks, a guardband (i.e., frequency degradation) is induced, which is determined by the slowest bank in the fastest 10 banks under the aggregated PV and NBTI effects.

Fig. 8 also shows the uneven utilizations across all the register file banks, which implies that NBTI causes different increase of the access latency to each bank. For example, bank 7 is utilized heavilier than bank 8 and bank 9 in *LAVAMD*. Thus after 7 years, bank 7 becomes much slower than bank 8 and bank 9 even though it was faster at the initial stage before aging. This motivates us to heavily utilize the banks far away from the boundary while reducing the stress on banks that are close to the boundary. Therefore, the guardband, i.e., the frequency degradation, can be well shrinked to boost the overall performance.

The uneven utilization across the banks is caused by a combined effect of several factors, such as the number of concurrent blocks, the block size, the number of frequently accessed registers and our proposed PV mitigation techniques. Figs. 6 and 9 show two different cases. In Fig. 6, two blocks execute concurrently in each streaming multiprocessor, and each block contains eight warps. Every warp has three frequently accessed registers (R0-R2). In contrast, there are five concurrent blocks in Fig. 9, and each block has only four warps. The first five registers (R0-R4) in each warp are frequently accessed.

For the case in Fig. 6, banks 2-7 are more heavily utilized than other banks. The reason is twofold. (1) All warps in block M are fast warps as discussed in Section 3.3.3, register resources allocated to block M (i.e., the first eight entries of each bank in Fig. 6) will be quickly released and re-assigned to the new coming block under the impact of our proposed FWAS technique. In other words, the first eight entries of each bank are serving more blocks and used more heavily than other entries. (2) Furthermore, when focusing on those first 8 entires of all banks, banks 2-7 serve more frequently accessed registers than other banks. E.g., bank 2 serves R2 for warp 0, R1 for warp 1 and R0 for warp 2, those three registers are all frequently accessed while bank 0 only serves one frequently accessed register, R0 for warp 0.

However, when the benchmark characteristics (e.g., number of concurrent blocks, the block size, and the frequently accessed registers) change, the case is totally different. For example, banks 3 and 4 instead of banks 2-7 are most heavily utilized in Fig. 9. In this case, entries that marked by the red dotted line in the figure will serve more blocks under our FWAS technique, and banks 3 and 4 serve more frequently accessed registers.

This observation indicates that we can design a mechanism that considers the bank utilization differences to combat the NBTI effect and shrink the required guardband.

4.2 Mitigating the NBTI Effect on GPGPUs Register File under PV

4.2.1 *The Block-Level RF Bank Renaming*

In this section, we introduce a novel mechanism that dynamically renames the RF banks required by each block to mitigate the NBTI impact. This mechanism takes benchmark characteristics (e.g., number of concurrent blocks, block size, and amount of frequently accessed registers) into consideration, and balances the utilization across the register banks based on their access latency under PV and NBTI effects. Since the register file allocation and deallocation are performed at the block level, we propose to rename the register banks when the block is launched to the GPGPUs SM, which provides a fine-grained and low-cost solution.

As we observed in Section 3.3.2, registers with small IDs are always frequently accessed, it is safe to assume that registers with smaller IDs are accessed more heavily than those with larger IDs. A simple way to combat the aggregated NBTI and PV effects is to map registers with smaller IDs to the faster virtual banks. In other words, R0s from all warps in a block will be mapped to the virtual bank 0 in our bank organization table, R1s will be mapped to the virtual bank 1, and so on so forth. This will require two-stage mapping: register-to-bank mapping so that the same-named registers are mapped to the same bank; and the bank-to-virtual-bank mapping so that the bank will be further directed to certain virtual bank with appropriate access delay. However, simply renaming all same-named registers (e.g., R0s and R1s) within a block to the same bank causes serious bank conflicts, as we discussed in Section 3.3.1. Our bank renaming technique will adopt the default register-to-bank mapping mechanism (as shown in Eq. (3)), and focus on renaming the bank IDs obtained from Eq. (3) to the appropriate virtual bank.

