Bank Stealing for Conflict Mitigation in GPGPU Register File

Naifeng Jing, Shuang Chen, Shunning Jiang, Li Jiang, Chao Li, Xiaoyao Liang*
Shanghai Jiao Tong University, Shanghai, China
jingnaifeng@ic.sjtu.edu.cn, liang-xy@cs.sjtu.edu.cn

Abstract— Modern General Purpose Graphic Processing Unit (GPGPU) demands a large Register File (RF), which is typically organized into multiple banks to support the massive parallelism. Although heavy banking benefits RF throughput, its associated area and energy costs with diminishing performance gains greatly limit future RF scaling. In this paper, we propose an improved RF design with a bank stealing technique, which enables a high RF throughput with compact area. By deeply investigating the GPGPU microarchitecture, we identify the deficiency in the state-of-the-art RF designs as the bank conflict problem, while the majority of conflicts can be eliminated leveraging the fact that the highly-banked RF oftentimes experiences under-utilization. This is especially true in GPGPU where multiple ready warps are available at the scheduling stage with their operands to be wisely coordinated. Our lightweight bank stealing technique can opportunistically fill the idle banks for better operand service, and the average GPGPU performance can be improved under smaller energy budget with significant area saving, which makes it promising for sustainable RF scaling.

Keywords— GPGPU, register file, bank conflict, bank stealing, area efficiency

I. Introduction

By exploiting the parallelism using massive concurrent threads, modern GPGPUs have emerged as pervasive alternatives for high performance computing. To hide the memory access latency, GPGPUs are featured by a *zero-cost context switching*, which is supported by a large-sized RF such that contexts of all the concurrent threads can be held in each Streaming Multiprocessor (SM). For instance, the up-to-date Nvidia Kepler architecture is equipped with 65,536 32-bit (256KB) registers to support 2,048 threads per SM. On top of that, the RF capacity keeps increasing at each product generation in seek of even higher thread-level parallelism (TLP).

To manage such a large RF, a multi-banked structure is preferred, which can sustain the multi-ported RF throughput by serving concurrent operand accesses as long as there are no bank conflicts. If conflict occurs, the requests are serialized and the RF throughput is reduced leading to performance loss. As a result, in face of the RF capacity scaling in future GPGPUs, adding more banks is a natural way for the increased registers without worsening the bank conflicts. Otherwise, the performance gained from higher TLP could be compromised because more threads from schedulers will compete for an insufficient number of RF banks, aggravating the conflicts and reducing the throughput.

However, banks are not free. Adding banks requires extra column circuitry such as sensor amplifiers, pre-chargers, bitline drivers and a bundle of wires [2]. Meanwhile, the crossbar between banks and operand collectors grows significantly with the increased number of banks. They affect RF area, timing and power [3]. Because modern GPGPUs tend to be area-limited as they have already been power-limited [4], it is important to incorporate a suitable number of banks while seeking for more economic ways to mitigate the conflicts for the ever-increasing RF capacity.

In this paper, we explicitly address the issue for an efficient RF design to sustain the capacity scaling in future GPGPUs. We observe many "bubbles" in bank utilization, and apply fine-grained architectural control by stealing the idle banks to serve concurrent operand accesses, which effectively recovers the throughput loss due to bank conflicts. By explicitly leveraging the multi-warp and multi-bank features in GPGPUs, we balance the utilization on the existing RF resources instead of simply adding more banks. Experimental results demonstrate that our proposed RF with bank stealing can deliver better performance, less energy and significant area saving.

The remainder of this paper is organized as follows. Section II reviews the background and previous work on GPGPU RF. Section III introduces our motivation. Section IV proposes our techniques and architectural support. Section V and VI present the evaluation methodology and experimental results. Finally, Section VII concludes this paper.

II. Background and Previous Work

A. Baseline GPGPU and RF Architecture

A modern GPGPU consists of a scalable array of multi-threaded SMs to enable the massive TLP. Threads are often issued in a group of 32, called a warp. At any given clock cycle, a ready warp is selected and issued by one scheduler. In Nvidia Fermi GPU, each SM has 2 schedulers, 32 processing lanes with lots of execution units as in Fig. 1.

