JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 15, No. 4, December 2021, pp. 135-147

Reducing GPU Energy Consumption by Packing Narrow-Width Operands

Xin Wang
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
wangx44@vcu.edu

Wei Zhang*
Department of Computer Science and Engineering, University of Louisville, Louisville, KY, USA
wei.zhang@louisville.edu

Abstract
In this paper, we study the use of an Operand-Width-Aware Register (OWAR) packing mechanism for graphics processing unit (GPU) energy saving. In order to efficiently use the GPU register file (RF), OWAR employs a power gating method to shut down unused register sub-arrays in order to reduce dynamic and leakage energy consumption of RF. As the number of register accesses was reduced due to the packing of the narrow width operands, the dynamic energy dissipation was further decreased. Finally, with the help of RF usage optimized by register packing, OWAR allowed GPUs to support more thread-level parallelism (TLP) through assigning additional thread blocks on streaming multiprocessors (SMs) for general-purpose GPU (GPGPU) applications that suffered from the deficiency of register resources. The extra TLP opens opportunities for hiding more memory latencies and thus reducing the overall execution time, which can lower the overall energy consumption. We evaluated OWAR using a set of representative GPU benchmarks. The experimental results showed that compared to the baseline without optimization, OWAR can reduce the GPGPU's total energy up to 29.6% and 9.5% on average. In addition, OWAR achieved performance improvement up to 1.97X and 1.18X on average.

Category: Compilers / Programming Languages

Keywords: Graphics processing units (GPUs); Register file; Leakage energy; Narrow-width operands

I. INTRODUCTION

Taking advantage of the ability to process multiple data in parallel, graphics processing units (GPUs) have been increasingly used to accelerate general purpose applications like compute-intensive data-parallel scientific computing programs [1-3]. General-purpose GPUs (GPGPUs) execute hundreds or even thousands of threads concurrently to ensure high throughput. To support a massive number of simultaneous threads and rapidly context switching between these threads, a huge number of compute units and a large register file (RF) are inevitable. GPU design trend shows that GPU performance improvement continues to rely on increasing the hardware resources and operating them at higher frequencies to accommodate even more active threads. Table 1 shows hardware configurations

Open Access http://dx.doi.org/10.5626/JCSE.2021.15.4.135 http://jcse.kiise.org

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/4.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 19 August 2021; **Accepted** 13 October 2021
*Corresponding Author

Copyright © 2021. The Korean Institute of Information Scientists and Engineers pISSN: 1976-4677 eISSN: 2093-8020

Table 1. Number of SIMD units and size of RF in NVIDIA GPUs

	RF per SM (kB)	# of SIMD units per SM	# of SMs	Total RF (kB)	Total SIMD units
Tesla G80 [4]	32	8	16	512	128
Tesla GT200 [4]	64	8	30	1920	240
Fermi GF100 [5]	128	32	16	2048	512
Kepler GK110 [6]	256	192	15	3840	2880
Kepler GK210 [6]	512	192	15	7680	2880

including number of single instruction multiple data (SIMD) execution units and the size of RF in recent GPUs [4-6]. The size of RF has increased significantly in recent NVIDIA GPU generations. A similar trend is also found in the number of SIMD computing units. Unfortunately, persistently accumulating hardware resources negatively exposes GPUs under the huge pressure of power consumption and chip area [7-9]. In particular, RF with the design of massive high leakage transistors has been demonstrated to contribute significantly to GPU's total energy consumption. Thus, we need to explore new techniques to exploit and manage GPU registers more efficiently. This paper exploits the narrow-width operands to reduce the GPU energy dissipation by 9.5%.

The narrow-width operands packing technique has been fully studied and approved to be a promising solution to reduce register pressure on CPUs [10-13]. Several RF optimization methods have been proposed by considering the narrow-width operands [11-15]. Brooks and Martonosi [10] proposed hardware mechanisms for general purpose microprocessors that dynamically recognized and capitalized on narrow-width operands and introduced both power-oriented and performance-oriented optimizations. Ergin et al. [11] extended Brooks and Martonosi [10]'s work to reduce the pressure on the RF in super-scalar processors. They proposed two schemes to dynamically packing multiple narrow-width results into partitions within a single register. These techniques which were proven to be helpful for RF management in CPUs, however, cannot be easily ported to GPUs due to the essential differences of RF's functionality and microarchitecture between CPU and GPU.

To the best of our knowledge, existing researches on GPU RF management do not make the most of the opportunity offered by narrow-width operands for a promotion of energy efficient and performance. Several studies have introduced similar techniques [16-19]. Observing that the register values of threads within the same warp are similar, Lee et al. [16] presented Warped-Compression, a warp-level register compression scheme which removes data redundancy of register values through register compression to enable power reduction opportunities. Tan et al. [17] proposed the narrow-width-aware register write back method which combines two narrow-width writes to share data bus resource and hence enhance the performance. The main purpose of the research was to leverage resistive memory to enhance the soft-error robustness and reduce the power consumption of registers in GPUs, while they used the narrow-width-aware register write back method only to counteract the performance loss from the long write latency of STT-RAM register file. Gilani et al. [18] noticed that many operands require considerably fewer bits for accurate representation and computations. They proposed a sliced GPU architecture that improves the performance of the GPU by dual-issuing instructions to two 16-bit execution slices.

The main aim of this paper is to propose a GPU RF management mechanism that attains energy efficiency by taking advantage of narrow-width operands. Inspired by the narrow-width-aware register packing research on CPUs, we designed a GPU register packing scheme called OWAR (operand-width-aware register) which took into account the characteristics of GPU's microarchitecture and RF organization. This novel technique first applied the register usage prediction methods proposed in [20] to dynamically identify narrow-width operands. Multiple narrow-width operands were then packed into a single register to improve energy efficiency and/or performance. A method was also developed to predict the shrunken register usage for the incoming thread blocks. By register packing and usage prediction, the remaining RF space increased and the pressure of future thread blocks on register resources decreased. As a result, more registers became available. OWAR was then turn off the sub-arrays that contained unused registers to save both dynamic and leakage energy. Combined with a renaming table, OWAR was able to map multiple architectural registers that stored narrow-width operands into a single physical register at run time. Consequently, OWAR offered the illusion that extra RF space was available, making it possible for streaming multiprocessors (SMs) to host more thread blocks. For GPU kernels whose occupancy of threads was limited by RF resources, OWAR can provide additional thread-level parallelism (TLP) to further boost performance and reduce energy consumption.

This paper is an extended version of a prior conference paper [21]. The major extensions in this paper include the following: (1) we studied the use of a coarser-grain power gating technique for unused GPU registers; (2) we evaluated the overheads of the proposed OWAR packing approach and found that the hardware cost accounts for 3% of the total GPU RF, which is insignificant; and (3) we showed the reduction of RF accesses with our method, and we found that on average, the total number of RF bank accesses was reduced by 13.1% for OWAR-PG and OWAR-TO-PG, indicating good opportunities for dynamic energy reduction for register banks.