Considering that our bank renaming technique requires the per bank access latency, a NBTI sensor is attached to each RF sub-bank. The speed of a virtual bank is determined by its slowest sub-bank. Based on the collected information, the RF virtual banks are sorted from the fastest to the slowest. When a block is launching, we first check the R0 from its last warp. By using the Eq. (3), we are able to find out the corresponding bank, saying bank α, for this R0. Bank α will then be renamed to the fastest virtual bank from the bank category (either fast or slow bank category) it belongs to. Similarly, the bank, saying bank β, that holds R1 from

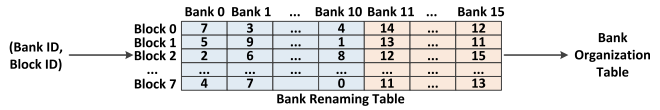


Fig. 10. The implementation of dynamic register file bank renaming for each block.

the last warp will be renamed to the fastest bank from the remained unmapped virtual banks in the bank category that bank β belongs to, and so on so forth. This is because the bank that holds R0 from the last warp of a block tends to hold the frequently accessed registers from other warps in the same block, thus, most heavily utilized. Take Fig. 9 as an example, when block P is coming, its last warp is warp15, and bank 15 will hold R0 from warp15. As can be seen from the figure, bank 15 also holds other frequently accessed registers (i.e., R3 from warp12, R2 from warp13, and R1 from warp14). It implies that bank 15 will be most heavily used for block P. Since bank 15 belongs to the slow bank category, our technique will map bank 15 to the fastest bank from that slow category, which is virtual bank 11 in our bank organization table. Moreover, bank 0 holds R1 from warp15 and it belongs to the fast bank category, therefore, bank 0 should be mapped to the fastest available bank in the fast category, which is virtual bank 0 in our bank organization table.

By doing this, the faster virtual banks from both fast and slow categories will be more heavily used, reducing the NBTI stress on banks whose access latency is close to the boundary. Note that our mapping mechanism considers faster virtual banks from the fast and slow bank categories, respectively, it still maintains the benefit of our PV mitigation techniques proposed in Section 3. One may notice that our technique could map the frequently accessed registers from different blocks to the same virtual bank. This has little impact on the bank conflict because warps from different blocks usually exhibit different execution progress and they are unlikely to access the same bank simultaneously.

To implement our mapping technique, we attach a register bank renaming table to each SM, as Fig. 10 shows. Each row in this table contains the bank renaming information for each block. All warps within the same block follow the same renaming. The column in the table indicates the virtual bank ID of the given bank ID. The block ID and the bank ID obtained from Eq. (3) are used together to find the corresponding virtual bank ID, which will be used as the index to the bank organization table. For example, one warp in block 2 is going to access bank 1 in Fig. 10. The corresponding virtual bank ID in the bank renaming table is bank 6, which means bank 1 is renamed as virtual bank 6 for this warp. Therefore, the sixth entry in the bank organization table will be accessed to finally retrieve the two physical sub-banks. The number of entries in our bank renaming table is determined by the maximum number of concurrent blocks supported by the SM (e.g., 8 blocks in our GPU configuration), and the number of columns is determined by the number of register banks (e.g., 16). Furthermore, each element in the table is 4-bit long to keep the index of each virtual bank ID. Thus the overall size of the bank renaming table is only 64 B per SM, which is very small and its latency is negligible based on our gate-level modeling.

4.2.2 The Kernel-Level RF Bank Re-Organization

Recall from Section 3, each virtual RF bank contains two sub-banks with distinct access latency, and its speed is determined by the slowest sub-bank. During the program execution, the two sub-banks may have different utilizations and degrade differently due to partially active warps described in Section 3.3.4. In other words, the access latency to each virtual bank is varying all the time which will affect its ranking in the bank reorganization table. Therefore, the bank re-organization table has to update periodically to ensure the correct ranking for our renaming technique.

Since the bank re-organization technique explored in Section 3.2.2. only considers the PV impact and is predetermined at the chip test time, in this subsection, we dynamically re-organize the sub-banks to well capture the aggregated impact of PV and NBTI on the access latency of all the sub-banks. We propose to trigger the bank re-organization at each kernel launch time. This is because dynamically re-organizing banks during the kernel execution (e.g., at the block allocation time) will cause the data remapping problem for the active contents currently saved in the banks, leading to large control overheads.

The implementation of this mechanism is quite simple: at kernel launch time, the speed information of each sub-bank is obtained and sorted. Every two continuous sub-banks in this ranking will be virtually combined into one virtual RF bank, and recorded in the bank organization table. For example, when the sub-bank ranking is listed as sb5, sb3, sb28, sb0, . . . , sb5 and sb3 will be organized as virtual bank 0 and saved in the first entry of the bank organization table, similarly, sb28 and sb0 are treated as virtual bank 1 and saved in the second entry, etc. In this way, the bank reorganization table is dynamically adapted to the PV and NBTI effects.