The massive TLP challenges the RF design because the RF size is quite large and keeps increasing. Meanwhile, simultaneous multiple reads/writes are required. In Nvidia PTX standard [5], each instruction can read up to 3 registers and write 1 register. Obviously, constructing such a heavily-ported (6-read and 2-write ports for dual-issue) large RF (hundreds of KB) is not feasible or economical. To solve this problem, banked RF structure has been proposed [6]. While various RF organizations are feasible, one of the most commonly used is to combine multiple single-ported banks into a monolithic RF for reduced power and design complexity. Registers associated with warps are distributed across the banks [7] , and each bank has its own

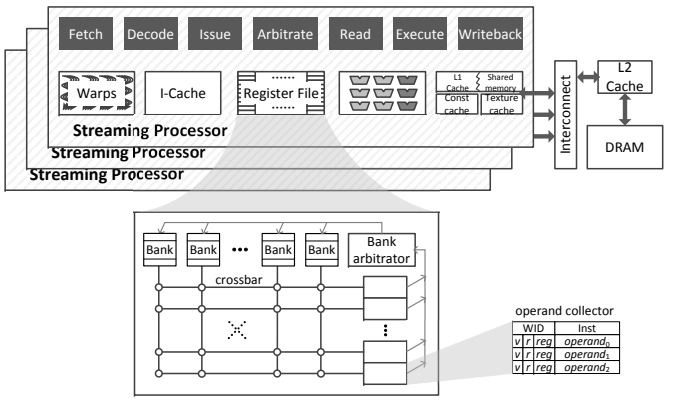


Fig. 1: Block diagram of a typical GPGPU pipeline, highlighting the banked RF with operand collectors and a crossbar unit.

decoder, sense amplifiers, etc. to operate independently.

A typical banked RF is sketched in Fig. 1. For a scheduled instruction to be dispatched, it first reserves a free operand collector. If fails, it has to wait till a collector becomes available. The arbitrator distributes RF requests from collectors to different banks but only allows a single access to the same bank at a time. Once bank conflicts occur, the requests have to be serialized. Collectors can receive multiple operands via the crossbar concurrently. After all the operands are collected, the instruction is dispatched to the execution unit and the hosting collector is released. Note that the bank conflicts directly affects the operand fetching latency that cannot be hidden by the multiple warps after the issue stage and causes issue stalls at the beginning of the execution pipeline.

B. Previous Work on GPGPU RF Design

There have been an amount of work for energy-efficient GPGPU RF designs, e.g. hybrid SRAM-DRAM [3], STT-MRAM [8] and eDRAM-based RFs [9][10]. As these new memory-based RFs compromises performance, these works focus on smart strategies to recover the performance loss.

Architectural techniques are intensively studied to improve RF performance or power. One study related to our work in CPU domain [11] proposes an area-efficient RF by reducing ports but leveraging buffer queues with pipeline augments to sustain performance. Their method is impractical on GPGPUs because there is no register renaming and the operand width is too wide for queuing. The work [12] proposes RF caching for frequently accessed operands, but the introduced storage (nearly 6KB per SM) aggravates the already congested RF layout. Another work [4] proposes a unified on-chip memory to accommodate application diversity, but the unified structure can lead to longer interconnect wires and longer access latency.

III. Motivation

A. Design Space Exploration

Conflicts have been acknowledged as a major reason for the reduced RF throughput. Adding banks can improve the throughput because registers can be spatially distributed onto more independent banks. In this section, we conduct a design space exploration to experimentally study how banking impacts the performance, area, delay and power on a pilot RF design as described in Section **5**.

Performance. We vary the number of banks with a fixed RF capacity of 128KB per SM that is typical in Fermi-like GPGPUs [6]. The performance results are averaged across all the simulated benchmarks as in Fig. 2(a). It shows that adding banks benefits the performance significantly for a small number of banks. With an increasing number of banks, the benefit diminishes as the 16-bank RF delivers almost the same performance as the 32-bank case due to the much reduced conflicts. The exploration reveals that 16 banks can be optimal for the 128KB RF.