II. BACKGROUND OVERVIEW

A. CUDA Programming Model

CUDA programming language allows the programmer to define C functions as several kernels which consist of thousands of threads that are executing in parallel [22]. Each thread within a CUDA kernel is marked with an assigned unique thread ID which is accessible through the built-in threadIdx variable. GPGPU applications always contain multiple kernels organized as a group of thread blocks. A thread block is formed by one-dimensional, two-dimensional or three-dimensional thread index allowing a vector, matrix or volume computation domain. Thirty-two threads within the same thread block with consecutive thread IDs are grouped as a warp. A warp is executed in a single instruction multiple threads (SIMT) way and has only one PC. However, threads in the same warp can access different memory address and follow different control flow paths. The warps in a thread block allow GPU to overlap long latency by conserving the context of the stalled warps and switching to the oldest ready warp. Each warp is assigned with a set of dedicated architectural registers that are one-to-one mapped to corresponding physical registers.

B. Baseline GPU RF Architecture

Modern GPUs are equipped with a huge RF in order to support massive TLP to maximize computation throughput. For example, NVIDIA GTX480 GPUs [5] feature 128 kB multi-banked SRAM RF per SM. To our knowledge, there is no official documentation on GPU RF organization that is publicly available. Thus, we take the detailed RF structure described in [23] as our baseline, which has been proven an efficient organization for GPU RF (Fig. 1). For each SM, the whole 128 kB RF is partitioned into 32 banks with 4 kB register per bank. The multi-banked design is employed to increase GPU RF bandwidth. Each bank is dual-ported (one read port and one write port) with 256 128-bit wide entries. All registers for threads within a warp reside in consecutive banks, while accessing to a single bank entry can fetch only four 32-bit register values and reading an operand for a warp instruction needs to access up to eight banks. Although writing and reading the same bank can happen concurrently, multiple accesses to the same bank lead to bank conflicts. Therefore, a unit named operand collector is applied to buffer the source operands of the instruction. The RF and the operand collectors are connected via a crossbar network. To support our register packing technique, we modified the baseline RF organization to 32 banks with 128 256-bit wide entries for each bank and the reason will be described in Section IV.

III. MOTIVATION

RF packing: The narrow-width operands packing techniques have been studied and employed on CPUs. Observing that many operands called narrow-width operands have fewer significant bits compared to the full

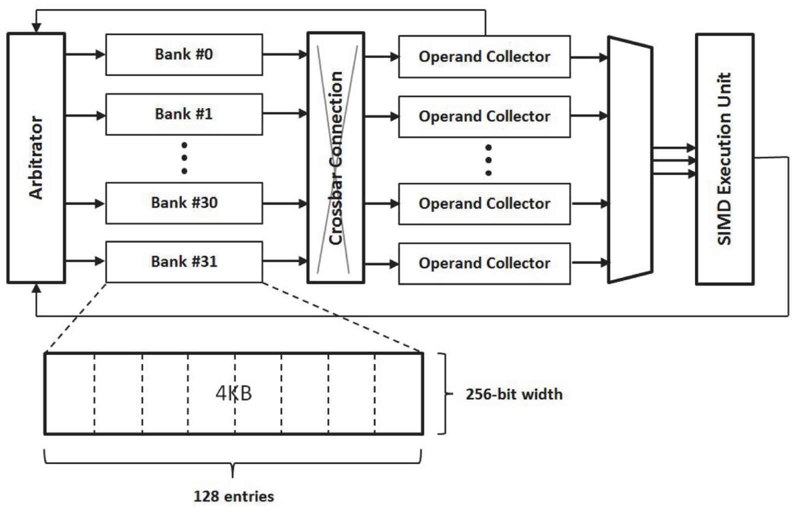


Fig. 1. Baseline RF architecture. Adapted from [20] with permission of IEEE.

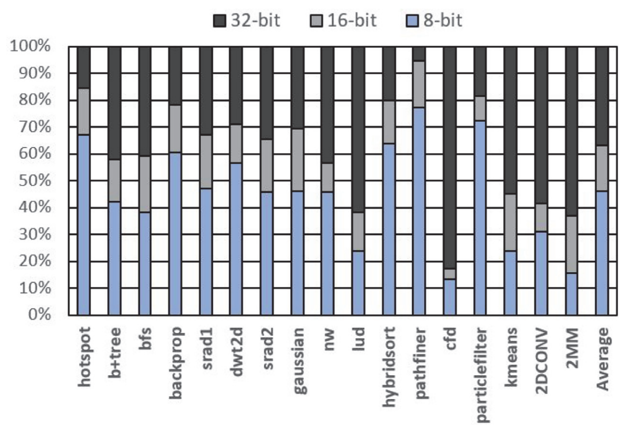


Fig. 2. Width distribution.

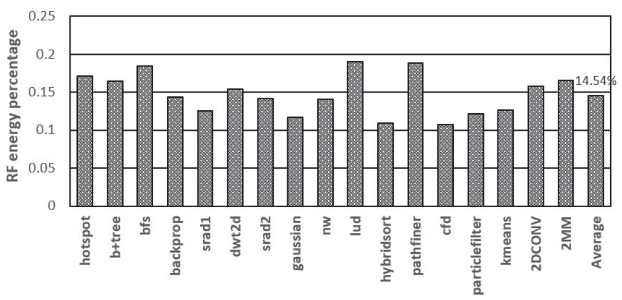


Fig. 3. The fraction of RF energy consumption compared to the total GPU energy.

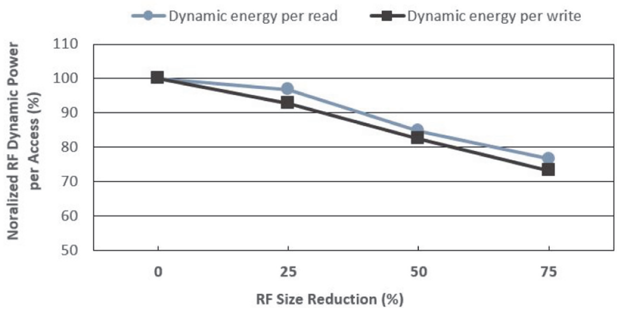


Fig. 4. RF dynamic energy savings with different RF size.

width of a 32-bit register, several works [10-12, 14, 15] proposed to detect them and merge multiple narrow-width operands into a single full-sized register. The register packing techniques were then designed to save power consumption [12] and improved performance [11, 12] as well as register file reliability [14, 15]. Unlike CPUs which primarily counted on caches to reduce memory latency, GPUs can exploit massive TLP to hide memory latency and increase throughput. To support fast and low-cost context switching for many concurrently running threads, a large number of registers were necessary. Therefore, compared to CPUs, GPUs were much thirstier for RF resources. We evaluated the operand width characteristics across 17 GPU benchmarks and classified the operand width into three level: 8-bit, 16-bit, and 32-bit. Fig. 2 shows the width distribution of values written into registers. On average, 45.3% of all values consume only 8-bits of a full 32-bit register, 16.1% of all values can be represented by only 16 bits and only the rest 38.6% need a full-sized register. Obviously, using 32-bit registers to conserve operands with varied significance bits can be a waste of precious RF resources. The evaluation results presented an opportunity to save plenty of RF space by packing narrow-width operands into a single 32-bit register. Although narrow-width packing techniques show advantages on CPUs, the effectiveness was still unknown on GPUs due to the fundamental architectural differences between CPU and GPU. In this paper, we employed a GPU register packing method called two-level (thread-level and warp-level) narrow-width packing proposed in [20]. The thread-level packing is to select the most economic number of bits which is sufficient to store each thread register belong to the same warp without any accuracy loss. The warp-level packing maps all registers of a warp to consecutive RF space.

