Microprocessors and Microsystems 50 (2017) 175–188

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Emerging technology enabled energy-efficient GPGPUs register file

Chenhao Xie[a], Jingweijia Tan[a], Mingsong Chen[b], Yang Yi[c], Lu Peng[d], Xin Fu[a,*]

[a] ECE Department, University of Houston, 4800 Calhoun Road, Houston, TX 77204, USA
[b] Software Engineering Institute, East China Normal University, Huashida, Putuo, Shanghai, 200062, China
[c] EECS Department, University of Kansas, 1450 Jayhawk Blvd, Lawrence, KS 660045, USA
[d] EECS Department, Louisiana State University, Baton Rouge, LA 70803, USA

ARTICLE INFO

Article history:
Received 6 October 2015
Revised 25 February 2017
Accepted 3 April 2017
Available online 4 April 2017

Keywords:
Tunneling field effect transistors (TFETs)
General-purpose computing on graphics processing units (GPGPUs)
Energy efficiency

ABSTRACT

Modern Graphics Processing Units (GPGPUs) employ the fine-grained multi-threading among thousands of active threads, leading to the sizable register file (RF) with massive energy consumption. In this study, we explore the emerging technology (i.e., Tunnel FET (TFET)) enabled energy-efficient GPGPUs RF. TFET is much more energy-efficient than CMOS at the low voltage operations, but always using TFET at the low voltage (so that low frequency) causes significant performance degradation. In this study, we first design the hybrid CMOS-TFET based register file, and propose the memory-contention-aware TFET register allocation (MEM_RA). MEM_RA allocates TFET-based registers to threads whose execution progress can be delayed to some degree to avoid the memory contentions with other threads, and the CMOS-based registers are still used for threads requiring normal execution speed. We further observe the insufficient TFET register resources for the memory-intensive benchmarks when applying the MEM_RA technique. We then develop the TFET-register-utilization-aware block allocation (TUBA) and TFET-regsiter-request-aware warp scheduling (TRWS) mechanisms to effectively utilize the limited TFET registers and achieve the maximal energy savings. Our experimental results show that the proposed techniques achieve 40% energy (including both dynamic and leakage) reduction in GPGPUs register file with negligible performance overhead.

1. Introduction

Modern graphics processing units (GPUs) support tens of thousands of parallel threads and deliver remarkably high computing throughput. General-Purpose Computing on GPUs (GPGPUs) are becoming the attractive platform for general-purpose applications that request high computational performance such as scientific computing, financial applications, medical data processing, and so on. However, GPGPUs are facing severe power challenge due to the increasing number of cores placed on a single chip with decreasing feature size [1].

In order to hide the long latency induced by the function unit computation and off-chip memory accesses, GPUs employ the fine-grained multi-threading that quickly switches among a large number of simultaneously active threads. As a result, a sizable register file is required to keep the context of each thread. For example, Nvidia Kepler GPUs supports more than 20,000 parallel threads and contains a 2 MB register file [7], and the register file size keeps increasing in recent GPU products. Accessing such large register file leads to massive power consumption [2–6]. It has been reported that the register files consume 15%–20% of the GPU streaming multiprocessor's power [8,36]. Effectively optimizing the register file power consumption is critical and the first step towards the energy-efficient GPUs.

Supply voltage scaling is the fundamental technique to reduce the dynamic power consumption, but it is limited by the leakage constraints in CMOS digital circuits. As an emerging technology, Inter-bank Tunneling Field Effect Transistors (TFETs) have shown to be the attractive candidates to operate at low supply voltages (e.g. 0.3 V) with ultra low leakage and higher frequency than CMOS [9,10]. However, at higher supply voltage, CMOS devices are able to achieve much better performance than TFETs. The unique characteristics of CMOS and TFETs at different voltage levels provide great opportunity in GPUs energy savings without hurting the performance.

In the GPGPU applications, all threads in a kernel execute the same code [11], and exhibit similar execution progress in the fine-grained multi-threading environment. When one thread encounters an off-chip memory access, other threads are likely to issue the requests at approximately the same time, leading to severe memory contentions which extend the memory access time. The pipeline in the GPUs streaming multiprocessor (SM) stalls when all threads stall due to the long-latency memory accesses. It has been

* Corresponding author.
E-mail address: xfu8@central.uh.edu (X. Fu).

http://dx.doi.org/10.1016/j.micpro.2017.04.002
0141-9331/

found that the performance of numerous GPGPU workloads are still bounded by the memory resources even modern GPUs provide very high memory bandwidth [29]. In order to alleviate the memory contentions and efficiently utilize the memory bandwidth, threads can run at different paces which effectively avoid the interferences among memory requests. It enables the implementation of the TFET-based registers in GPGPUs for a number of threads so that they can run at a lower frequency without any performance degradation, and meanwhile, both the dynamic and leakage power of the registers reduces substantially. On the other hand, applying TFETs for all registers in the GPGPU will cause significant performance penalty since many threads still need to execute at a high frequency to achieve the high computational throughput.

In this paper, we exploit the emerging technology (i.e., TFET) enabled energy-efficient GPUs register file: we propose the hybrid CMOS-TFET based registers and explore a set of mechanisms to obtain optimal energy reduction with negligible performance penalty. The contributions of this study are as follows:

- We observe that threads in GPGPU workloads can be seriously delayed while executing in the GPU streaming multiprocessors (SMs) due to the memory access interference with others. Instead of stalling in the pipeline on the occurrence of serious memory contentions, threads can execute at a low speed by using TFET-based registers to postpone their memory requests. It helps to achieve the win-win scenario: preventing the interferences and achieving the attractive power savings.
- We propose to build the hybrid TFET-based and CMOS-based registers, and perform the memory contention-aware register allocation (MEM_RA). Based on the access latency of previous memory transaction, we predict the thread stall time during its following memory access, and allocate TFET-based registers to that thread to postpone its execution progress to the maximum degree without performance loss.
- We further observe the overall insufficient TFET registers throughout the entire execution for memory-intensive benchmarks under MEM_RA. Moreover, the TFET registers utilization is unbalanced across the GPU SMs, especially, TFET registers in some SMs are largely under-utilized at the run-time. We then propose the TFET-register-utilization-aware block[1] allocation (TUBA) to balance the TFET register utilization at the inter-SM level so that the overall limited TFET registers can be better effectively used for more energy savings without hurting the performance.
- We also propose the TFET-register-request-aware warp[2] scheduling (TRWS) inside each SM so that the limited TFET registers are able to serve more TFET register requests at the intra-SM level and maximize the energy savings.
- Our evaluation results show that our proposed techniques in the hybrid register design exhibits the strong capability of reducing the register energy consumption (including both dynamic and leakage energy) by 40% compared to the case with naive energy optimization technique (i.e. power gating the unused registers [2]). Especially, it achieves 54% energy reduction (21% dynamic saving and 33% leakage saving) in memory-intensive benchmarks with only 1.5% performance degradation.

The rest of the paper is organized as follows: Section 2 provides the background of GPGPUs and TFETs. Section 3 presents our key observation on the memory contentions in GPGPUs. Section 4 proposes the hybrid CMOS-TFET based register file, and a set of mechanisms to gain the energy savings including the memory contention-aware TFET-based register allocation, the TFET-register-utilization-aware block allocation, and warp scheduling techniques. Section 5 describes our experimental methodologies and Section 6 evaluates the proposed mechanisms. We discuss the related work in Section Section 7, and conclude with Section 8.

[1] Threads are allocated to the SMs at the granularity of blocks in GPUs.

[2] In each GPUs SM, threads are executed in warps and follow the SIMD mode.