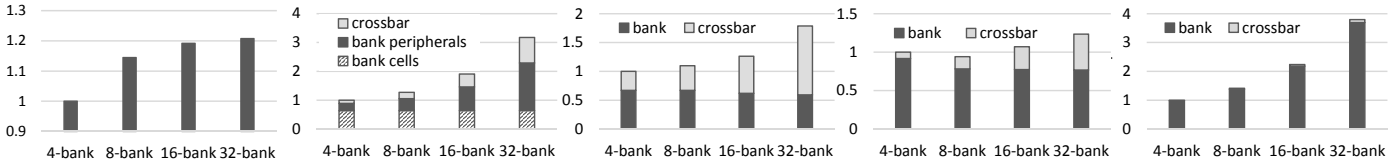
RF area. Fig. 2(b) plots the breakdown of the banking area in a 128KB RF, such as array cells, peripheral circuitry and the crossbar. It clearly shows that more banks lead to significant area overhead. The area of array cells stays constant across different schemes due to fixed RF capacity. In contrast, the area for the peripheral circuitry such as decoders, pre-chargers and equalizers, bitline multiplexers, sense amplifiers and output drivers, expands notably with an increasing number of banks. This is because a GPGPU register of 128B is much wider than that in CPU. This significantly increases the number of bitlines and widens the data buses. Each bank duplicates the peripheral circuitry for independent array control, resulting in even larger area expansion compared to conventional CPUs [13].

Meanwhile, the crossbar dimension between RF banks and collectors grows significantly with the number of banks. For example, nearly 50% additional area is observed from a 16-bank to a 32-bank RF, although these two designs deliver quite similar performance as shown in Fig. 2(a).

RF timing. Because we fix the RF capacity, more banks naturally result in shorter access time in the memory array as shown in Fig. 2(c). However, array access time is just a portion of the total RF delay and the advantage is compromised by the wire transfer time via a larger crossbar. With an increasing number of banks, the crossbar delay tends to dominate. However, even the worst-case scheme studied in this paper can meet the sub-1GHz GPGPU frequency, so we do not consider their timing differences.

RF power. Fig. 2(d) shows the dynamic bank power. The cell array power slightly decreases with more banks because the bitlines are shorter under fixed RF capacity. In contrast, the dynamic power on data transfers via the crossbar increases considerably due to the larger RF area. Meanwhile, the leakage power increases notably as in Fig. 2(e) because banks duplicate the peripheral circuitry.

Summary. The design space exploration reveals the fact that fewer banks benefits the RF physical design such as area, timing and power. However, simply reducing the number of banks is not acceptable from the performance point of view. For example, there is around 5% performance loss from a 16-bank to an 8-bank design as shown in Fig. 2(a), while our study shows that there is still plenty of room for performance improvement even above the 16-bank, if the conflicts can be effectively eliminated.



(a) Normalized performance (b) Normalized area (c) Normalized delay (d) Normalized dyn. power (e) Normalized leakage power

Fig. 2: Design space exploration with a varying number of banks.

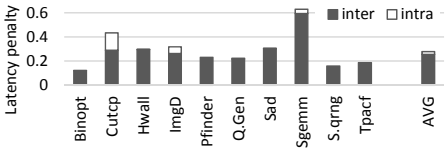


Fig. 3: Latency penalty due to bank conflicts (16-bank RF).

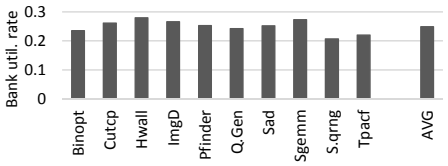


Fig. 4: Utilization rates for different benchmarks (16-bank RF).

B. Causes for Bank Conflicts

Without conflicts, the operands should pass through the RF with one cycle latency offering maximum throughput. Upon a bank conflict, only one operand can proceed on a bank, causing latency penalty to other competing operands. To quantify the penalty on the average RF access latency, we conduct simulation in Fig. 3, where higher penalty means longer operand latency that results in reduced RF throughput and more issue stalls. From this figure, we see that the inter-instruction conflicts are quite common, resulting in an extra 0.1-0.6 cycle latency and directly cause performance loss. There is another type of conflicts, namely intra-instruction conflicts, come from the competing operands in the same instruction. However, they rarely happen and can be effectively eliminated by the compiler.