4.2.3 The Combined PV and NBTI Mitigation Technique

As can be seen, the above two NBTI techniques work at the block level and kernel level, respectively. They can be combined together, called NBTI optimization (NBTI-opt). NBTI-opt further adds a 8-entry bank renaming table and simple control logics. Based on our HSPICE simulation, these hardware extensions increase the access latency by 0.00059 ns and area by 0.002 mm^2 under 32 nm technology. Since NBTI-opt considers the bank speed variations under both PV and NBTI effects, it can be combined with our PV mitigation techniques (named as PV-opt) naturally. We name the combined PV and NBTI mitigation technique as PV-opt +NBTI-opt. We also evaluate the power overhead of PV-opt and NBTI-opt, which is only 1.02mW on average, which is negligible. Note both the hybrid bank technique in Section 3.3.4 and the kernel-level re-organization in Section 4.2.2 dynamically form virtual banks at the kernel launch time. When combined together, the kernel-level re-organization will be triggered earlier to regroup the sub-banks with similar speed, and the hybrid banks are further built up based on the new re-organization table.

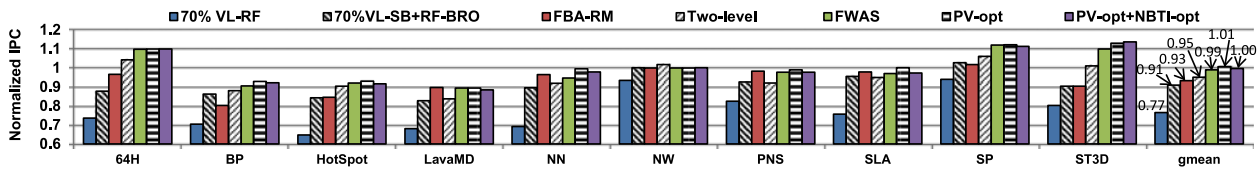


Fig. 11. Normalized IPC results under variable-latency RF (70 percent VL-RF), our proposed variable-latency sub-banks and RF bank re-organization (70 percent VL-SB+RF-BRO), fast-bank aware register mapping (FBA-RM), two-level scheduling, our proposed fast-warp aware scheduling, 70% VL-SB + RF-BRO + FWAS + our hybrid banks (PV-opt), and PV-opt + our NBTI mitigation techniques (PV-opt+NBTI-opt). The results are normalized to the baseline case without any optimization.

5 Experimental Methodology

We use a cycle-accurate, open-source, and publicly available simulator GPGPU-Sim (v3.1.0) [26] to evaluate the IPC optimization under our proposed methodologies. Note that our 70 percent variable-latency technique causes the frequency variations among SMs, we also model that SM level frequency differences into the simulator. Our baseline GPGPUs configuration models the Nvidia Fermi style architecture: the GPU contains 15 SMs; the warp size is 32; each SM supports 1,536 threads and eight blocks at most; each SM contains 128 KB registers, 16 KB L1 data cache, and 48 KB shared memory; L2 cache size is 768 KB; the scheduler applies the round robin among ready warps scheduling policy. The experimental methodology to model the PV and NBTI impacts have been introduced in Sections 3.1 and 4.1, respectively. We collect a large set of GPGPU workloads from Nvidia CUDA SDK [5], Rodinia [6], and Parboil [7]. The benchmarks show significant diversity according to their kernel characteristics, divergence, memory access patterns, and so on.

6 Evaluation

6.1 IPC Improvement

6.1.1 Evaluation of VL-SB with RF-BRO

To evaluate the effectiveness of variable-latency sub-banks with register file bank reorganization (RF-BRO), we compare it with the VL-RF technique explored by Liang and Brooks [21]. Fig. 11 shows the IPC of the investigated benchmarks when those two PV mitigation techniques are enabled. The results are normalized to the baseline case without any optimization (i.e., frequency is determined by the slowest critical path). As discussed in Section 3.3.1, bank0-bank10 are always fast while bank11-bank15 are always slow in all chips under RF-BRO. The IPC results are identical for all chips and only one chip's IPC results for RF-BRO based techniques are shown in Fig. 11. The averaged IPC results across all 100 chips are shown for 70 percent VL-RF technique. In the baseline case, all registers have one-cycle access latency, and it has the same IPC as the ideal case without PV. As it shows, when compared with the VL-RF mechanism, VL-SB+RF-BRO successfully reduces the IPC loss from 23 to 9 percent on average across all the benchmarks since it re-organizes sub-banks to deliver a considerable amount of fast RF banks. While VL-RF focuses at a quite fine-grain register level classification, making it impossible to apply the RF-BRO for fast RF bank reorganization and almost all the register vectors accesses take two cycles.