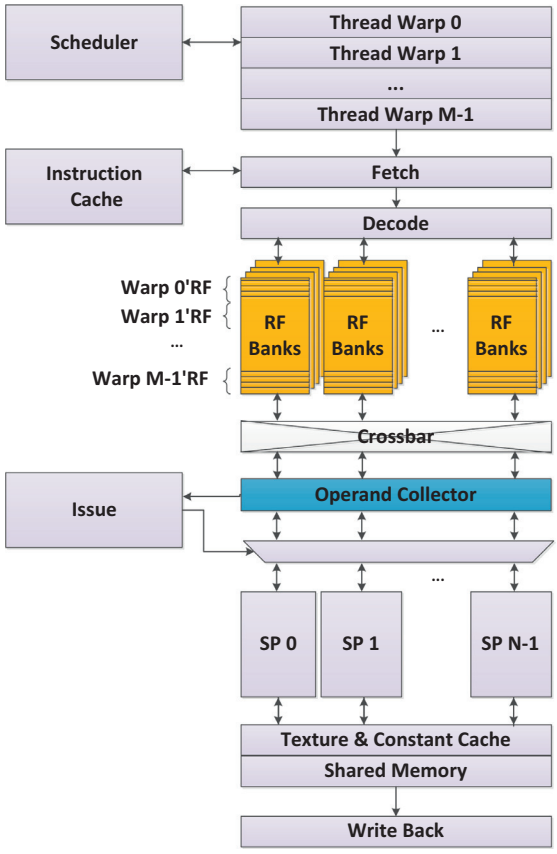


Fig. 1. Streaming multiprocessor microarchitecture.

2. Background

2.1. General-purpose computing on graphics processing units (GPGPUs) architecture

Typical GPUs consist of a scalable number of in-order streaming multiprocessors (SMs) that can access multiple on-chip memory controllers via an on-chip interconnection network [11]. In GPU programming models, highly-parallel kernel functions are launched to the GPUs to execute. The kernel is composed of a grid of light-weighted threads which is divided into a set of blocks. Each block concludes hundreds of threads and threads in the kernel are assigned to the SMs at the granularity of blocks. After the kernel is launched to the GPUs, block scheduler will checks the physical resources (e.g. register, shared memory and threads capacity) requirement for each block and allocates available blocks into the appropriate SM having plenty of physical resources in a round-robin and load balance fashion.

Fig. 1 illustrates the SM micro-architecture. Threads in each SM execute following the single-instruction multiple-data (SIMD) model. A number of individual threads (e.g. 32 threads) 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. As Fig. 1 shows, each warp has a dedicated slot in the warp scheduler. At every cycle, a ready warp is selected by the scheduler to feed the pipeline. The instruction is then fetched from the instruction cache based on the PC of the warp, and further decoded. In the SM, numbers of registers are statically allo-

cated to each warp when the block is distributed. Before the warp executes across streaming processors (SP), the operand collector (OC) reads register values for all threads from the register file (RF). Each instruction can read up to 4 register values and write to 1 operand for each thread so that a extreme large number of registers is required.

GPUs are usually equipped with the off-chip external memory (e.g. global memory) connected to the on-chip memory controllers. The off-chip memory access can last hundreds of cycles, and a long-latency memory transaction from one thread would stall all threads within a warp. In other words, the warp cannot proceed until all the memory accesses from its threads complete.

2.2. GPU register file

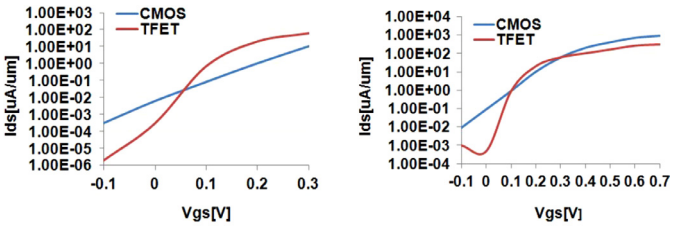
Modern GPGPUs utilize the thread-level parallelism (TLP) which groups threads as warp and concurrently execute many warps on the same SM. To support thousands of threads executing simultaneously, each SM consists of large register file to hold all active thread contents. For instance, the NVIDIA Kepler GPUs have 65,536 32-bit registers per SM which equals to 2MB register file in total [7]. The register file size is much larger than it for commercial CPU and even larger than the L2 cache in GPGPUs.

In order to reduce the area overhead of the register file, the 128 KB baseline register file in each SM is composed of 32 dual-ported (1W/1R) 4 KB banks [4,32,36,39,40]. There are 256 register entries in a bank, each of which is 128-bit wide and contains four 32-bit registers [4,39,40]. All the 32 same-named registers required by a warp (e.g., R0s of all threads within warp1) are allocated to the same register entries of 8 banks. In every cycle, register access requests to different banks can proceed simultaneously, while concurrent requests to the same bank have to be serialized, causing bank conflicts. In order to handle additional access latency caused by bank conflicts, operand collectors are used to buffer the operands without stalling the pipeline. In our baseline case, each SM is equipped with four operand collector units, each of which contains three operand entries [40]. In modern GPUs, 32 SIMD lanes are grouped into 8 clusters and each cluster contains 4 register banks for 4 SIMD lanes [4]. Hence, one 4×4 128-bit wide crossbar is used in each cluster to transfer the operand values to proper operand collector unit.

In the baseline case, the register file in SM is statically allocated to each warp as shown in Fig. 1 which means the register cannot be used by other warps until the current warp finishes its execution. To avoid losing the active thread contents, register file need to be powered up even warps are stalled or waiting to execute. Hence, accessing such large register file not only consumes massive dynamic but also leakage power [2–6].

2.3. Tunneling field effect transistors (TFETs)

The sub-threshold slope of the transistor is the key factor in leakage power consumption, and a steep sub-threshold device achieves low leakage current. Traditional CMOS devices are limited to 60 mV/decade sub-threshold slope which induces high leakage current during the voltage scaling [12]. While TFETs [9] exhibit sub-60 mV/decade sub-threshold slope and achieve very low leakage power consumption at low supply voltage. Fig. 2(a) compares the OFF-state leakage current (I_{OFF}) and ON current (I_{ON}) of the two kinds of devices when V_{CC} is 0.3 V. As it shows, TFETs are able to obtain much lower leakage current and stronger driven current, therefore, ultra low leakage with high frequency. They are promising for energy-efficient computing. On the other hand, as Fig. 2(b) exhibits, although TFETs are still able to achieve low I_{OFF} at high supply voltage (e.g. 0.7 V), CMOS devices have larger driven current and better performance than TFETs.



(a) VCC=0.3V (b) VCC=0.7V

Fig. 2. OFF-state leakage current and ON current of TFET and CMOS (adapted from [13]).

TFETs have the characteristic of uni-direction conduction which causes a challenge on designing the SRAM storage cell. Recently, many different TFET SRAMs have been explored to overcome this limitation [14–17]. Meanwhile, Avci et al. [38] shows the fabrication cost of integrating TFET and CMOS is negligible as the TFET can be fabricated by using conventional CMOS technologies. By comparing those designs on several aspects (e.g. frequency, noise margins, power, and area), in this study, we apply the 6T TFET SRAM proposed by Singh et al. [14] to implement the TFET-based register files.

3. The key observation: memory contentions in GPGPUs

In GPUs, the off-chip memory requests from SMs need to go through the on-chip network routing to certain memory controller and wait there to be served. When numerous requests are issued at similar time by multiple SMs, both on-chip network and memory controllers will be severely congested which significantly increases the memory access time. Unfortunately, such congestion issue occurs frequently in GPUs due to the unique characteristic of the GPGPUs applications: all threads in the kernel across SMs execute the same instructions and proceed at similar rate in the fine-grained multithreading environment. Although there are up to thousands of active threads running in each SM, they are unlikely to fully hide the extremely long-latency memory transaction caused by the memory contentions. As a result, the SM suffers long-time pipeline stall. The GPUs memory bandwidth is already considered as one of the resource constraints for many GPGPUs workloads even modern GPUs provide pretty high memory bandwidth [29].