Energy savings: The GPU RF is demonstrated to be responsible for a large fraction of GPU's total energy consumption. We evaluate the dynamic and leakage energy consumptions of GPU RF for 17 benchmarks and compare them with the GPU's total energy consumptions. We find that 6.5% of dynamic energy and 20.9% of leakage energy is consumed by RF, respectively. Fig. 3 shows that the GPU RF contributes 14.5% to the GPU's total energy. By shrinking the RF size, the RF leakage energy consumption will decrease due to the reduction of the number of leaking transistors. In addition, a smaller RF size also leads to lower dynamic energy consumption per access. We have evaluated the dynamic energy savings due to the reduction of the RF size. The results represented in Fig. 4 tell that for a halved RF, the dynamic energy consumption of one RF read and one RF write is reduced by 15.2% and 17.5% respectively. All these results are obtained by using GPUWattch [8] starting with the baseline described in Section II. The results inspire us to reduce dynamic and leakage energy through scaling down RF capacity. Furthermore, due to register packing, a fewer number of RF accesses also lead to further reduction of the dynamic energy consumption.

IV. OPERAND-WIDTH-AWARE REGISTER PACKING MECHANISM

The narrow-width operands packing technique shows its effectiveness of improving energy efficiency and performance on CPUs. Unfortunately, this promising mechanism cannot friendly adapt to GPUs as a result of the distinction between GPU and CPU architectures. In

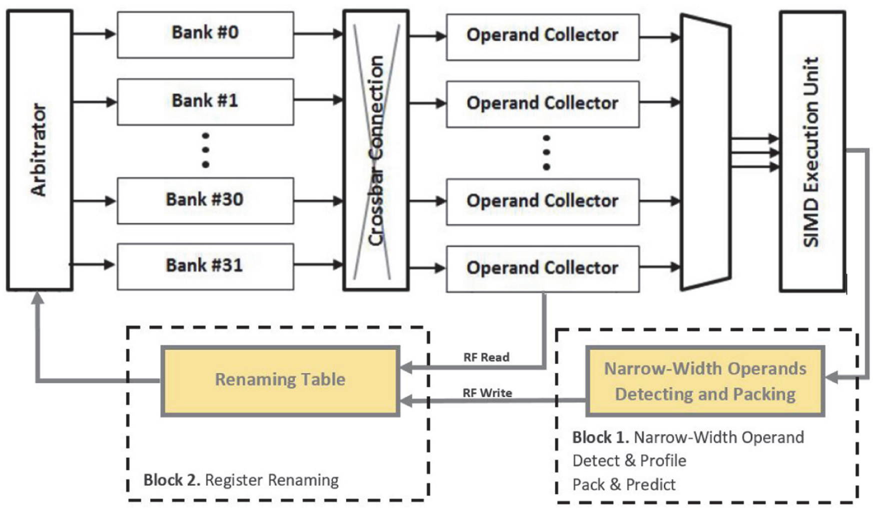


Fig. 5. Overview of OWAR. At write back stage, narrow-width operands are detected, profiled and packed. A shrunken register usage is predicted to guide the scheduling of future thread blocks (Block 1). A renaming table is featured for register reallocation (Block 2). Adapted from [20] with permission of IEEE.

this work, we proposed OWAR, a novel GPU RF management approach supported by affordable hardware to improve the energy efficiency of the RF and the GPU.

A. Overview

Fig. 5 illustrates an overview of OWAR. Block 1 (Section IV-B) involves a two-step register packing mechanism. The first step was to dynamically detect narrow-width operands and profile operand-width boundaries at write back stage. The second step acts a two-level register packing and makes a prediction about future register usage. Detailed descriptions about Block 1 can be found in [20]. Block 2 (Section IV-D) is the renaming table-based register reallocation mechanism that dynamically maps architectural registers to physical registers to achieve RF resource saving.

B. Detecting and Packing Narrow-Width-Operands

Detect and profile: The narrow-width operands are identified at the write-back stage using 32 zero-detection logic units [10]. Each unit inspected the length of one operand from a single thread and outputs a 2-bit operand width indicator. Specifically, "01" represents an 8-bit operand, "10" represents a 16-bit operand, and "11" represents a 32-bit operand. All the outputs from the 32 zero-detection logic units were then compared to find the upper bound width of 32 values which can guarantee sufficient bits for all operands. The upper bound width was forwarded to the thread-level packing table (TLPT) with register index in the form of profiling information.

Pack and predict: A TLPT was added to each SM to store all narrow-width information provided at the narrow-width detecting stage. When the TLPT received an operand width from the narrow-width detecting logic, it makes a comparison between the newcomer and the current maintained operand-width and it only keeps the width that was wider. In NVIDIA Fermi architecture [5], each thread can use up to 63 registers. In this case, the TLPT has 63 entries and each entry was indexed by a registered id and holds 2 bits data indicating the bit width of the value stored in the corresponding register. The total TLPT size per SM was calculated by Eq. (1):

$$\textit{TLPT Size} = (\#\textit{entry} \times \#\textit{bit per entry}) = 63 \times 2 \text{ bits} = 126 \text{ bits} \quad (1)$$

At the beginning of application execution, the TLPT was empty and it was filled dynamically during run time. The TLPT kept itself updated with the widest width (i.e., upper bound) fed by the narrow-width detecting logic and profiler. Whenever all threads in a thread block were completed, the prediction on register usage can be made by adding up the sizes of total registers according to the narrow-width status held in the TLPT. Then register usage prediction was sent to the thread block scheduler to loosen the register file constraint for future thread block scheduling. A misprediction happens when the TLPT found that the maintained width of a register was insufficient to meet the requirement of the current computed result. A mechanism to deal with misprediction, though very rare, will be introduced in Section IV-C.

C. Renaming Table

The conventional use of register renaming on CPUs was to eliminate the false data dependencies caused by reusing of architectural registers among several contiguous

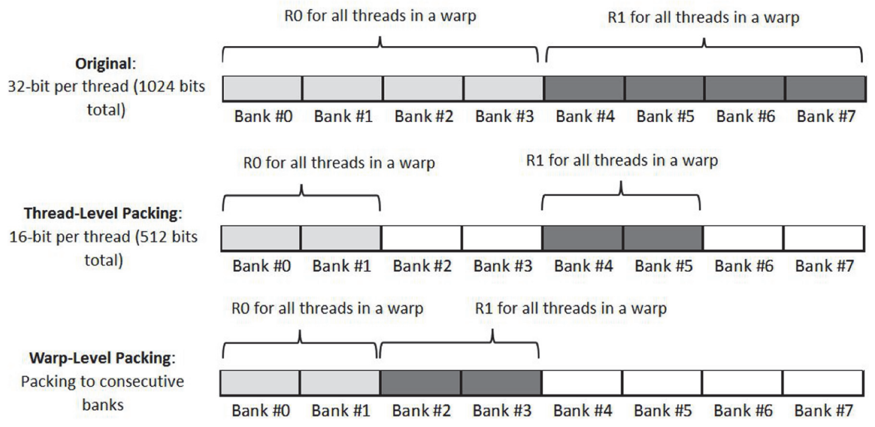

R0 for all threads in a warp
R1 for all threads in a warp
Original:
32-bit per thread (1024 bits total)
Bank #0
Bank #1
Bank #2
Bank #3
Bank #4
Bank #5
Bank #6
Bank #7
R0 for all threads in a warp
R1 for all threads in a warp
Thread-Level Packing:
16-bit per thread (512 bits total)
R0 for all threads in a warp
R1 for all threads in a warp
Warp-Level Packing:
Packing to consecutive banks