Interestingly, the IPCs of benchmark *NW* under both VL-RF and VL-SB+RF-BRO are already approaching to 1. This is because *NW* includes very few threads, and there are insufficient warps running concurrently in the SM to even hide the stalls for true data dependencies between consecutive instructions from a single warp (in absence of the long memory operations). As a result, the extra register access time is well absorbed by those stall cycles, and has little impact on IPC. On the other hand, the IPC decreases considerably in several computation intensive benchmarks under VL-SB+RF-BRO, because there are few stall cycles helping to hide long RF access delay. For instance, the IPC reduction is 18 percent in benchmark *LavaMD* that makes SM active 80 percent of the total execution time.

6.1.2 Evaluation of FWAS

Fig. 11 also presents the normalized IPC results when the fast-warp aware scheduling policy is enabled. We compare FWAS with the fast-bank aware register mapping (named as FBA-RM) discussed in Section 3.3.2. Recently, a two-level scheduling policy has been proposed to boost performance [8]. This policy splits warps into groups and execute groups in different paces, which alleviate the memory contentions. We thus introduce the two-level policy into VL-SB+RF-BRO and compare it with our proposed FWAS policy. As shown in Fig. 11, all the three techniques are able to mitigate the IPC reduction compared to VL-SB+RF-BRO technique: the IPC losses under FBA-RM, two-level, and FWAS are 7, 5, and 1 percent, respectively.

The effectiveness of FBA-RM is highly related to the register utilization. For instance, *NN* uses less than 10 percent of the register file through the entire execution. The fast RF banks are far enough to support the requirement. Moreover, since extremely few registers are utilized per warp in *NN*, pushing them to the fast banks negligibility increases the bank conflicts. As a result, the IPC loss of *NN* decreases from 11 percent under VL-SB+RF-BRO to only 4 percent under FBA-RM. Note that VL-SB+RF-BRO applies the default register mapping mechanism, the slow RF banks have the same utilization as the fast ones no matter how many registers are needed, leading to a considerable IPC loss for *NN*. On the other hand, the performance penalty is severely exacerbated when benchmarks need more register resources. Take *HotSpot* as an example, it requires around 80 percent of the register file. When FBA-RM allocates the frequently used registers to fast banks, the negative impact caused by the increased bank conflicts outweighs the positive effect of the decreased slow banks accesses, and results in 16 percent IPC degradation.

As Fig. 11 shows, the two-level technique integrated with VL-SB+RF-BRO can effectively improve IPC on multiple memory intensive benchmarks, such as *64H* and *ST3D*, as it decreases stall cycles caused by long-latency memory

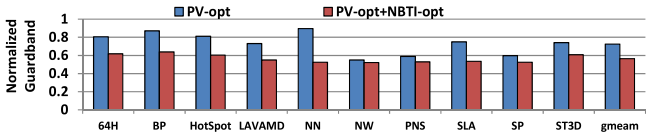


Fig. 12. Normalized guardband under PV-opt and PV-opt+NBTI-opt.

operations. The group size is fixed (i.e., eight warps) in two-level policy, and there are only eight warps, i.e., one group, in most SMs when executing *PNS*. The inter-group level scheduling in the two-level is inactive.

Similar to the two-level technique, FWAS shows the strong capability to mitigate the IPC loss for memory-intensive benchmarks. Moreover, FWAS does not have any constraint on warp grouping, and it successfully mitigates the IPC loss of *PNS* to only 2 percent which is far better than that under the two-level technique. On average across the memory-intensive benchmarks (i.e., *64H*, *NW*, *PNS*, and *ST3D*), FWAS boosts the IPC to 105 percent when normalized to the baseline case, which implies a 12 percent performance improvement compared to VL-SB+RF-BRO. Additionally, FWAS can effectively optimize IPC for computation-intensive benchmarks, especially those including numerous blocks, because it makes the fast RF banks to support warps at the most degree. For instance, it induces 8 percent IPC gains for both *HotSpot* and *LavaMD* compared to VL-SB+RF-BRO.