Fig. 3(a) shows an example of the memory resource contentions among SMs. Several SMs encounter the global memory access instructions and send out memory requests simultaneously. The buffers in network-on-chip (NoC) and memory controllers are quickly filled up by those requests and they have to be served sequentially to access the DRAM buffers (Fig. 3(a) takes a snapshot on the NoC and memory controllers). Therefore, the memory transactions spend longer time to finish, and the pipeline in SMs quickly turns to be idle (highlighted as orange ellipse in Fig. 3(a)) since other active threads in the SM will stall at the memory instructions in the near future as well. The thread throttling mechanism has been proposed recently to alleviate the memory contentions and shrink the pipeline idle time [18]. It dynamically stalls certain threads to restrict the number of concurrent memory tasks and avoid the interferences among memory requests. As can be seen, appropriately slowing down the threads before their memory accesses can even introduce positive effect on performance. Allocating the TFET-based registers to those threads and managing them to execute at low frequency during the register read/write operations provides the perfect approach to control the thread progress. Fig. 3(b) demonstrates the example of intelligently leveraging the low frequency operations on TFETs to absorb the pipeline stall time

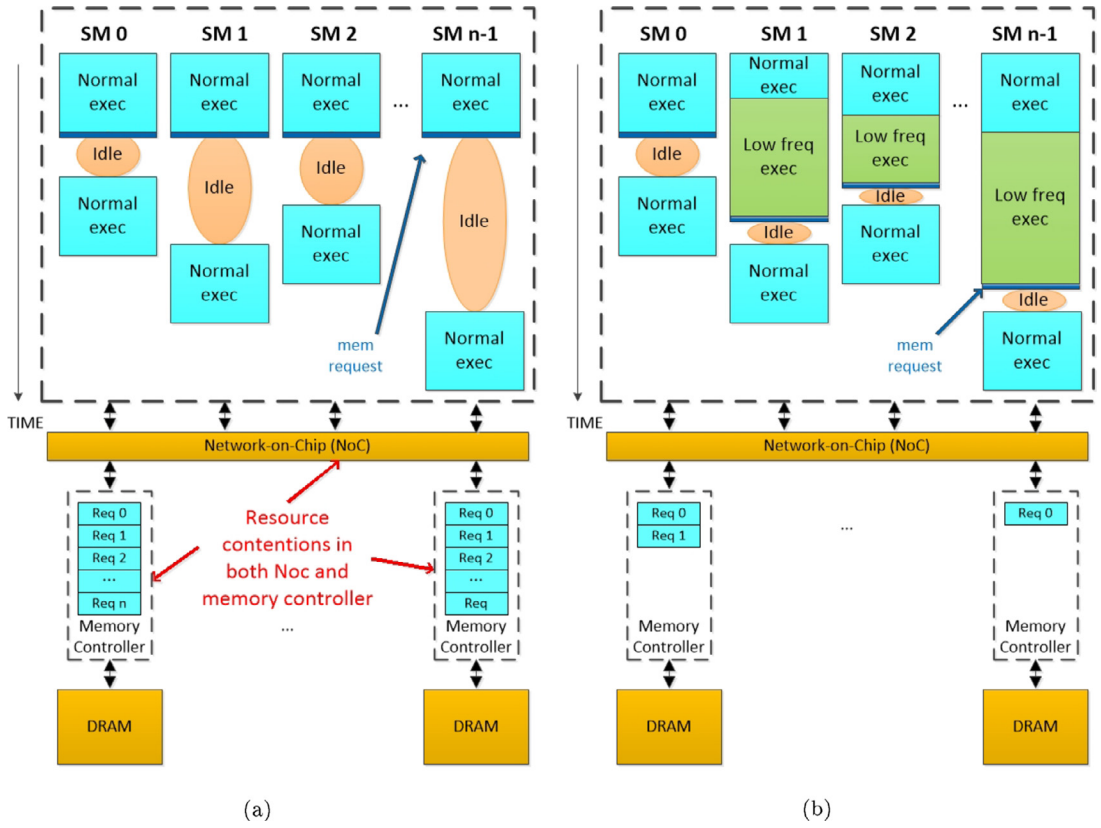


Fig. 3. (a) SMs suffer long pipeline stall due to the severe memory contentions. (b) Leveraging TFETs to absorb the long pipeline stall and alleviate memory contentions.

(shown in green rectangles) and meanwhile, separate the memory requests from SMs. As it shows, both NoC and memory controllers have few queued requests, and the off-chip memory access time reduces significantly. More importantly, the benefit of TFETs on reducing both dynamic and leakage energy is effectively explored. Obviously, CMOS-based registers are essential during the normal execution.

4. Hybrid CMOS-TFET based register files

In this section, we explore the hybrid CMOS-TFET based registers, and propose a set of novel mechanisms to effectively use the TFET-based registers in delaying threads' execution speed to gain the maximal energy savings and meanwhile, still maintain the performance.

4.1. Memory contention-aware TFET register allocation (MEM_RA)

4.1.1. The idea

As described in Section 2.1, when threads are launched to the SM, a number of registers are statically designated to them according to their resource requirements. The register ID encoded in the instruction is used as the index to the physical register being read/written. In other words, the mapping between the register ID and the physical register is fixed all the time. However, when applying the same mapping mechanism in the hybrid register, the use of the TFET-based registers cannot be managed at the run time.

In this work, a register renaming table is applied to record the physical register number corresponding to the register ID encoded in the instruction. A register renaming stage is inserted into the SM pipeline following the decode stage. Note that this additional stage does not affect back-to-back instruction latencies. It only induces 1.5% performance overhead based on our evaluation across a large set of GPGPU benchmarks (detailed experimental methodologies are described in Section 5.), which also matches the observation made in [4]. During the register renaming stage, the destination register ID is renamed to a free physical register. The renaming table also provides the information of physical registers to be read according to the source register IDs. Therefore, the thread has the flexibility to map a register to either CMOS or TFET based physical register. A register in the renaming table is released after its last read, and the register lifetime information can be simply obtained by the compiler which indicates the last instruction reading the register. Since threads in a single warp execute the same instruction, they share the same renaming information. The execution of a branch instruction may cause warp divergence, threads in a diverged warp execute in serial fashion. A physical register will not be released until the last read finishes across all threads in the warp.

The critical challenge in the hybrid register design becomes the runtime CMOS-TFET physical register allocation to the destination register ID in the warp instruction. Aggressively utilizing the TFET registers degrades the performance significantly; on the other hand, too conservatively using the TFET registers fails to achieve the goal of maximizing the registers energy savings. Moreover, the TFET utilization among warps needs to be different to well control the warp execution progress and avoid the interferences. As can be seen, it is crucial that the TFET-based register allocation adapts to the memory access pattern of the workloads. For example, randomly or periodically renaming the destination registers to TFET registers can easily hurt the performance as they are blind to the memory accesses. It is highly possible that the TFET registers are improperly used when there are few memory transactions and the high throughput is expected during that period of the workload execution. We propose the MEMory contention-aware TFET Register Allocation (named as MEM_RA as abbrevia-

tion) to achieve the optimal power savings with little performance penalty.