Fig. 6. An example for two-level packing approach. Adapted from [20] with permission of IEEE.

instructions. By changing the names of registers for certain instructions, the false data dependencies can be removed. As a result, more instruction-level parallelism (ILP) can be achieved to improve performance. Jeon et al. [24] adapted register renaming for achieving the opposite effect of CPU renaming, namely reducing the number of physical registers without reducing the architectural register space. Motivated by work in [24], in this paper, we leveraged a similar register renaming approach to create an illusion that the total number of architectural registers exceeds the capacity of physical registers so that the barrier set by hardware resources may be overcome. As shown in Fig. 6, the GPU RF was featured with a renaming table which holds all mapping information. An architectural register was mapped to a physical register at the moment of being written. The registers in GPU RF were organized as warp registers. Each warp register has 32 full-size 32-bit registers, each for a thread. Each entry of the renaming table was indexed by global architectural id which can be simply derived with warp ID, local architectural register ID and architectural register usage per thread. A renaming table entry keeps reallocation details for one architectural warp register composed of two data fields including 10-bit for a physical warp register address and 4-bit for a bit mask that indicates the location of the target data fields. In our Fermi like baseline architecture, each SM has a 128-kB register file which is partitioned into 32 banks with 128 256-bit wide entries per bank. Each physical warp register can hold four entries from consecutive banks with the same index. Normally, an architectural warp register is one-to-one mapped to a physical warp register. OWAR offers the opportunity to map multiple architectural registers to a single physical register, for example, 216-bit or 48-bit architectural registers can be assigned to a physical register of 32 bits. A 4-bit mask was used to identify where the values of an architectural register were located within a physical register. Each bit in a 4-bit mask represented a bank entry and a 4-bit mask denoted four entries from consecutive banks with the same index. An architectural register can also be mapped to entries from discrete banks within a physical register. For instance, the data fields of a 16-bit architectural registered whose first half was mapped to the second entry of a physical register and the second half was stored in the last entry, can be indicated by a 4-bit mask, “0101”.

According to the baseline architecture, the total number of physical warp registers per SM was 1024. By using a renaming table with 2048 entries, 1024 physical warp registers can support at most 2048 architectural warp registers on the condition of sufficient narrow-width operands. The total size of renaming table required to maintain all mapping information was calculated by Eq. (2):

$$\textit{Renaming Table Size} = (\#\textit{entry} \times \#\textit{bit per entry}) = 2048 \times 14 \text{ bits} = 3.5 \text{ kB} \tag{2}$$

By increasing the number of entries of the renaming table, OWAR was able to provide even more architectural registers without increasing the number of physical registers. However, naively growing the renaming table capacity can increase hardware overhead and resulted in larger waste in case of the lack of narrow-width operands. We examined 17 benchmarks and found that the demand on RF resources has a 47% reduction on average, thereby we limited the renaming table with 2048 entries (i.e., twice the number of total physical registers) to fit the requirement of most applications and avoid unnecessary waste and overhead. Moreover, a single bit bank entry availability vector per RF bank entry was used to indicate if a corresponding bank entry within a physical warp register was assigned to an architectural warp register or unused [24].

Modification on RF operations: As shown in Fig. 5, to write back results to RF, the width of the computed results was profiled by the narrow-width detecting logic first and then sent to the TLPT. The TLPT made a comparison between the newcomer and the currently maintained operand-width and performed different actions accordingly. If the newly detected width was wider, the TLPT updated the width of the corresponding register and forwarded a misprediction message to the renaming table. The TLPT kept its context unchanged if the detected width was narrower or equal. The misprediction message informed the renaming table to re-map the width-mismatched architectural register to another physical register which has sufficient free entries. The renaming table was then visited with architectural register information to retrieve the corresponding physical register address and the data field indicator (a 4-bit mask). With all this information, the bank arbitrator decided to write results to several entries within the target physical register. The procedure of reading RF was quite straightforward with no concern for the misprediction. Moreover, the register renaming assisted RF offers an economical way to complete a register operation, namely writing or reading RF with a fewer number of entry accesses. By reducing the total number of RF accesses, the RF dynamic energy consumption can be reduced

D. Handling Unused Registers

Due to the use of the register packing technique, OWAR was able to meet the architectural register usage and at the same time lower the demand on physical register resources. Consequently, a great portion of GPU RF can be saved and remain unused. In this work, we explored two methods to handle unused registers for further energy reduction.

Power gating: We applied a traditional coarser-grained subarray power gating [25]. As shown in Fig. 7, GPU RF was partitioned horizontally into several sub-arrays. Taking advantage of register renaming, the in-service physical registers can be concentrated in certain sub-arrays, making the rest of sub-arrays, if any, unoccupied. To save energy consumption, the spared sub-arrays were then power gated. An inactive subarray will be woken up only when active sub-arrays run out of free registers. With a sufficient architectural register supply, which is common for GPUs, the sub-array level power gating scheme shuts down excessive hardware resources to avoid energy waste without significant performance loss.

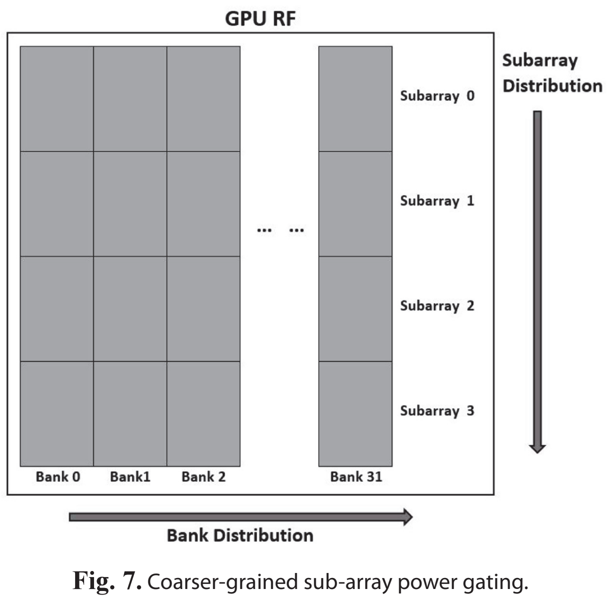


Fig. 7. Coarser-grained sub-array power gating.

Running beyond the limit: Instead of turning off the unused hardware resources, we proposed to fully utilize them for reducing the execution time, with the aim to reduce the energy dissipation. The performance of many GPU applications was prevented from further improvement due to the limitation of hardware resources, especially the number of registers. Since OWAR offered additional RF resources, the GPU was able to increase the occupancy by hosting more co-running threads (also called thread overrun in this paper), which can enhance its capability of hiding memory latencies and hence contribute to better performance. The overall energy consumption may also be reduced due to the shortened execution time.