6.1.3 Evaluation of Hybrid Banks

We further evaluate the normalized IPC results when virtually building hybrid RF banks at kernel launch time for partially active warps, named as hybrid banks. The results of building hybrid banks on top of all other proposed PV mitigation techniques are presented in Fig. 11, which is named as PV-opt. As expected, the hybrid technique can further improve IPC to even 1 percent higher than baseline on average across all benchmarks. This is because it directs the special register vectors in hybrid RF banks whose slow portions are rarely utilized. For instance, *NN* is a typical benchmark that each warp has less than 16 active threads, its IPC result is the same as the baseline case since each RF bank is treated as fast. In addition, all divergent branches in *BP* are depending on programmatic values, hence, most register vectors in divergent warps are allocated to hybrid banks, which helps to boost the IPC significantly.

6.1.4 Evaluation of NBTI Mitigation Techniques

Since the proposed NBTI optimizations, i.e., bank-level renaming and kernel-level re-organization, are built on top of the PV mitigation techniques, we further investigate the IPC results of combining all NBTI and PV mitigation techniques, named as PV-opt+NBTI+opt in Fig. 11. As can be seen, PV-opt+NBTI-opt maintains the IPC improvement achieved by the PV-opt since it adopts the same register organization and scheduling policy proposed by the PV-opt. The IPC results of some benchmarks (e.g., *HotSpot* and *SLA*) under PV-opt+NBTI-opt are slightly lower than that under PV-opt. This is because warps from different blocks may access the same virtual bank under the bank renaming technique, leading to a bank conflict. However, this inter-block bank conflict occurs infrequently, and causes less than 1 percent performance loss as shown in Fig. 11.

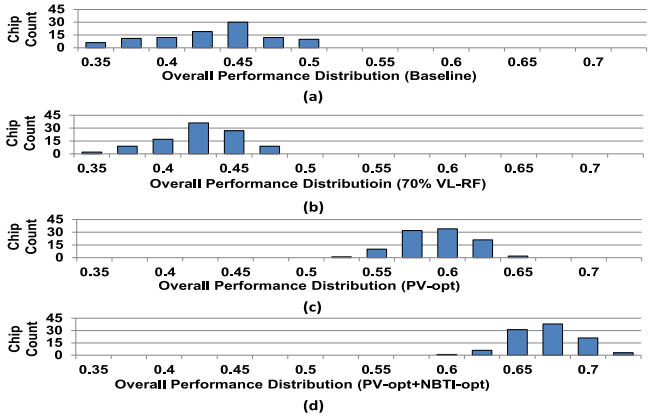


Fig. 13. Overall performance (IPC×frequency) distribution for 100 chips under (a) baseline, (b) 70 percent VL-RF, (c) PV-opt, and (d) PV-opt +NBTI-opt. The performance of each chip is normalized to the chip without PV and NBTI imapct.

6.2 Guadband Reduction

Fig. 12 shows the guardband requirement for each benchmark under PV-opt and PV-opt+NBTI-opt, respectively. All results are normalized to the baseline guardband requirement without any PV or NBTI optimizations. On average, PV-opt reduces the guardband requirement by 27 percent, while PV-opt+NBTI-opt further substantially reduces it to 56 percent. This confirms that our proposed PV and NBTI mitigation techniques effectively reduce the frequency loss due to the hardware variability. One may notice that PV-opt +NBTI-opt only reduces the guardband by 3 percent compared with PV-opt for *NW*. This is because *NW* has extremely low utilization on registers, and the NBTI impact on *NW* is very small. In contrast, PV-opt+NBTI-opt achieves 38 percent guardband reduction on top of PV-opt for *NN* because *NN* has quite unbalanced utilization across RF banks, which are then well balanced by the NBTI mitigation techniques in PV-opt+NBTI-opt.