Recall that SM supports the SIMD execution mode, threads from a warp exhibit the same progress and stall for the same amount of time, therefore, the stall time at the warp level is the finest granularity can be considered. The warp stall time due to the off-chip memory access implies the severity of the memory contentions. A long waiting time means the occurrence of serious contentions, and if the memory request from the warp had been postponed by using the TFET registers, such contentions may be removed successfully. Unfortunately, the waiting time is not available until the request has already been serviced and the contentions take place. We use the last value prediction mechanism to predict the warp stall time in its next global memory transaction based on the previous memory access latency, and utilize TFET registers to absorb that predicted stall time before the warp sends out its memory request.

Note that the warp has already been slowed down to some degree in previous memory transaction, its following memory request might not interfere with others and it is unnecessary to further delay its progress. This happens in kernels with heavy computation tasks which help to separate the memory transactions and relief the memory contentions. However, the case is different in memory-intensive benchmarks. Even the warp has been delayed before, its following memory access can still get involved with memory transactions from other warps due to the frequently issued memory requests, and further postponing its execution progress is desired.

In order to delay the warp appropriately across various types of workloads, we sample the memory access latency periodically at run time and introduce it into the warp stall time prediction. Eq. (1) describes the analytical model to predict the stall cycles (represented as *SC*) of a warp based on its previous memory access latency (represented by *prev_acc*) and the latest sampled memory access latency (represented by *sample_acc*),

$$\mathbf{SC} = \begin{cases} \mathbf{0}, & if \quad prev\_acc \le thr\_acc \\ \left[\dfrac{\mathbf{sample\_acc}}{\mathbf{ref\_acc}} \times \mathbf{(prev\_acc\text{-}thr\_acc)}\right], & \\ if \quad sample\_acc < ref\_ac, prev\_acc > thr\_acc & \\ \mathbf{prev\_acc\text{-}thr\_acc}, & \\ if \quad sample\_acc \ge ref\_ac, prev\_acc > thr\_acc & \end{cases} \tag{1}$$

where *thr_acc* is the threshold latency to determine whether the warp should be delayed in the near future. It is set as the memory access cycles under perfect memory system (e.g. 10 core cycles in our GPU machine configuration). When the *prev_acc* is no longer than the *thr_acc*, it implies that the previous memory transaction does not run into any congestion and the warp proceeds at good speed rate, so no delay is required. *ref_acc* is the referred memory access latency describing the memory access time with moderate resource contentions. When *sample_acc* is longer than *ref_acc*, it implies that the kernel currently exhibits the memory-intensive characteristic, the aggressive delay on the warp execution is preferred. The stall cycle is directly set as the extra waiting time in the previous memory access (i.e. *prev_acc* minus *thr_acc*). To the contrary, a short *sample_acc* compared to *ref_acc* means that the kernel involves heavier computation tasks, the predicted stall time is scaled down according to the ratio of *sample_acc* to *ref_acc*.

Once the stall time is calculated by using the analytical model above, the warp starts to allocate TFET-based registers to the destination register IDs in its following execution. Generally, the access delay of TFET-based SRAM at low supply voltage is 1ns slower than that of the CMOS-based SRAM at normal voltage [15]. Therefore, the access time to TFET registers is modeled as 2 cycles in our study regarding to the core frequency (detailed machine configurations are listed in Table 1). In other words, one extra cycle is required to finish the TFET register read/write operation. The TFET register allocation is disabled when the predicted stall time is expected to be fully absorbed. Note that the register read time lasts 2 cycles as long as there is one TFET-based source register. When a warp diverges at a branch instruction, the extra delay is also modeled for all the sequentially executed threads if they use TFET registers. A warp issues multiple memory transactions when a load instruction is executed and the load requests from threads belonging to that warp fail to get coalesced. Those transactions may complete at different time, as a result, the register write back cannot be performed concurrently. Writing values to TFET registers in a load instruction is likely to induce quite long delay which easily over-postpones the warp. The TFET register allocation for load instructions are skipped in MEM_RA.

Table 1
Our baseline GPGPU configurations.

GPGPUs core configuration	
Core(SM) frequency	1 GHz
Number of SMs	15
Warp size	32
SIMT pipeline width	32
Warp scheduling policy	Greedy-Then-Oldest(GTO)
Register file size	128 KB per SM (32 banks)
Number of collector unit	4 per SM
Register file banks	128-bit wide, 256 entries SRAM
Operands collector	1024-bit wide SRAM
OC crossbar network	8 128-bit wide 4x4 crossbars
Max block occupancy	8 blocks (1536 threads) per SM
Memory hierarchy	
Shared memory size	48 KB per SM
L1 data cache size	16 KB per SM
L2 cache size	128 per memory channel
Memory channel	6
Memory frequency	1.25 GHz
Off-chip bandwidth	128 GB/s
Interconnect network	
Interconnect frequency	1 GHz
Topologies	Mesh
Routing algorithm	dimension-order routing
Channel width	16 B

4.1.2. The implementation

4.1.2.1. The number of the TFET-based registers. Since CMOS- and TFET-based SRAMs have similar size [14,28], we set the total amount of hybrid registers in each SM as the same as that in the baseline case with default GPU configuration (i.e. 32 dual-ported 4 KB banks per SM) for the fair comparison. The partition of CMOS- and TFET-based registers is important to the effectiveness of our proposed MEM_RA mechanism since the partition can not be changed after fabrication. Fabricating the sizeable TFET registers forces the use of TFET registers when there are insufficient CMOS registers, it reduces power by sacrificing the high computational throughput; while the small TFET registers cannot provide enough TFETs for the energy saving purpose. In the ideal case, the number of CMOS-based registers should perfectly matches their utilization under the impact of MEM_RA, which is largely determined by the warp waiting time during the off-chip memory accesses. The quantity of TFET registers may be more than required in the ideal case, it is better to have idled TFET-based instead of CMOS-based registers considering the extremely low leakage power consumed by TFET circuits.

Fig. 4 shows the percentage of the warp stall time to its total execution time in various types of GPGPU benchmarks.(the detailed experimental setup is described in Section 5). In the

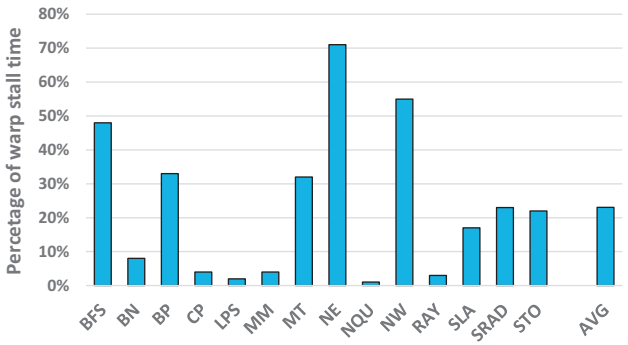


Fig. 4. The percentage of the warp stall time caused by the off-chip memory accesses. It is near zero in CP, LPS, MM, and RAY.

computation-intensive benchmarks (e.g. CP, LPS, MM, and RAY), there are few memory accesses, and the warp stall time is very close to zero that means the TFET-based register should not be applied to extend the normal execution. To the contrary, the numerous memory transactions in the memory-intensive benchmarks (e.g. BFS, BP, MT, NE, and NW) causes much longer warp stall time to allocate TFET-based registers for energy saving. On average across all the benchmarks, the stall time is around 22%. In other words, CMOS registers should be applied in the remaining 78% of the execution time. Therefore, in order to choose a starting point to explore the optimal partition of CMOS- and TFET-based registers, CMOS-based registers are designed to account for 78% of the total registers, and the remains are TFET. We then perform detailed sensitivity analysis on varying the percentage of CMOS registers (e.g. 62.5%, 78% and 87.5%) in the hybrid design, and find that adopting 100 KB CMOS-based registers (78% of total register file) and 28 KB TFET-based registers (22%) in our hybrid register file is the optimal design regarding to the total energy saving and performance overhead. Details about the sensitivity analysis are shown in Section 6.1.2.