C. RF Utilization Rate

Even for the optimal 16-bank design in Fig. 2(a), we still observe under-utilization in the RF resources. Fig. 4 gives the average utilization rates of all the RF banks. Averagely, the banks are accessed only 25% of the total execution time. This result can be explained by runtime factors such as data and control hazards, or misses on memory units that might incur waiting cycles in RF. Unbalanced operand loading of PTX instructions is another factor. Not all instructions fully use the 3 source operands, leading to unbalanced stress on RF resources. In addition, the ideal case of accessing all the banks at the same cycle hardly sustains in real programs. Most of the time, operands are not evenly distributed across banks, leaving some of banks free of accesses. A balanced utilization manifests as an intelligent operand coordination to eliminate the bank bubbles and accelerate the operand fetching during execution.

IV. Proposed Technique

A. Overview

In our work, we propose a bank stealing technique for a balanced RF utilization in front of the conflicting accesses from multiple instructions. It opportunistically "steals" an unused bank at the current cycle for an operand supposed to be fetched in the next cycle to avoid a potential upcoming conflict. In fact, unlike CPU, GPGPU pipeline is likely to have multiple ready warps waiting at the issue stage (14 out of 48 warps as observed in our simulation) with operands ready, which enables a wise operand coordination especially on the under-utilized bank resources. This technique adequately exploits the multi-warp, multi-bank feature essential in GPGPU and therefore is broadly applicable to a wide range of GPU architectures.

B. Bank Stealing for Inter-instruction Conflicts

B.1 Basic Idea

Conventionally upon a conflict, one of the register reads is deferred causing pipeline stall. However, if it can be moved to the previous cycle and steal the bank when vacant, the conflict and pipeline stall can be eliminated.

As illustrated in Fig. 5, at time T, warp W_a and W_b are issued, and their operands r_a and r_b are granted to read the RF by passing the conflict checking at the arbitration stage at $T+1$. Also at $T+1$, W_c and W_d are issued, and their operands r_c and r_d are arbitrated to be conflicting at $T+2$. Conventionally, either r_c or r_d has to be delayed causing serialized reads at $T+3$ and $T+4$. Differently, in our scheme, the read of r_d can be moved one cycle earlier if it is not conflicting with r_a and r_b by passing conflict checking at $T+1$. In that case, W_c and W_d can finish operand reading at $T+2$ and $T+3$ without pipeline stalls, saving one cycle penalty. However if r_d conflicts with r_a or r_b at $T+1$, the stealing is aborted and the pipeline behaves as before. In fact, the bank stealing increases the opportunity for read success by stealing the unused RF bandwidth in the previous cycle.

B.2 Warp Scheduling for Read Stealing

To enable the read stealing, we need to identify the warps (W_c, W_d) to be issued in the next cycle at the arbitration stage $(T+1)$ of the current warps (W_a, W_b), and check for the bank conflicts of operands (r_a, r_b, r_d) between the current and next warp instructions. The question is how to identify the candidate warps to be issued one cycle ahead for bank stealing, so that we can put them into the

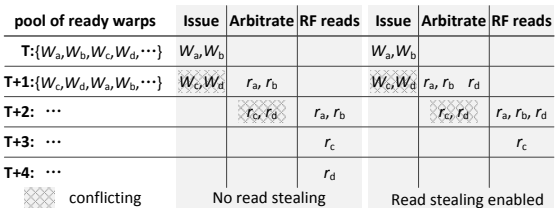


Fig. 5: Operand read stealing for conflict removal.

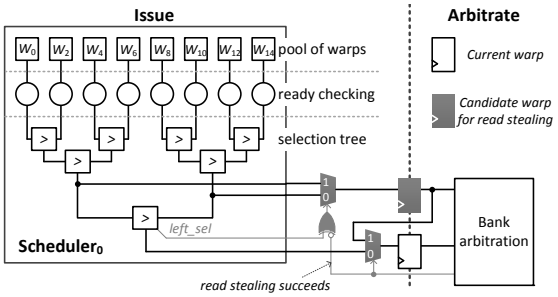


Fig. 6: Warp selection in read stealing using default scheduler.

bank conflict checking performed in the current warp instructions' arbitration stage.