6.3 Overall Performance Improvement

Fig. 13 compares the overall performance (IPC × frequency) of each case, and each sub-figure presents the performance distribution over the 100 investigated chips. The performance of each chip is the average performance of all benchmarks normalized to the ideal case without any PV and NBTI impacts. As Fig. 13a shows, the mean performance of the baseline case (i.e., the case with PV and NBTI impacts but no optimization) is only 45 percent of the ideal case without PV and NBTI. PV-opt significantly improves the mean performance by 15 percent over the baseline case, as shown in Fig. 13c. Our PV-opt achieves 12 percent higher performance than 70 percent VL-RF (shown in Fig. 13b) due to its substantial (i.e., 24 percent) IPC improvement. Our PV-opt+NBTI-opt further improves the frequency by 7 percent comparing to PV-opt with less than 1 percent IPC degradation, and improves the overall mean performance by 22 percent comparing to the baseline case, as shown in Fig. 13d.

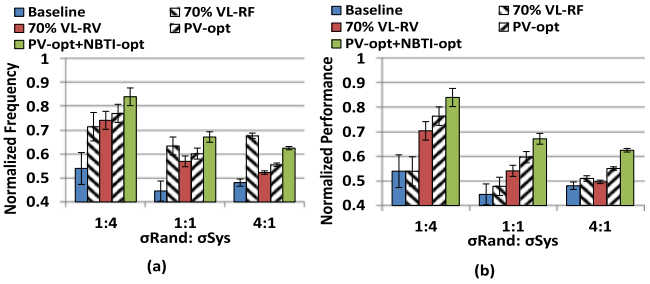


Fig. 14. Averaged (a) frequency and (b) overall performance (IPC×frequency) results under different random and systematic component ratios. The standard deviations across all chips are also shown in each case.

6.4 Sensitivity Analysis

In this section, we further evaluate the effectiveness of our proposed techniques when varying the random and systematic component ratios. Figs. 14a, 14b shows the averaged frequency, the averaged overall performance and the standard deviation under various $\sigma_{rand} : \sigma_{sys}$ scenarios. Both the frequency and overall performance are normalized to the ideal case without any PV or NBTI impacts. As it shows, the frequency of PV-opt and PV-opt+NBTI-opt increases as the systematic variations component increases. Under the scenario of $\sigma_{rand} : \sigma_{sys}$ = 1:4, the frequency loss of PV-opt +NBTI-opt is only 16 percent compared to the ideal case without PV and NBTI impacts. This is because our technique is effective to leverage the systematic variations to provide good frequency boosting. Fig. 14 also compares PV-opt and PV-opt+NBTI-opt with baseline case without any optimization, 70 percent VL-RF, and 70 percent VL-RV. As it shows, the frequency of PV-opt and PV-opt+NBTI-opt is better than 70 percent VL-RV under all three $\sigma_{rand} : \sigma_{sys}$ scenarios. This confirms that PV exhibits stronger systematic effects in the vertical direction than that in the horizontal direction. Also, the overall performances of both PV-opt and PV-opt+NBTI-opt are better than all other three mechanisms under the three $\sigma_{rand} : \sigma_{sys}$ scenarios. Especially, even when systematic component has minimum impact on variations (i.e., $\sigma_{rand} : \sigma_{sys}$ = 4:1), PV-opt+NBTI-opt is still 13 percent better than the baseline case due to its frequency benefit, and 11 percent better than 70 percent VL-RF due to its IPC benefit.

7 Related Work

There have been several studies on analyzing and mitigating the PV impact on GPGPUs. For instance, Lee et al. [11] observed the large frequency variations across SMs, and proposed the PV mitigation techniques at the SM level that running each SM at its maximum frequency independently and disabling the slowest SM. Krimer et al. [14], [15] investigated the timing errors under the impact of process variations, and proposed lane decoupling that enables each SPMD lane to tolerate timing errors independently of other adjacent lanes, therefore, boosting the GPGPUs throughput. Our PV mitigation techniques are orthogonal to those previously proposed mechanisms.

Several techniques are proposed to mitigate the NBTI impact in GPUs [37], [38], [39]. For example, Zhang et al. [38] proposed to a two-level scheduler to mitigate the aging effect in the warp scheduler of GPUs. Namaki-Shoushtari et al. [39] proposed a novel register file allocation mechanism to reduce the stress on GPGPUs RF. However they did not consider the PV impact. Our block-level bank renaming and kernel-level bank re-organization mitigate the NBTI impact for GPGPUs under PV.