E. Overheads of OWAR

The total hardware overheads of OWAR include primarily three components: the TLPT, the renaming table, and the bank availability indicator. The TLPT size was 16 bytes and the size of the renaming table was 28672 bits = 3.5 kB as calculated in Eqs. (1) and (2), respectively. According to the baseline RF organization, the total number of bank entries was 4096. Thus, the size of the bank availability indicator was 4096 bits = 0.5 kB (each bit indicated the status of a bank entry, 0 for **free**, 1 for **unused**). The total hardware cost of OWAR implementation was thus 126 + 28672 + 4096 = 32894 bits ≈ 4 kB accounting for 3% of GPU RF size. These energy overheads were included into GPU's total energy model and were described in Section VI. We also considered performance overheads caused by sub-array wake-up delay. Whenever a new sub-array was required to be re-activated from the power gated status, the pipeline was stalled until the sub-array was completely ready for service. The sub-array wake-up penalty was conservatively set to one cycle by default, although it is proven to be less than one cycle by calculation using CACTI-P [25].

V. METHODOLOGY AND HARDWARE DETAILS

We used GPGPU-Sim v3.2.2 [26], a cycle-level GPU performance simulator that focused on general-purpose computation on GPUs, to evaluate the proposed GPU register packing scheme. The energy consumption was measured using GPUWattch [8]. CACTI [27] was used to

Table 2. Simulated GPU architecture configuration

GPU	15 SMs, 700 MHz
SM configuration	16 thread blocks/SM, 32 threads/warp 2 warp schedulers, 1024 ROB entries, 32 SIMD width, 5-Stage pipeline
Register file	128 kB per SM 32 banks, dual-ported (1 read port and 1 write port) for each bank 4 kB register per bank, 256-bit wide entry, 128 entries per bank
L1 cache	16 kB, 4-way set-associative, line size 128 byte, 128 MSHR entries
L2 cache	768 kB, 8-way set-associative, line size 128 byte
Shared memory	48 kB, 32 banks, 1 cycle latency

Data from [20].

Table 3. GPU benchmarks

Benchmarks	Grid	CTA	#Warp per CTA
backprop	(1,1024,1)	(16,16,1)	8
bfs	(4096,1,1)	(512,1,1)	16
b+tree	(1000,1,1)	(256,1,1)	8
cfd	(909,1,1)	(256,1,1)	8
dwt2d	(6,15,1)	(192,1,1)	6
gaussian	(64,64,1)	(16,16,1)	8
hotspot	(43,43,1)	(16,16,1)	8
hybridsort	(490,1,1)	(256,1,1)	8
kmeans	(121,1,1)	(256,1,1)	8
lud	(31,31,1)	(16,16,1)	8
nw	(200,1,1)	(16,1,1)	1
particlefilter	(128,1,1)	(256,1,1)	8
pathfinder	(128,1,1)	(256,1,1)	8
sad1	(128,1,1)	(512,1,1)	16
sad2	(16,16,1)	(16,16,1)	8
2DCONV	(16,64,1)	(32,8,1)	8
2MM	(8,32,1)	(32,8,1)	8

Data from [20].

estimate the dynamic and leakage power of the renaming table. The baseline GPU architecture used was modeled based on NVIDIA GTX480 GPUs [5], which consisted of 15 GPU cores called SMs (Table 2). Each SM has its own private L1 data cache, read-only texture cache, constant cache and software managed shared memory (scratchpad memory). Each SM core owned two warp schedulers and two instruction dispatch units, enabling to issue two independent instructions from two different warps. Each SM core contained 32 single instruction multiple data execution units and four special function units (SFUs). All SMs shared a unified L2 cache. The SMs and the shared L2 cache were connected via an on-chip network. For the GPU RF organization, we assumed a 128-kB register file for each SM. The register file was partitioned to 32 banks with 4 kB register per bank. The multi-banked design was employed to increase GPU RF bandwidth. Each bank was dual-ported (one read port and one write port) with 128 256-bit wide entries. For the benchmarks, we used 17 applications from Rodinia [28] and PolyBench/GPU [29] to evaluate our GPU register packing schemes. Table 3 lists all benchmarks, 15 from Rodinia [28] and 2 from PolyBench/GPU benchmark suit [30]. We summarized the CUDA kernel grouping patterns of grid and CTA for benchmarks.

VI. EXPERIMENTAL RESULTS

We evaluated two OWAR based GPU RF management approaches, OWAR-PG (OWAR with power gating) and OWAR-TO-PG (OWAR with thread overrun and power gating). OWAR-PG focused on reducing RF energy consumption by shutting down spared RF resources. OWAR-TO-PG attempted to improve the energy efficiency through making the most use of the additional RF resources freed by OWAR. And the power gating was reserved by OWAR-TO-PG to further reduce the energy consumption. We compared the energy improvement of these two schemes with the baseline (without OWAR). We also measured the performance improvement of OWAR-TO-PG. We used the geometric mean to report all the averaged values in this paper.

A. RF Utilization

As aforementioned in Section IV, OWAR was able to split a physical register into up to four parts and assign them to multiple architectural registers containing narrow-width results. Although the architectural register usage per warp stayed the same, the corresponding physical register usage was significantly reduced by OWAR. As shown in Fig. 8, OWAR can achieve up to 70.8% off the original register usage per warp and the averaged

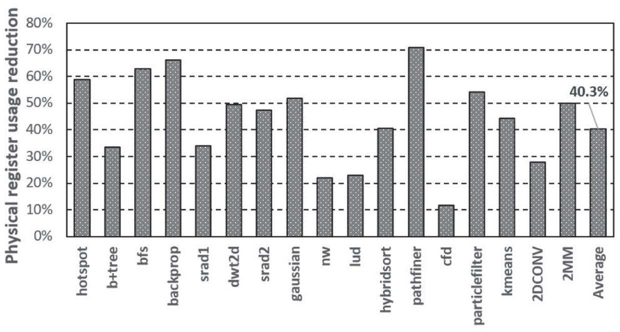


Fig. 8. Reduction in physical register usage.

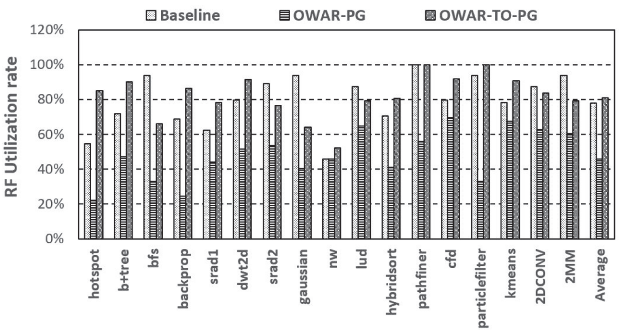


Fig. 9. GPU RF utilization of baseline, OWAR-PG, and OWAR-TO-PG.

reduction for all evaluated benchmarks was 40.3%. The register usage reduction resulted in high underutilization of RF resources. During the run-time, we sampled the number of in-service registers periodically and take the average of all samples as an estimated RF utilization of an application. Fig. 9 shows the GPU RF utilization rates for 17 benchmarks by using the RF management approaches proposed in this work. With no optimization, 77.9% of total RF was used on average. As expected, the RF utilization rate dropped sharply to 45.7% when OWAR-PG was employed, indicating that a great number of registers which were supposed to be used, were now saved by OWAR. The large idle portion of RF can be power gated by OWAR-PG to save RF's dynamic and leakage energy. Unlike OWAR-PG, OWAR-TO-PG tries to improve the RF utilization and the performance by increasing the capacity of concurrent threads. Provided that the performance was boosted, the improvement of energy efficiency was then applied not just to RF, but to all other components in GPU. However, OWAR-TO-PG can result in a higher RF utilization (81.1% on average) compared to the baseline. Even though RF was further exploited by OWAR-TO-PG, for some benchmarks like `bfs`, `Gaussian`, and `nw`, there are still considerable unused registers remaining, which thereby can be power gated to enable more RF energy savings.