In fact, this is feasible due to the warp organization featured in GPGPU. It is very common that multiple warps are ready and waiting for issue in the instruction pool. The selection of candidate warp can be done with the aid of the default warp scheduler. As in Fig. 6, a typical scheduler uses a hierarchical tree to select and wake up the warp with the highest priority for issue. At one stage above the bottom level of the tree, there are two warps. In our scheme, we always treat the other warp that is not selected for issue at the current cycle as the candidate warp for stealing. The operands of the current and candidate warps will be latched into the arbitration stage for conflict checking. Upon conflicts, the operands can be read ahead if two conditions are met: 1) There must be a free collector to hold the stolen operands; 2) The stealing should resolve the conflicts in the next cycle and not cause conflict in the current cycle. Note that the scheduler should be overridden in the next cycle to prioritize issuing the candidate warp if their operands have been read ahead, as achieved by the shaded multiplexors shown in Fig. 6.

The stealing might modify the default instruction flow from the original scheduler, but the impact is minimal. In our experiments, we find that although the instruction flow may deviate, the occasional overriding of the default scheduler turns out to have little impact on the program behavior and performance. In fact, because the candidate warps are chosen from the second to the bottom level of the selection tree, they are likely to be scheduled for the next cycle by the default scheduler anyway. Therefore, it is lightweight and compilable to the schedulers like GTO, round-robin, and other common schedulers [14][15].

The essence of stealing lies in the fact that RF banks are under-utilized as in Fig. 4. By operand coordination across multi-warps, it leverages the idle banks at current cycle and opportunistically fills them with predicted future

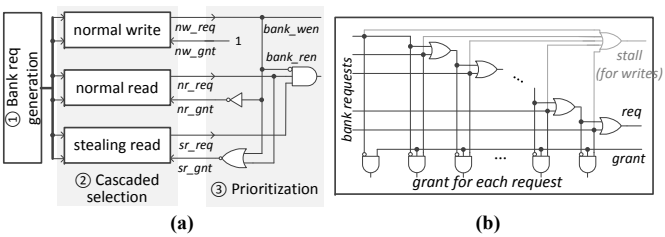


Fig. 7: Hardware support for read stealing in the arbitrator, (a) overview, (b) cascaded selection. Note that the arbitration on normal reads forms the critical path.

TABLE I: Key parameter settings in the GPGPU-Sim simulator

Parameters	Setting	Parameters	Setting
# cores (SMs)	15	SM Freq. (MHz)	700
warp size	32	# threads/SM	1,536
# active CTAs/SM	8	# registers/SM	32,768
# collecters/SM	10	# schedulers/SM	2

reads. Note that read stealing does not reduce the total number of reads. It just balances the utilization on the available RF bandwidth.

C. Arbitration for Bank Stealing

Finally, the arbitrator has to be modified to enable the bank stealing. Fig. 7 (a) shows the arbitration chain to generate the read request to a bank. At first, all operand requests are decoded into bank requests as in ①. Then, the requests are passed through a set of cascaded selection in ②, where the bank requesting signal will be generated from each type of accesses such as "normal read" or "stealing read" as shown in Fig. 7 (b). Finally, the priority logics in ③ will grant the ultimate access respecting the normal read first.

We design and synthesize the arbitration circuit and find it to be less than 0.5% of the RF area. In terms of timing, the original critical path, consisting of many normal reads to be arbitrated in the cascaded selection tree, is generally long enough to overlap the delay of stealing read. The only added delay on the critical path is the additional input from stealing read on the OR gate in ③. We simulate and prove that tighter delay constraints and gate sizing can easily absorb its delay to the sub-1GHz nominal frequency.

V. Evaluation Methodology

In our experiment, we evaluate the RF designs with different number of banks, regarding their performance, area and energy efficiency using our proposed bank stealing. For the performance measurement, we use the cycle level architectural simulator GPGPU-Sim v3.2 [16]. The key simulator configurations are shown in Table I, which generally conform to the Fermi style architecture.