In GPU SM, the per-block resources (e.g. registers, shared memory) are not released until all the threads in the block finish execution, they limit the number of blocks that can simultaneously run in the SM. Different per-block resources become the bottleneck for kernels that have different resource requirements. The bottleneck structure is prone to be fully utilized while other structures are usually underutilized. Therefore, a portion of CMOS registers may be free through the entire kernel execution, leading to the considerable leakage power consumption. In [2], the power gating technique has been introduced into GPU SM to remove leakage. We apply it to power off the unused CMOS registers in SMs. Information such as the maximum number of threads allocated to each SM, and the quantity of physical registers required per thread can be easily obtained during the kernel launch process. Hence, the total register utilization would not exceed the product of those two factors. The requirement on CMOS registers can be estimated by scaling down the total register utilization to 78%, and the power gating is enabled on the remaining idled CMOS registers for a long time until the kernel completes. The energy and time overhead caused by the power gating is negligible with regard to the large power reduction by keeping those registers in the power-gated mode during the entire kernel execution period.

4.1.2.2. Structure of MEM_RA. Fig. 5 demonstrates the implementation of our proposed memory contention-aware TFET-based register allocation in the hybrid register design. A counter(❶) is attached to each warp slot in the warp scheduler. When a warp encounters an off-chip memory access, its counter is re-set as zero and starts the auto increase every cycle to record the memory access latency. Upon the completeness of the memory transaction, the cycle number stored in the counter is sent to an ALU(❷) for the warp stall time prediction, meanwhile, the sampled memory access cycles with the static information (i.e. threshold latency, referred access latency) are also sent to the ALU. The output is written back to the counter, it will be read when the warp enters into the pipeline, and a larger-than-one value in the counter implies the necessity of writing to the TFET-based register. In [4], Gebhart et al. found that 70% of the register values are read only once in GPGPU workloads. It implies that most TFET register values are no-reused, therefore, renaming the destination register to the TFET register usually causes 2-cycle extra delay: one additional cycle during the value write back, and another one when it is read by a subsequent instruction. As can be seen, one TFET register allocation takes two cycles of the warp stall time in most cases. And the counter value will decrease by two upon a successful TFET register allocation. Note that the counter decrease is just used to estimate the possible delay to the warp when renaming to the TFET registers. For TFET registers being read multiple times, the warp stall time will be taken more than two cycles. Moreover, the counter auto-increases occurs at warp waiting time while its value decrease is performed at the normal execution time, there is no overlap between the counter auto-increase and decrease processes.

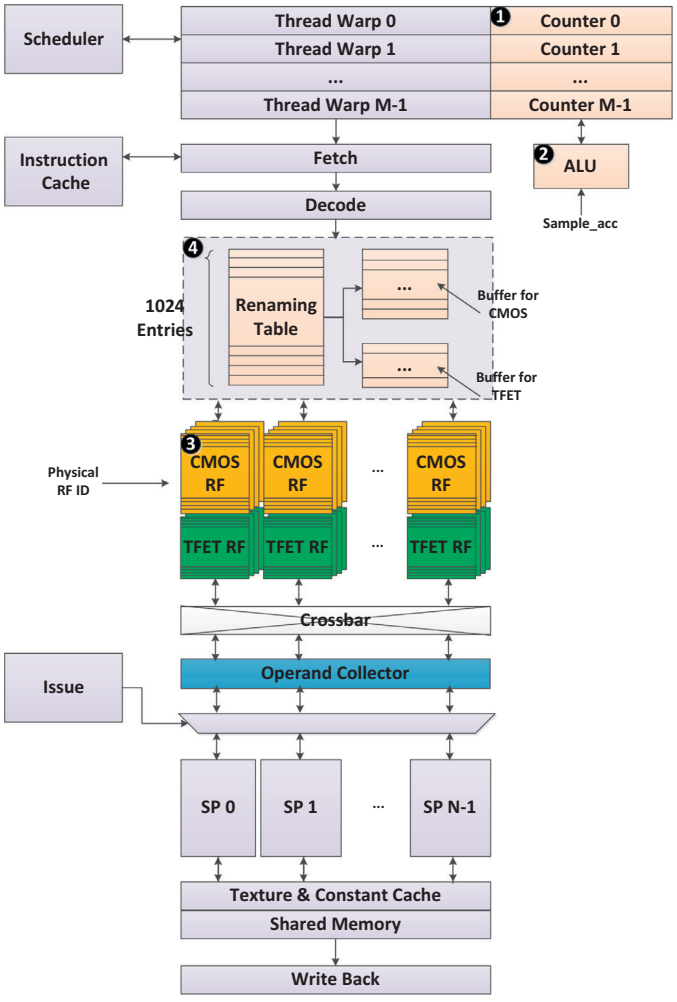


Fig. 5. Memory contention-aware TFET-based register allocation.

Considering that the ALU is used once a warp completes a memory instruction, and the major computation in it is division (as shown in Eq. (1)) lasting for tens of cycles, we set the referred access latency as 2 to power of n and translate the division into

logical shift. It will operate based on the product of the sampled memory access latency and the previous stall time. We performed the detailed sensitivity analysis on the referred access latency, and found that MEM_RA achieves optimal trade-off between power and performance when setting it as 2^7=128 cycles. Note that the warp stall time estimation occurs in parallel with the write back stage, it does not introduce any extra delay to the critical path in the pipeline.

As Fig. 5 shows, register file(❸) is partitioned into CMOS-based and TFET-based registers in the unit of entries in each register bank, and each physical register entry has a unique identification number. For example, in our design, 78% register file is CMOS-based and 22% is TFET-based. Hence, we set the first 200 entries of each register bank as CMOS-based registers and the rest 56 entries as TFET-based registers. Two power supply lines are used to support the high (low) voltage operations on CMOS (TFETs) registers. The register renaming stage(❹) is added into the SM pipeline (shown as the dotted rectangle), during which the register renaming table is accessed. It is indexed by the warp ID and register number encoded in the instruction, and each entry holds the corresponding physical register ID which will be used for register access in the following stage. Two FIFO buffers are attached to the renaming table to keep the released CMOS and TFET register vectors, respectively. The top register in each buffer is consumed for the renaming, while the bottom is filled by the newly released register. In the case that a CMOS register is requested while the buffer for CMOS registers is empty, the buffer for TFET registers will provide a free TFET register instead, and vice versa. Note that there is always at least one free CMOS/TFET register available for renaming since the required resources have already been well estimated when the block is assigned to the SM. Note that for each register value, the physical register renaming (to either CMOS-based or TFET-based regsiter) happens at the first write of the value, and the physical register will be released at the last read of the value. Thus, there is no register value migration between the two types of registers.

4.1.2.3. Hardware and power overhead. The major hardware added into the SM is the register renaming pipeline stage including the register renaming table, two buffers for the released CMOS and TFET register vectors, and some simple combinational logics. In order to keep the renaming information for all physical registers, the number of entries in the renaming table is equal to the amount of register entires which is 256 in our default GPU configuration. Similarly, the total size of the two buffers is 256 as well. Each entry in those three structures contains 10 bits. The hardware in the renaming stage causes around 1% area overhead to the register files in the SM. In addition, to predict the warp stall time, thirty-two 11-bit counters (we set the maximum memory access time as 2048 cycles), and the unit performing simple integer arithmetic and logic operations are added in the SM. The overall hardware overhead to the SM register files is 2%. We develop the power model (including both dynamic and leakage power) for the added hardware, and find that it induces around 1.2% power overhead to the register files by running a large set of GPGPUs benchmarks.