8 Conclusions

This paper aims to mitigate the susceptibility of GPGPUs register file to PV and NBTI. For PV mitigation, we first propose to vertically divide RF banks into sub-banks, and explore the variable-latency technique at sub-bank level to perform the coarse-grain register classification which maximizes the frequency optimization. We then propose to reorganize RF banks by virtually combining the same-type sub-banks at the chip test time, leading to the fast and slow RF bank categories. The IPC degrades when using the slow RF banks during the kernel execution. Thus we further explore two techniques that leverage the unique features in GPGPUs application to mitigate the IPC loss: the FWAS (fast-warp aware warp scheduling) technique helps to minimize the use of slow RF banks; and the hybrid bank technique allocates the partially used register vectors into the fast sub-bank portions of the virtually built hybrid RF banks and eliminate the negative impact of their slow sub-bank portions on performance. To further tolerate the NBTI degradation, we dynamically rename the register banks during the block launch time, which effectively migrates the NBTI stress on banks near the frequency boundary to those that are much far away to the boundary. We also re-organize the RF banks at the kernel launch time to well capture the aggregated PV and NBTI impact which keeps changing as time goes by. Our PV and NBTI mitigation techniques obtains 22 percent overall performance improvement compared to the baseline case without any optimization.

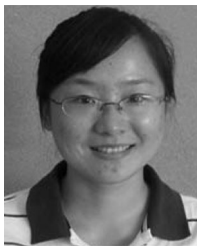
Acknowledgments

This work is supported in part by US National Science Foundation (NSF) grants CCF-1320730, CCF-1351054 (CAREER), EPS-0903806 and matching support from the State of Kansas through the Kansas Board of Regents. This work is also cofunded by NSF of China grant 91418203.

References

[1] NVIDIA. CUDA Programming Guide Version 3.0., Nvidia Corporation, 2010.
[2] Y. Pan, J. Kong, S. Ozdemir, G. Memik, and S. W. Chung, "Selective wordline voltage boosting for caches to manage yield under process variations," in *Proc. 46th Annu. Des. Autom. Conf.*, 2009, pp. 57–62.
[3] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas, "EVAL: Utilizing processors with variation-induced timing errors," in *Proc. 41st IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 423–434.
[4] A. Agrawal. A. Ansari, and J. Torrellas, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on chip eDRAM modules," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 84–95.
[5] (2015). [Online]. Available: http://www.nvidia.com/object/cuda_sdks.html.
[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.
[7] (2012). Parboil Benchmark Suite [Online]. Available: URL: http://impact.crhc.illinois.edu/parboil.php.