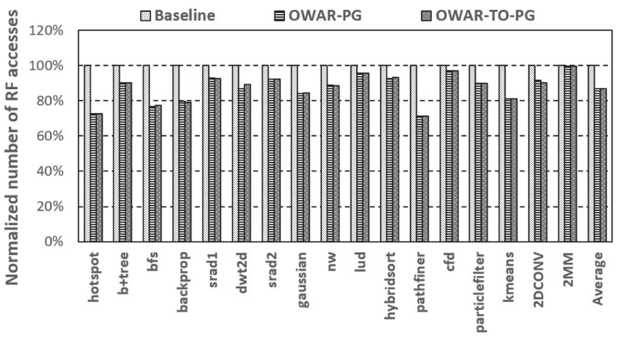


Fig. 10. Normalized number of RF bank accesses.

B. Reducing the Number of RF Bank Accesses

Based on our baseline RF architecture, accessing to a single bank entry can only fetch eight 32-bit register values and reading an operand for a warp instruction needs to access up to four banks in the absence of RF optimization. Through the help of OWAR, fewer bank accesses were necessary to collect all values from RF, for example, one access for 8-bit width data, two accesses for 16-bit width data, if the desired data can be represented with narrowed width and was stored in packed registers. We kept the trace of accesses to narrow-width values and record the total number of RF bank accesses for each benchmark. As shown in Fig. 10, both OWAR-PG and OWAR-TO-PG have the power to complete register operations with less numbers of bank accesses. The benchmark `pathfinder` represented the most significant reduction (28.6%, OWAR-PG and OWAR-TO-PG achieve the same result). On average, the total number of RF bank accesses was reduced by 13.1% for OWAR-PG and OWAR-TO-PG, indicating good opportunities for dynamic energy reduction for register banks.

C. Performance Results

To allow more thread blocks to run on SMs concurrently, we loosen the constraints of the maximum number of thread blocks and warps. Fig. 11 shows the IPC improvement

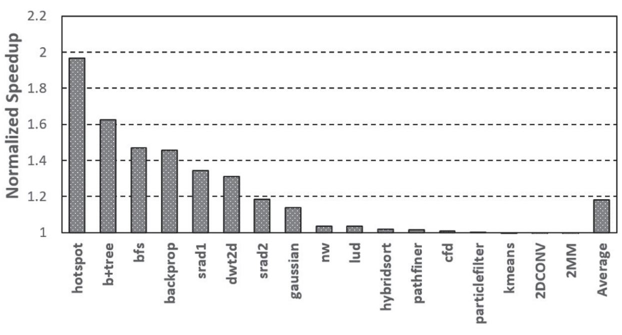


Fig. 11. IPC Improvement.

Table 4. Energy consumption of renaming table compared to RF

	RF	Renaming table
Leakage power (mW)	9.75	0.08
Dynamic energy per read (pJ)	53.6	0.61
Dynamic energy per write (pJ)	57.0	0.63

of 17 benchmarks. As expected, the increased occupancy generally results in better performance. Overall, the performance speedup reached up to 1.97X with the average of 1.18X.

D. Energy Results

Implementation overheads: Table 4 lists the power parameters of the renaming table and RF calculated by CACTI [27]. The leakage energy consumed by the renaming table only accounted for 0.79% of the RF leakage energy. Similarly, the dynamic energy per access to the renaming table only slightly increased the dynamic energy per RF access (1.13% increase per read, 1.10% increase per write). Thus, the energy overhead introduced by the renaming table was negligible in comparison to the energy consumption of the RF.

We also evaluated the performance overhead of sub-array reactivation. We assumed that it took one cycle to completely reactivate a sub-array from the power gated status. As shown in Fig. 12, the performance of only a few benchmarks (`b+tree, backprop, lud`, and `hybirdsort`) was negatively affected by the sub-array re-activation latency. Most of the benchmarks maintain their performance. The reason was that the sub-array re-activation barely happened during the execution, resulting in negligible performance overhead. Unlike other benchmarks, the total number of execution cycles of benchmark `2DCONV` was reduced instead because the overhead cycle unintentionally alleviated the high pressure of cache resource contention. The performance overhead caused by the sub-array re-activation was only 0.2% on average as shown in Fig. 12 for all benchmarks except `2DCONV`.

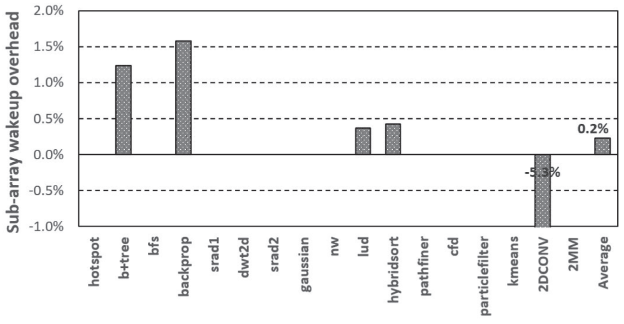


Fig. 12. Sub-array re-activation overhead.

Energy reduction: In Section III, we mentioned that RF consumed a large amount of total energy (i.e., 14.5% of GPU's total energy consumption). Here, we further demonstrated that leakage energy contributes to most of RF's total energy consumption (leakage energy 71%; dynamic energy 29%). Therefore, decreasing RF leakage energy was more effective at improving RF energy efficiency.

First, we focused on evaluating the efficacy of OWAR-PG and OWAR-TO-PG on RF energy reduction. Fig. 13 represents the dynamic, leakage, and total RF energy consumption of two OWAR schemes, respectively. The RF dynamic energy reduction was a result of RF accessing frequency reduction and sub-array level power gating. OWAR-PG and OWAR-TO-PG produced a similar reduction in the number of RF accesses, while OWAR-