Table II shows the benchmarks used in our evaluation. We investigate 10 benchmarks from a wide range of applications in CUDA SDK [17], Rodinia [18] and Parboil [19] suites. We use CUDA SDK 4.2 for compilation and the simulator is configured to use PTXPlus that provides more accurate simulation results as reported in [16].

TABLE II: Benchmark characteristics

Name	Abbr.	From	#Inst	#regs	Name	Abbr.	From	#Inst	#regs
binomialOptions	Binopt	SDK	275M	20	Cutcp	Cutcp	Parboil	122M	25
heartwall	Hwall	Rodinia	15M	28	imageDenoising	ImgD	SDK	120M	59
pathfinder	Pfinder	Rodinia	277M	13	quasirandomGenerator	Q.Gen	SDK	271M	15
SAD	Sad	Parboil	165M	15	sgemm	Sgemm	Parboil	195M	45
SobolQRNG	S.qrng	SDK	19M	14	tpacf	Tpacf	Parboil	2940M	28

TABLE III: Area, timing and power primitives (128KB RF)

Structures	Dyna.(nJ)	Leak.(nJ)	Area(mm^2)	Delay(ns)
16-bank	0.185	384.4	0.565	0.31
Crossbar	0.071	11.2	0.225	0.31

To learn the RF delay and power, we synthesis a pilot RF with multiple single-ported banks by using SMIC commercial memory compiler tool [20] with industrial cell library. We build the RF by varying the number of banks with the area, power and delay number reported from the memory compiler. We also evaluate the crossbar using H-SPICE. Detailed primitives are shown in Table III. We collect RF statistics from GPGPU-Sim and use these primitives to compute the RF power, including the normal bank accessing, crossbar transmission and bank stealing if enabled. For all other non-RF chip-wide components, we use GPUWattch [21] to calculate the power.

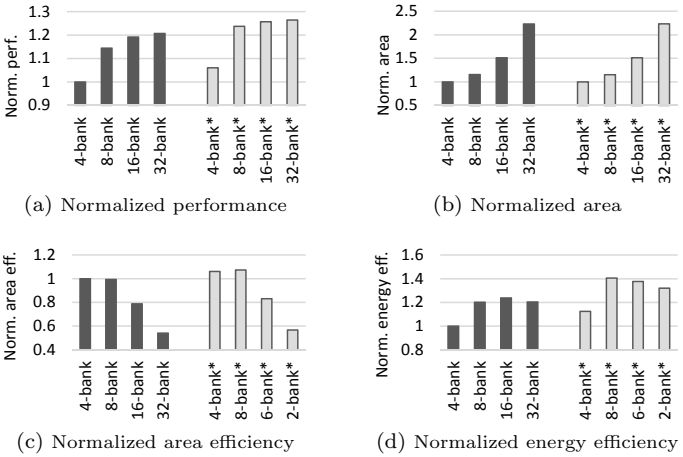
VI. Experimental Results

In our experiments, we first show the overall performance improvement by the proposed RF on different banking, and analyze representative designs in details. We use *straight RF* and *proposed RF* to denote the traditional RF and the RF equipped with the proposed read stealing technique.

A. Design Space Study

Fig. 8 studies the straight and proposed RFs with a fixed capacity of 128KB but different numbers of banks (4, 8, 16, 32) and a crossbar unit accordingly. In Fig. 8 (a, b), we see that the proposed RFs always outperform the straight RFs with the same bank, while having almost the same area due to the negligible hardware cost for bank stealing. The overall performance gain ranges from 6% to 10%. Given the fact that we only make light changes on a single GPGPU unit, this improvement is significant. Another observation is that the proposed RFs often outperform the straight RFs with more banks. For instance, the proposed 8-bank RF outperforms the straight 16- or 32-bank by eliminating the potential bank conflicts. Fig. 8 (c, d) also demonstrate the superiority of our proposed RFs in terms of area and energy efficiency, respectively, where the area efficiency is calculated as normalized performance over area, while the energy efficiency is normalized performance over energy.