4.2. TFET-register-utilization-aware block allocation (TUBA) and TFET-register-request-aware warp scheduling (TRWS)

4.2.1. The inadequate TFET registers for memory-intensive benchmarks under MEM_RA

It is obvious that the size of the TFET registers is fixed at the chip fabrication time, but benchmarks exhibit largely diversified requirements on the TFET registers. Especially for memory-intensive benchmarks, they encounter frequent long-latency memory accesses, and are eager to have a quite large TFET register file

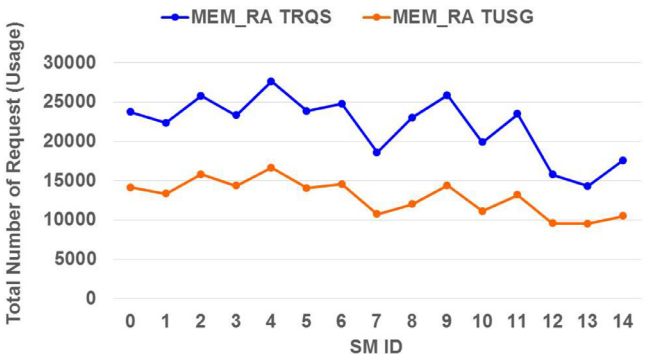


Fig. 6. The total number of requests for the TFET-based registers (TRQS) vs. the total number of TFET-based registers that are used (TUSG) in each SM during the entire execution of BFS.

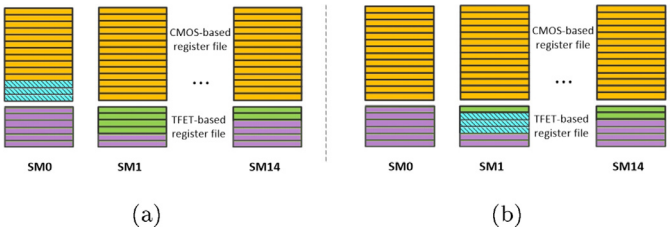


Fig. 7. (a) SMs suffer long pipeline stall due to the severe memory contentions. (b) Leveraging TFETs to absorb the long pipeline stall and alleviate memory contentions.

to absorb the long pipeline stall time. We collect the total number of requests for the TFET-based registers (TRQS) and the total number of TFET-based registers that are used (TUSG) under MEM_RA for each GPGPUs benchmark across its entire execution. As we expected, the TRQS is much higher than the TUSG for the memory-intensive benchmarks. For example, Fig. 6 shows the large difference between the total numbers of TRQS and TUSG in each SM during the execution time of a representative memory-intensive benchmark *BFS*. The ratio between the total TUSG and the total TRQS is only 59% on average across all SMs. Note that the total number of TUSG is much higher than the number of TFET registers since one physical TFET register can be allocated and released many times, and each time that it is allocated is considered as one use in TUSG. As can be seen from Fig. 6, although 22% of the registers are TFET in our hybrid register file design, they are far from enough to satisfy the high TFET register requirements in BFS. The inadequate TFET registers seriously limit the opportunities in effectively reducing the energy consumption when applying the proposed MEM_RA for the memory-intensive benchmarks. Techniques are desired to efficiently utilize the limited TFET register resources to maximize the energy savings.

4.2.2. TFET-register-utilization-aware block allocation (TUBA)

Note that Fig. 6 only exhibits the overall TFET registers usage per SM across the entire benchmark's execution, it is unable to provide the run-time TFET registers usage information. We sample the number of free TFET registers every 1K cycles for each SM and observe that some SMs have quite few free TFET registers while others have a plenty of free TFET registers at the same time. Fig. 7(a) uses an example to illustrate this phenomenon. During certain short period, SM0 that suffers the extremely long memory access latency due to the severe memory contentions will use all the available TFET registers and even require more which are highlighted by the blue color in Fig. 7(a), while SM1 requests few TFET registers and has quite a lot of free TFET registers which are highlighted by the green color in Fig. 7(a). Such significantly un-

balanced TFET registers utilization at the run-time motivates us to explore a novel technique to well control the TFET register requests across SMs so that the free TFET registers in some SMs can be effectively used to serve the excessive requests from other SMs, as shown in Fig. 7(b). By doing this, the overall TFET registers utilization is increased to gain more energy savings.

A naive method to balance the TFET registers usage across SMs is to migrate a number of threads from SMs that lack of TFET registers to those with free TFET registers. However, thread migration induces substantial performance and energy overheads. Managing the number of blocks assigned to each SM at the block allocation stage is a promising approach to control the number of threads thus the number of TFET requests per SM. In this study, we propose the TFET-register-Utilization-aware Block Allocation (TUBA) to dynamically control the number of blocks in each SM and balance the TFET utilization across SMs.

When the number of available TFET registers in a SM is lower than a certain threshold (named as dw_th), it implies that there are excessive threads in the SM which greatly increase the TFET registers requirements. Our TUBA technique will decrease the number of blocks in that SM. On the other hand, when the number of available TFET registers increases and becomes higher than another threshold (named as up_th), TUBA will relax the block limitation which will gradually increase to the maximal number of blocks that a SM can support in the default GPGPUs configuration. By doing this, fewer blocks are allocated to SMs with few idle TFET registers, while SMs with many idle TFET registers are able to take more blocks.

The effectiveness of TUBA is largely determined by the selection of dw_th and up_th. We perform the comprehensive sensitivity analysis (e.g. 10%, 15% and 20% for dw_th and 30%, 40%, 50% and 60% for up_th) and find that setting dw_th and up_th as 15% and 40% of the total number of TFET registers, respectively can achieve the optimal results. We then introduce one counter into each SM to monitor the number of free TFET registers (named as n_tfet). When a block finishes in the SM, the n_tfet is collected and compared with the dw_th. If $n\_tfet < dw\_th$, the block limitation is reduced by 1 for that SM which implies it will not accept a new coming block even one block just finishes and releases the resources. Note that the block limitation will not further decreases when reaching 1 to ensure that there is always at least one block executing in the SM. On the other hand, if $n\_tfet > up\_th$, the block limitation increases by 1 and the SM is able to take new blocks. Similarly, the block limitation will not increase when reaching the maximal number of blocks that a SM can support that is determined by the GPU architecture (e.g., 8 in our baseline GPU configuration).

4.2.3. TFET-register-request-aware warp scheduling (TRWS)

Although TUBA can balance the TFET utilization across SMs, the TFET requests may be still much higher than the available TFETs in the SM. Since the number of physical TFET registers is fixed, it is important to increase the renaming frequency for each physical TFET register so that it is able to serve more TFET register requests during the entire execution time. To achieve this goal, a simple but effective method is shrinking the period from each write of a new value to its last read in the same physical TFET regsiter. However, such period is excessively extended in registers due to the nature of the GPUs microarchitecture design. In each SM, warps are interleaved at cycle level. As a result, the time interval between the execution of two consecutive instructions from the same warp can be extremely long. Take the warp 0 in Fig. 8(a) as an example, the execution of its instructions n and $n+1$ are greatly separated by the execution of other warps' instruction under the round-robin warp scheduling policy. Assume the destination register $R2$ in instruction n of warp 0 is renamed to a TFET register and its last