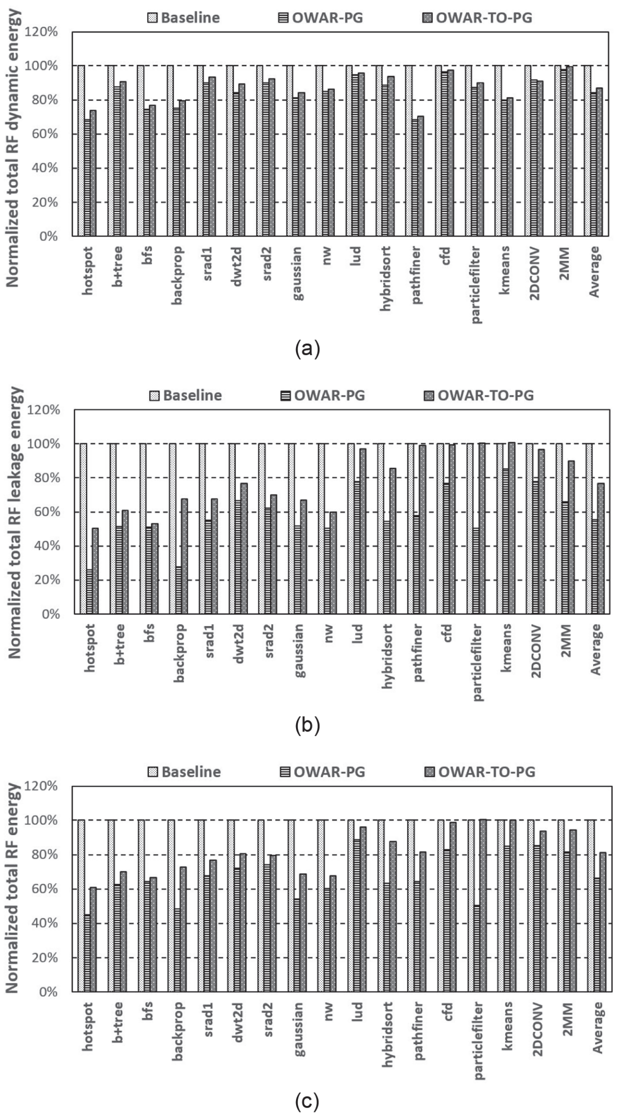
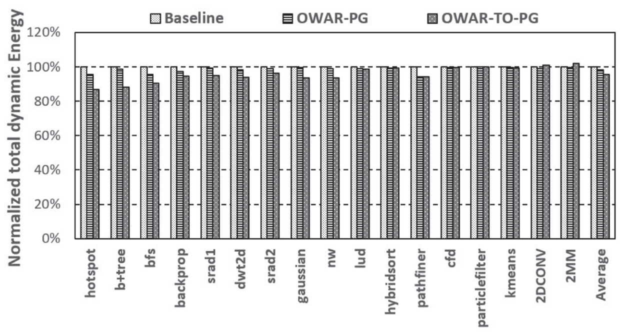
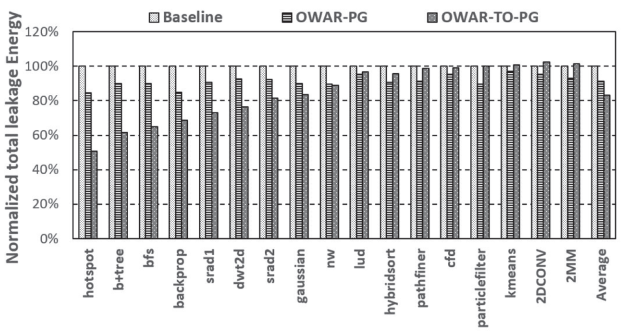


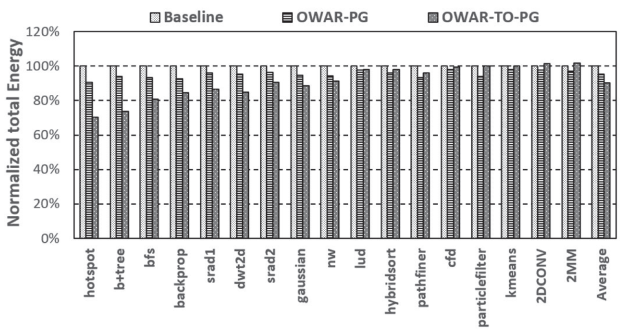
Fig. 13. RF energy reduction: (a) RF dynamic energy, (b) RF leakage energy, and (c) total RF energy.



(a)



(b)



(c)

Fig. 14. GPU energy reduction: (a) GPU dynamic energy, (b) GPU leakage energy, and (c) total GPU energy.

PG was better at power gating due to the availability of more free registers. Thus, OWAR-PG can save more RF dynamic energy than OWAR-TO-PG (OWAR-PG 15.7%; OWAR-TO-PG 13.0%). Similarly, OWAR-PG achieved more RF leakage energy reduction than OWAR-TO-PG (OWAR-PG 44.4%; OWAR-TO-PG 23.1%). Overall, for the purpose of RF energy saving only, OWAR-PG was better than OWAR-TO-PG (OWAR-PG 33.7%; OWAR-TO-PG 18.9%), because OWAR-PG enabled to apply power gating more aggressively.

Second, we examined the effects of OWAR-PG and OWAR-TO-PG in reducing GPU's total energy consumption. Fig. 14 illustrates that OWAR-TO-PG was better as performance was significantly improved. In contrast, OWAR-TO-PG was inferior to OWAR-PG if it resulted in only limited performance improvement. In general, OWAR-TO-PG outperformed OWAR-PG in overall energy reduction (OWAR-TO-PG 9.5%; OWAR-PG 4.8%), because the former can not only reduce the energy consumption of RF but also other components in GPU owing to the overall performance improvement.

VII. RELATED WORK

The narrow-width operand packing techniques were first employed on CPUs, especially multi-threaded CPU processors to mitigate RF pressure. By noticing that many operands called narrow-width operands have fewer significant bits compared to the full width of a 32-bit register, several researchers proposed to detect them and merge multiple narrow-width operands [10-12, 14, 15]. The narrow-width operand packing techniques for CPU were then designed to save power consumption [12] and improve performance [11, 12] as well as register file reliability [14, 15]. There is also prior work to reduce GPU RF leakage energy by exploiting a drowsy register file [31], and application-specific information [32]. Brooks and Martonosi [10] proposed hardware mechanisms for general-purpose microprocessors that dynamically recognize and capitalize on narrow-width operands. They introduced both power-oriented and performance-oriented optimizations. Power-oriented optimization reduced processor power consumption by using aggressive clock gating to turn off portions of integer arithmetic units that will be unnecessary for narrow-width operations and can achieve 50% power consumption reduction [10]. Performance-oriented optimization improved performance by merging together narrow integer operations and allowed them to share a single functional unit and the performance speedup is around 10% [10]. Ergin et al. [11] extended Brooks and Martonosi [10]'s work to reduce the pressure on the register file in super-scalar processors. They proposed two prediction-based schemes to dynamically packing multiple narrow-width results into partitions within a single register. Their approach can improve the CPU performance by 15% on average. Although these two techniques showed advantages on CPUs, they cannot be applied on GPUs directly due to the difference of architecture and RF size between CPU and GPU. Modern GPUs features a huge size of the RF in order to support a massive number of active threads. GPUs rely on tens of thousands of concurrent threads to obtain high throughput. On contrary, CPUs mainly focus on single applications performance and only support tens of threads and utilize large caches to reduce memory access latency instead of massive TLP. For the reasons above, conventional CPU register packing techniques were not suitable for GPUs.

To the best of our knowledge, there is no prior GPU-exclusive register packing technique aiming at leakage energy reduction for GPGPU programs. However, several studies have introduced similar techniques [16-19].