In summary, the proposed RF is much more area-efficient, e.g. nearly 25% and 50% area savings over the 16- and 32-bank straight RFs. At the same time, the performance is also superior with 3.8% and 2.5% improvement over the 16- and 32-bank straight RFs. Because the RF size keeps increasing, the much improved area efficiency with better performance will be beneficial for the future scaling.



(a) Normalized performance (b) Normalized area

(c) Normalized area efficiency (d) Normalized energy efficiency

Fig. 8: RF design space study using different banks. Proposed RFs are marked with "*".

B. Case study

As a case study, we choose the proposed 8-bank RF that exhibits the highest area efficiency. For comparison, we choose 8- and 16-bank straight RFs. The straight 16-bank RF acts as the baseline and all the results are normalized to it. Detailed results are shown in Fig. 9. From the bottom bar, we see that the straight 8-bank RF degrades the performance by nearly 5% compared to the straight 16-bank RF mainly due to the increased bank conflicts.

Performance. In Fig. 9, the average performance gets improved by around 8% over the straight 8-bank RF by read stealing. That means a better performance of 3.8% than the straight 16-bank RF. The results confirm that read stealing can effectively reduce the potential conflicts and improve the overall performance. Greater performance gains could be expected on more RF bounded codes.

It is worth mentioning that we have evaluated different types of warp schedulers including round-robin, two-level and etc. The conclusion stays the same as using GTO. Although read stealing may occasionally modify the instruction flow, the experiments verify it to be generally applicable and not sensitive to the host scheduling policies.

RF access latency. Fig. 10 shows the latency penalty measured in pipeline cycles on RF accessing. Compared with Fig. 3(a) on straight 16-bank RF, the straight 8-bank RF nearly doubles the latency penalty due to aggravated conflicts. Instead, the proposed 8-bank RF effectively shortens the operand latency by around 70%. It also detects part of the intra conflicts and eliminates them as well.

We also notice an increasing utilization rate of 55% in

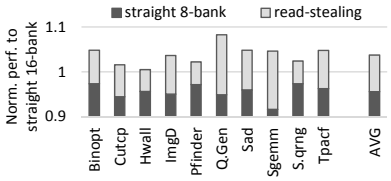


Fig. 9: Performance comparison on the RF designs w/o and w/ stealing.

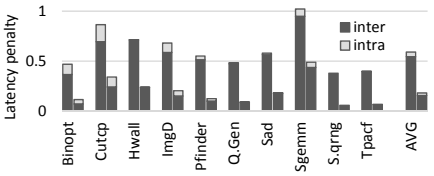


Fig. 10: Latency penalty on straight (left) and proposed (right) RF with 8-bank.

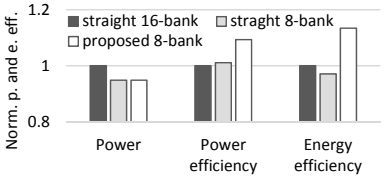


Fig. 11: Normalized power and energy efficiency.

the proposed 8-bank RF with read stealing against 25% and 45% in the straight 16- and 8-bank RFs, respectively. This is mainly because the greatly reduced number of banks and more balanced usage of the RF.

Power and energy efficiency. We show the power number and use the metric $perf/power$ to quantify the power efficiency and $perf/energy$ for the energy efficiency. The power results reported are chip-wide power counting in all the GPGPU core units, and L1/L2 caches and etc.

Fig. 11 gives the results. The straight 8-bank RF consumes 5% less power than the straight 16-bank RF. This is because fewer banks reduce the crossbar power and the leakage despite the slightly increased bank dynamic power as explored in Fig. 2. However, straight designs using fewer banks are inferior when performance is taken into account. In contrast, our proposed RF is more energy efficient. Although there is additional arbitration logic, the power overhead is so small and negligible. When counting in the performance, the energy efficiency of our proposed design can beat the straight 16-bank RF by 15%, owning to the superior performance gains.

VII. Conclusions

In this study, we propose an efficient RF design with conflict elimination to sustain the high RF throughput facing ever-increasing TLP. Our read stealing technique explicitly exploits the available bandwidth in the highly banked RF structure to eliminate conflicts and leverages the massive parallelism in GPGPU to access the RF more intelligently. It improves the performance using a half or even quarter number of banks with smaller area and energy budget, which implies that the proposed RF can sustain the fast-increasing capacity of RF scaling in future GPGPUs.