[8] V. Narasiman, C. J. Lee, M. Shebanow, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 308–317.
[9] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of Many-cores to process variations at Near-threshold voltages," in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2012, pp. 1–11.
[10] N. S. Kim, T. Kgil, K. Bowman, V. De, and T. Mudge, "Total power optimal pipelining and parallel processing under process variations in nanometer technology," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2005, pp. 535–540.
[11] J. Lee, P. Ajgaonkar, and N. S. Kim, "Analyzing throughput of GPGPUs exploiting within-die core-to-core frequency variation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2011, pp. 237–246.
[12] S. Seo, R. G. Dreslinski, M. Woh, Y. Park, C. Charkrabari, S. Mahlke, D. Blaauw, and T. Mudge, "Process variation in Near-threshold wide SIMD architectures," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 980–987.
[13] 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," *IEEE Trans. Semicond. Manuf.*, vol. 21, no. 1, pp. 3–13, Feb. 2008.
[14] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of Wide-SIMD architectures," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012, pp. 237–248.
[15] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang, "Synctium: A near-threshold stream processor for energy-constrained parallel applications," *IEEE Comput. Archit. Lett.*, vol. 9, no. 1, pp. 21–24, Jan. 2010.
[16] (2009). NVIDIA's Next Generation CUDA Compute Architecture: Fermi [Online]. Availavle: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
[17] D. Kanter. (2010). AMD's cayman GPU architecture [Online]. Available: http://realworldtech.com/page.cfm?ArticleID=RWT121410213827
[18] (2012). NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110, http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.
[19] J. Tschanz, J. Kao, and S. Narendra, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1396–1402, Nov. 2002.
[20] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A process-tolerant cache architecture for improved yield in nano-scale technologies," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 1, pp. 27–38, Jan. 2005.
[21] X. Liang, and D. Brooks, "Mitigating the impact of process variations on processor register files and execution units," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2006, pp. 504–514.
[22] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating parameter variation with dynamic fine-gain body biasing," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2007, pp. 27–42.
[23] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M.i Guo, R. Canal, and X. Liang, "An energy-efficient and scalable eDRAM-based register file architecture for GPGPU," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 344–355.
[24] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 487–498.
[25] M. Abdel-Majeed and M. Annavaram, "Warped register file: A power efficient register file for GPGPUs," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 412–423.
[26] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2009, pp. 163–174.
[27] W. W. L. Fung and T. Aamodt, "Thread block compaction for efficient SIMT control flow," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 25–36.
[28] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 235–246.
[29] M. Gebhart, S. W. Keckler, and W. J. Dally, "A Compile-time managed multi-level register file hierarchy," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2011, pp. 465–476.
[30] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to understand large caches," Univ. Utah and Hewlett Packard Lab., HP Laboratories, Tech. Rep. HPL-2009-85, 2007.
[31] M. Yoshimoto, K. Anami, H. Shinohara, T. Yoshihara, H. Takagi, S. Nagao, S. Kayano, and T. Nakano, "A divided word-line structure in the static ram and its application to a 64k full CMOS ram," *IEEE J. Solid-State Circuits*, vol. 18, no. 5, pp. 479–485, Oct. 1983.
[32] M. Tehranipour, Z. Navabi, and S. Falkhrai, "An efficient BIST method for testing of embedded SRAMs," in *Proc. IEEE Int. Symp. Circuits and Systems*, 2001, pp. 73–76.
[33] N. Goswami, B. Cao, and T. Li, "Power-performance co-optimization of throughput core architecture using resistive memory," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, 2013, pp. 342–353.
[34] M. Rhu and M. Erez, "Maximizing SIMD resource utilization in GPGPUs with SIMD lane permutation," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 356–367.
[35] 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 *Proc. Conf. Des., Autom. Test Eur.*, 2014, pp. 1–6.
[36] J. Tan and X. Fu, "Mitigating the susceptibility of GPGPUs register file to process variations," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2015, pp. 969–978.
[37] X. Chen, Y. Wang, Y. Liang, Y. Xie, and H. Yang, "Run-time technique for simultaneous aging and power optimization in GPGPUs," in *Proc. 51st Annu. Des. Autom. Conf.*, 2014, pp. 1–6.
[38] Y. Zhang, S. Chen, L. Peng, and S. Chen, "Mitigating NBTI degradation on finfet GPUs through exploring device heterogeneity," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2014, pp. 577–582.
[39] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Guppta, and R. K. Gupta, "ARGO: Aging-aware GPGPU register file allocation," in *Proc. 9th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codes. Syst. Synthesis*, 2013, pp. 30:1–30:9.
[40] A Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicors," in *Proc. 41st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 129–140.



Jingweijia Tan received the BS degree in computer science and technology from Jilin University, China, and the MS degree in computer science from the University of Kansas. She is currently working toward the PhD degree in electrical engineering at the University of Houston. Her research interests include computer architecture, high-performance computing, GPUs, hardware reliability and variability, energy-efficient processor design, and emerging technologies. She is a student member of the IEEE.

Mingsong Chen received the BS and ME degrees from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006, respectively, and the PhD degree in computer engineering from the University of Florida, Gainesville, in 2010. He is currently an associate professor with the Institute of Computer Science and Software Engineering of East China Normal University. His research interests are in the area of design automation of cyber-physical systems, computer architecture and formal verification techniques. Currently, he is an associate editor of *Journal of Circuits, Systems and Computers*.

Yang Yi received the BS and MS degrees in electronic engineering from Shanghai Jiao Tong University, and the PhD degree in electrical and computer engineering from Texas A&M University. She is an assistant professor in the Department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU). Her research interests include very large scale integrated (VLSI) circuits and systems, computer aided design (CAD), neuromorphic architecture for brain-inspired computing systems, and low-power circuits design with advanced nano-technologies for high-speed wireless systems.

Xin Fu received the PhD degree in computer engineering from the University of Florida, Gainesville, in 2009. She is currently an assistant professor at the Electrical and Computer Engineering Department, the University of Houston, Houston. Her research interests include computer architecture, high-performance computing, hardware reliability and variability, energy-efficient computing, and mobile computing. She received 2014 US National Science Foundation (NSF) Faculty Early CAREER Award, 2012 Kansas NSF EPSCoR First Award, and 2009 NSF Computing Innovation Fellow. She is a member of the IEEE.