Observing that the register values of threads within the same warp were similar, Lee et al. [16] presented Warped-Compression, a warp-level register compression scheme which removed data redundancy of register values through register compression to enable power reduction opportunities. This technique saved 25% of the total RF power consumption. Tan et al. [17] proposed the narrow-width-aware register write back method which combined two narrow-width writes to share data bus resource and hence enhance the performance. Gilani et al. [18] noticed that many operands required considerably fewer bits for accurate representation and computations. They proposed a sliced GPU architecture which was much alike the method designed for CPUs in [10]. Their approach improved the performance of the GPU up to 15% by dual-issuing instructions to two 16-bit execution slices. Wang and Zhang [20] exploited narrow-width operands for GPU performance improvement, while this paper focuses on dynamic and leakage energy reduction with different techniques and different architectural support.

VIII. CONCLUSION

In this paper, we dynamically exploited narrow-width operands to improve the RF and GPU energy efficiency. Our approach utilized the additional registers saved by register packing to reduce energy dissipation through power gating and thread overrun. The experimental results showed that our method can reduce the GPGPU's total energy up to 29.6% and 9.5% on average, with a hardware overhead within 3% of the GPU RF. In addition, the performance can be improved up to 1.97X and 1.18X on average.

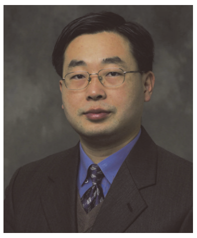
REFERENCES

1. K. Fatahalian and M. Houston, "A closer look at GPUs," *Communications of the ACM*, vol. 51, no. 10, pp. 50-57, 2008.
2. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370-1380, 2008.
3. P. Karnick, "GPGPU: general purpose computing on graphics hardware," 2006; https://www.researchgate.net/publication/262293516_GPGPU_General_purpose_computation_on_graphics_hardware.
4. E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: a unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39-55, 2008.
5. NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. Santa Clara, CA: NVIDIA, 2009.
6. NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210*. Santa Clara, CA: NVIDIA, 2014.
7. S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, Saint-Malo, France, 2010, pp. 280-289.
8. J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: enabling energy optimizations in GPGPUs," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 487-498, 2013.
9. J. Lim, N. B. Lakshminarayana, H. Kim, W. Song, S. Yalamanchili, and W. Sung, "Power modeling for GPU architectures using McPAT," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 3, article no. 26, 2014. https://doi.org/10.1145/2611758
10. D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proceedings 5th International Symposium on High-Performance Computer Architecture*, Orlando, FL, 1999, pp. 13-22.
11. O. Ergin, D. Balkan, K. Ghose, and D. Ponomarev, "Register packing: exploiting narrow-width operands for reducing register file pressure," in *Proceedings of the 37th International Symposium on Microarchitecture (MICRO-37'04)*, Portland, OR, 2004, pp. 304-315.
12. M. Kondo and H. Nakamura, "A small, fast and low-power register file by bit-partitioning," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, San Francisco, CA, 2005, pp. 40-49.
13. M. Budiu, M. Sakr, K. Walker, and S. C. Goldstein, "BitValue inference: detecting and exploiting narrow bitwidth computations," in *Euro-Par 2000 Parallel Processing*. Heidelberg, Germany: 2000, pp. 969-979.
14. J. Hu, S. Wang, and S. G. Ziavras, "In-register duplication: exploiting narrow-width value for improving register file reliability," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, Philadelphia, PA, 2006, pp. 281-290.
15. J. Hu, S. Wang, and S. G. Ziavras, "On the exploitation of narrow-width values for improving register file reliability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 7, pp. 953-963, 2009.
16. S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: enabling power efficient GPUs through register compression," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, Portland, OR, 2015, pp. 502-514.
17. J. Tan, Z. Li, and X. Fu, "Soft-error reliability and power co-optimization for GPGPUS register file using resistive memory," in *Proceedings of 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2015, pp. 369-374.
18. S. Z. Gilani, N. S. Kim, and M. J. Schulte, "Power-efficient computing for compute-intensive GPGPU applications," in *Proceedings of 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Shenzhen, China, 2013, pp. 330-341.
19. M. Abdel-Majeed and M. Annavaram, "Warped register file: a power efficient register file for GPGPUs," in *Proceedings of 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Shenzhen, China, 2013, pp. 412-423.

20. X. Wang and W. Zhang, "GPU register packing: dynamically exploiting narrow-width operands to improve performance," in *Proceedings of 2017 IEEE Trustcom/BigDataSE/ICESS*, Sydney, Australia, 2017, pp. 745-752.
21. X. Wang and W. Zhang, "Packing narrow-width operands to improve energy efficiency of general-purpose GPU computing," in *Proceedings of 2020 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, 2020, pp. 1-7.
22. NVIDIA, *CUDA Programming Guide Version 2.1.* Santa Clara, CA: NVIDIA, 2008.
23. N. Jayasena, M. Erez, J. H. Ahn, and W. J. Dally, "Stream register files with indexed access," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA)*, Madrid, Spain, 2004, pp. 60-72.
24. H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, "GPU register file virtualization," in *Proceedings of the 48th International Symposium on Microarchitecture*, Waikiki, HI, 2015, pp. 420-432.
25. S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proceedings of 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, 2011, pp. 694-701.
26. A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceedings of 2009 IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, 2009, pp. 163-174.
27. S. J. Wilton and N. P. Jouppi, "CACTI: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677-688, 1996.
28. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: a benchmark suite for heterogeneous computing," in *Proceedings of 2009 IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, 2009, pp. 44-54.
29. S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *Proceedings of 2012 Innovative Parallel Computing (InPar)*, San Jose, CA, 2012, pp. 1-10.
30. L. N. Pouchet, "PolyBench/C: the polyhedral benchmark suite," 2012; http://web.cs.ucla.edu/~pouchet/software/polybench/.
31. X. Wang and W. Zhang, "Drowsy register files for reducing GPU leakage energy," in *Proceedings of 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, Shenzhen, China, 2017, pp. 632-639.
32. X. Wang and W. Zhang, "Energy-efficient DNN computing on GPUs through register file management," in *Proceedings of 2018 IEEE High Performance extreme Computing Conference (HPEC)*, Waltham, MA, 2018, pp. 1-7.
33. X. Xie, Y. Liang, X. Li, Y. Wu, G. Sun, T. Wang, and D. Fan, "Enabling coordinated register allocation and thread-level parallelism optimization for GPUs," in *Proceedings of 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Waikiki, HI, 2015, pp. 395-406.

Xin Wang

Xin Wang received his B.S. degree in electronic engineering from Peking University, China, in 2008. He is currently pursuing his Ph.D. degree in computer engineering at Virginia Commonwealth University, USA. His research interests include GPU and heterogeneous CPU-GPU architectures.



Wei Zhang https://orcid.org/0000-0003-1343-2817

Wei Zhang is a professor and Chair of the Department of Computer Engineering and Computer Science at the University of Louisville. He received his Ph.D. in Computer Science and Engineering from the Pennsylvania State University in 2003. Dr. Zhang served as an assistant/associate professor in Electrical and Computer Engineering at Southern Illinois University Carbondale (SIUC) from 2003 to 2010 and as an associate and full professor at Virginia Commonwealth University from 2010 to 2019. His research interests are in computer architecture, compiler, real-time computing, and hardware security. Dr. Zhang has led 8 NSF projects as the PI and has published 160+ papers in refereed journals and conference proceedings. He received the 2016 Engineer of the Year Award from the Richmond Joint Engineer Council, the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and the 2007 IBM Real-time Innovation Award.