Acknowledgments

This work is partly supported by the Natural Science Foundation of China (Grant No. 61402285, No. 61202026 and No. 61332001), China Postdoctoral Science Foundation (Grant No. 2013M540362 and No. 2014T70418) and Program of China National 1000 Young Talent Plan.

References

[1] NVIDIA Whitepaper. Nvidia's next generation CUDA compute architecture: Kepler GK110.

[2] Xiaoyao Liang, Kerem Turgay and David Brooks. Architectural power models for SRAM and CAM structures based on hybrid analytical/empirical techniques. In *Proceedings of the International Conference on Computer-Aided Design*, 2007.

[3] Wing-kei S. Yu, Ruirui Huang, Sarah Q. Xu, Sung-En Wang, Edwin Kan, and G. Edward Suh. SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading. In *Proceedings of the 38th International Symposium on Computer Architecture*, 2011.

[4] Mark Gebhart, Stephen W. Keckler, Brucek Khailany, Ronny Krashinsky, and William J. Dally. Unifying primary cache, scratch, and register file memories in a throughput processor. In *Proceedings of the 45th International Symposium on Microarchitecture*, pages 96–106, 2012.

[5] NVIDIA Whitepaper. Parallel thread execution isa version 3.0.

[6] NVIDIA Whitepaper. NVIDIA's Next Generation CUDA Compute Architecture: Fermi.

[7] Mohammad Abdel-Majeed and Murali Annavaram. Warped register file: A power efficient register file for GPGPUs. In *Proceedings of the 19th HPCA*, pages 412–423, 2013.

[8] Bingyi Cao Nilanjan Goswami and Tao Li. Power-performance co-optimization of throughput core architecture using resistive memory. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, 2013.

[9] Naifeng Jing, Yao Shen, Yao Lu, Shrikanth Ganapathy, Zhigang Mao, Minyi Guo, Ramon Canal, and Xiaoyao Liang. An energy-efficient and scalable eDRAM-based register file architecture for GPGPU. In *Proceedings of the 40th International Symposium on Computer Architecture*, 2013.

[10] Naifeng Jing, Haopeng Liu, Yao Lu, and Xiaoyao Liang. Compiler assisted dynamic register file in gpgpu. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2013.

[11] Nam Sung Kim and Trevor Mudge. Reducing register ports using delayed write-back queues and operand pre-fetch. In *Proceedings of the 17th ICS*, pages 172–182, 2003.

[12] Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, William J. Dally, Erik Lindholm, and Kevin Skadron. Energy-efficient mechanisms for managing thread context in throughput processors. In *Proceedings of the 38th International Symposium on Computer Architecture*, pages 235–246, 2011.

[13] Jessica H. Tseng and Krste Asanović. Banked multiported register files for high-frequency superscalar microprocessors. In *Proceedings of the 30th International Symposium on Computer Architecture*, 2003.

[14] Veynu Narasiman, Michael Shebanow, Chang Joo Lee, Rustam Miftakhutdinov, Onur Mutlu, and Yale N. Patt. Improving GPU performance via large warps and two-level warp scheduling. In *Proceedings of the 44th International Symposium on Microarchitecture*, 2011.

[15] Timothy G. Rogers, Mike OConnor, and Tor M. Aamodt. Cache-conscious wavefront scheduling. In *Proceedings of the 45th International Symposium on Microarchitecture*, December 2012.

[16] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009.

[17] NVIDIA. `https://developer.nvidia.com/cuda-toolkit`.

[18] Shuai Che, Michael Boyer, and etc. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the International Symposium on Workload Characterization*, 2009.

[19] John A Stratton, Christopher Rodrigues, and etc. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.

[20] Semiconductor Manufacturing International Corporation. Register-file user guide, 2012.

[21] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. GPUWattch: Enabling energy optimizations in GPGPUs. In *Proceedings of the 40th International Symposium on Computer Architecture*, 2013.