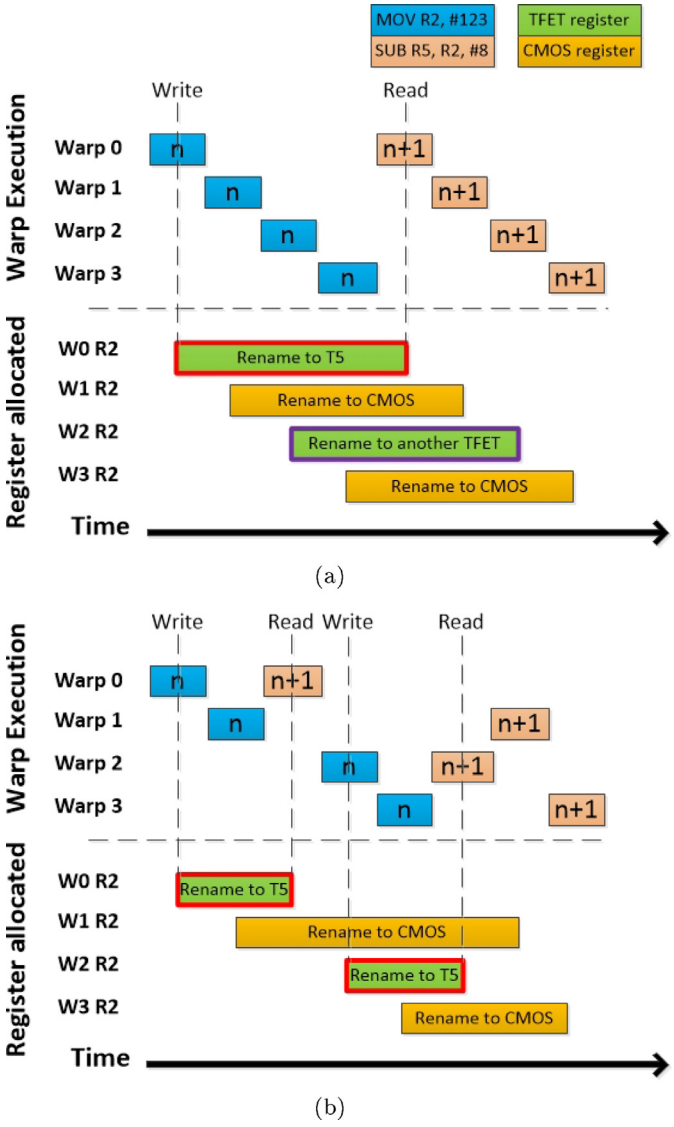


Fig. 8. An example of warp execution and the TFET register renaming frequency under (a) round-robin scheduling policy (b) the proposed TFET-register-request-aware warp scheduling policy.

read occurs in instruction $n+1$, as Fig. 8(a) shows, the period between the write and last read is dramatically extended due to the interleaved warp execution. In other words, one TFET register (e.g., $T5$ in Fig. 8(a)) is only able to serve one TFET request (e.g., $R2$ of warp 0 in Fig. 8(a)) in a long period (highlighted by the red rectangle in Fig. 8(a)). Although Fig. 8(a) adopts the round-robin policy to show the example, we observe the similar case when applying other common warp scheduling policies (e.g., greedy-then-oldest).

In this study, we further develop a new TFET-register-Request-aware Warp Scheduling (TRWS) policy to improve the TFET utilization at the intra-SM level. Therefore, when combined with the TUBA, we are able to hierarchically serve the maximal number of TFET register requests at both coarse-grained (i.e., inter-SM by TUBA) and fine-grained (i.e., intra-SM by TRWS) levels for the optimal energy reductions. In our TRWS, warps that have TFET register requests will be given a higher issue priority than those that do not have TFET register requests. For example, since the instruction n of warp 0 in Fig. 8 requests to rename $R2$ to a TFET register $T5$, warp 0 will be granted higher priority, and its instruction $n+1$ is able to finish the last read of the value much earlier (as shown in Fig. 8(b)). Therefore, the same TFET register $T5$ is released quickly

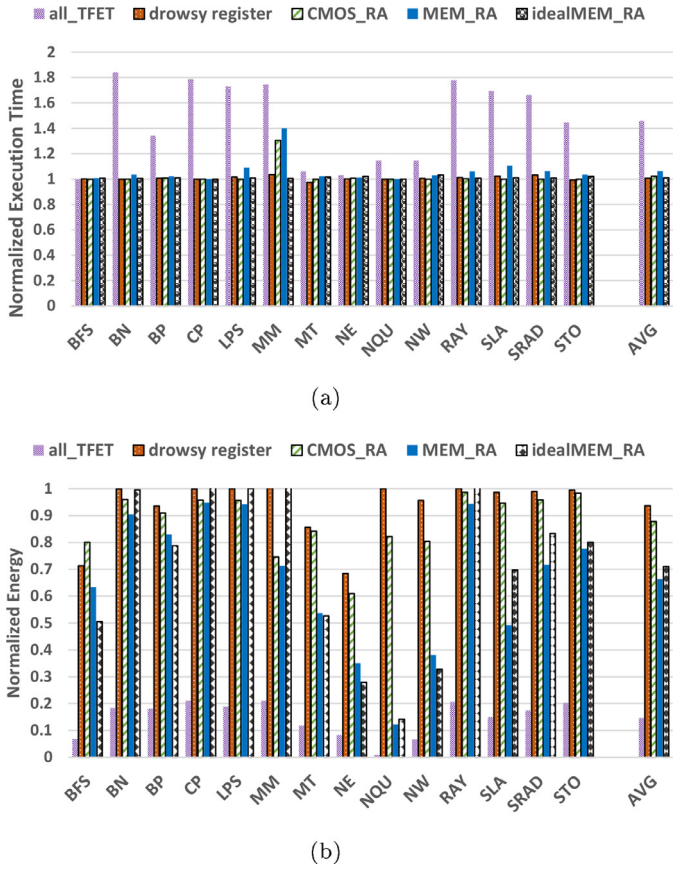


Fig. 9. The (a) normalized execution time (b) normalized energy consumption when running benchmarks under several power optimization techniques.

and further re-allocated to warp 2 in a short period (highlighted by the red rectangle in Fig. 9(b) that is comparable with the period highlighted in Fig. 8(a)). As can be seen, TRWS significantly shrinks the interval between a value write and its last read in a physical TFET register such that it can support more TFET register requests in a given amount of time, and the overall number of satisfied TFET requests is greatly improved for the maximal energy savings. To implement TRWS, the cycle counter introduced by MEM_RA (as shown in Fig. 5) is reused. During the warp scheduling stage, each warp's cycle counter will be sent to the warp selection logic, and ready warps with cycle counters greater than zero will be issued earlier.

5. Experimental methodology

We evaluate our proposed technologies by modifying the cycle-accurate, open-source, and publicly available simulator GPGPU-Sim [20]. We implement the hybrid register file and insert the register renaming stage into SM pipeline. We also model all hardware and power overheads induced by our hybrid architecture. Table 1 shows the machine configurations for the baseline GPGPU.

We collect a large set of available GPGPU benchmarks from Nvidia CUDA SDK [22], Rodinia Benchmark [23], Parboil Benchmark [24] and some third party applications [37]. In total, we study 14 available benchmarks which are described in Table 2. These benchmarks show significant diversity according to their kernel characteristics, branch divergence characteristics, memory access patterns, and so on.

To evaluate the energy consumption of our proposed technologies, we model the register file as banked dual-port (1w/1r) SRAM blocks based on the energy analysis tool CACTI [21]. Each register bank provides 256 register entries, and each of which includes four 32-bit CMOS- or TFET-based registers. We refer to the parameters from Singh et al. [14] to implement the 6T TFET-based and 6T CMOS-based SRAM cell. We also include the 4×4 crossbar interconnect network in our power model. Each crossbar link is as wide as the register banks. The parameters of the crossbar network are determined by the number of register banks per cluster and the number of operand collector units. Similar to the power modeling methods adopted in [35,36], we further model the operand collector units, renaming table and renaming buffer as the CMOS SRAM array. We implement the high supply voltage as 0.7 V for CMOS and low supply voltage as 0.3 V for TFET. The result of the power model has similar energy consumption of memory cell with previous work [19,26]. The read/write times to CMOS/TFET-based registers and the total execution time are collected from the modified GPGPU-Sim to evaluate both dynamic and leakage energy consumption.

Table 2
Lists of benchmarks.

Benchmarks	Abbr.	Blocks/SM
Breadth First Search	BFS	4
Binomial Options	BN	4
Backpropogation	BP	4
Coulombic Potential	CP	6
3D Laplace Solver	LPS	6
Matrix Multiplication	MM	4
Matrix Transpose	MT	4
Nearest Neighbo	NE	8
N-Queens Solver	NQU	1
Needleman-Wunsch	NW	7
Ray Tracing	RAY	3
Scan of Large Arrays	SLA	8
SRAD1	SRAD	3
storeGPU	STO	1

6. Results

6.1. The effectiveness of MEM_RA

6.1.1. The energy and performance results

In order to justify the effectiveness of MEM_RA, we compare it with several power reduction techniques. The baseline case studied in this paper is employing only CMOS-based registers and power gating the unused registers during the kernel execution. Another naive mechanism for power saving is simply applying TFETs to all SM registers, it is named as all_TFET. In previous work, the drowsy cache has been proposed to reduce the cache leakage power [25]. Similarly, registers belonging to a warp can be put into the sleep mode when the warp stalls in the pipeline, but it takes couple of cycles to wake them up for further accesses. We also investigate the effect of drowsy register from the performance and power perspectives. In the hybrid register design, the long access time to TFET registers may largely degrade the performance when they are randomly used. A straightforward technique to maintain performance is to avoid the allocation of TFET registers if possible. In other words, the CMOS register is selected for renaming as long as there is any one free. We name this technique as CMOS_RA, it is applied on the hybrid 25K CMOS registers and 7K TFET registers. Note that the power gating technique is integrated into drowsy register and CMOS_RA, respectively, for the fair comparison. Since TFET has extremely low leakage power, the power gating is not triggered in all_TFET mechanism.

As discussed in Section 4.1.2, ideally, the size of CMOS registers would exactly match their usage. We further investigate the effectiveness of MEM_RA when the CMOS registers size is set ideally, called idealMEM_RA. (We name the MEM_RA using 25K CMOS and

7K TFET registers as MEM_RA for short.) Since benchmarks exhibit different memory access patterns, their requirements on CMOS registers vary greatly. Although designers rarely fabricate a GPU with certain number of CMOS (TFET) registers to specifically satisfy a single benchmark's requirement, the results of idealMEM_RA provide a more accurate evaluation on the capability of MEM_RA on power optimizations while maintaining the performance.

Fig. 9 describes (a) the execution time and (b) the register file energy consumption when running the investigated benchmarks under the impact of several power reduction techniques described above. The results are normalized to the baseline case. Note that the performance and energy overhead caused by each technique is also included in the results. As Fig. 9(a) shows, all_TFET hurts the GPU performance significantly, the execution time is almost doubled in several benchmarks (e.g. BN, CP, MM, and RAY). On average, all_TFET degrades the performance by 45%. Although it reduces the energy consumption significantly (total energy decreases to 14% as shown in Fig. 9(b)), it is not worth to scarify such large portion of throughput to achieve the low energy consumption. Interestingly, the kernel execution time under drowsy register mechanism remains the same although there is time overhead to wake up registers staying in the sleep mode, because the wake up time is trivial with regard to the hundred-cycle long memory access. Moreover, the energy reduction achieved by drowsy register is small, only around 7%.

CMOS_RA is performance friendly which causes 2.3% performance penalty on average. Because it uses the CMOS registers in majority of the time, and there is no performance loss when the benchmark needs less than 25K registers. However, the performance penalty is high for benchmarks requesting a large amount of registers, as TFET registers are consumed in that case. For example, the execution time for MM increases 30% under CMOS_RA because the RF utilization in that benchmark is 100%.

As Fig. 9 shows, the energy reduction under CMOS_RA is 12.3%, while MEM_RA is able to achieve 34% energy savings with similar performance loss. Different from CMOS_RA, MEM_RA intelligently migrates the resource usage from CMOS to TFET registers which reduces the total dynamic energy. Meanwhile, the extra access delay in TFETs absorbs the warp waiting time and prevents the interferences among memory requests which minimizes the impact on performance. Especially for the memory-intensive benchmarks, such as the BFS, BP, MT, NE, and NW, MEM_RA generally reduces the energy by 47% with only 1.5% performance loss.

One may notice that MEM_RA introduces the long execution time in MM as well. Because MM fully utilizes the RF resources and contains quite few memory accesses to trigger the memory-contention aware TFET register allocation. As Fig. 9(a) demonstrates, the performance of MM maintains the same under idealMEM_RA since the GPU will be equipped with all CMOS registers if running such type of benchmarks, and it cannot reduce the energy. On average, idealMEM_RA slightly outperforms MEM_RA on performance but meanwhile, obtains less energy savings. In summary, MEM_RA successfully explores the energy-efficient GPGPUs and its effectiveness is quite close to that of idealMEM_RA.

The performance degradation in SLA is noticeable under MEM_RA and idealMEM_RA. Because they use the last memory access latency to predict the warp waiting time and enable the TFET register allocation correspondingly, the prediction accuracy is affected when the next memory access pattern differs greatly from the last one. As a result, the TFET registers are excessively utilized which hurts the performance. Generally, the last value prediction mechanism achieves pretty high accuracy for most benchmarks and helps MEM_RA to minimize the performance penalty.

We further split the normalized overall energy obtained by MEM_RA into the dynamic and leakage portions and present them in Fig. 10. The energy partition under the baseline case and

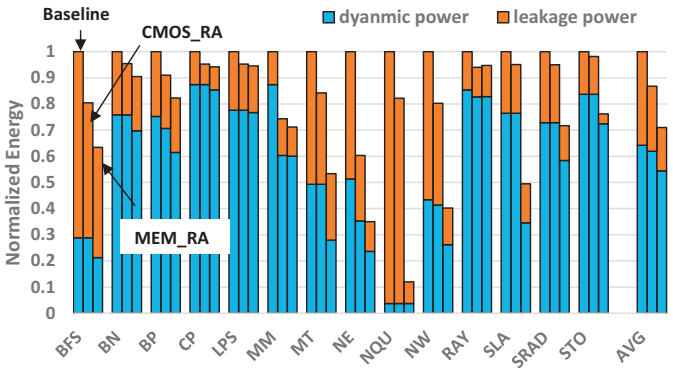


Fig. 10. Dynamic and leakage energy consumptions under baseline case, CMOS_RA, and MEM_RA.

CMOS_RA is also included in the figure. As it shows, CMOS_RA can barely optimize the dynamic power since the CMOS register are frequently accessed. MEM_RA exhibits strong capability in reducing not only leakage but also dynamic energy. On average, the dynamic energy reduction compared to the baseline case is 14%, while the leakage decreases 19%.

We also observe that the TFET-based registers only consume 4.2% of the overall register files energy consumption, and it increases to 7.9% for memory-intensive benchmarks. As can be seen, MEM_RA can leverage the high energy efficiency of TFET to reduce the overall register file energy consumption.

6.1.2. Sensitivity analysis on CMOS-TFET parition percentage

Fig. 11 shows the effectiveness of MEM_RA when 62.5%, 78% and 87.5% of the overall register files are CMOS based. The 78%–22% partition is our proposed partition of CMOS- and TFET-based register file. We consider the 87.5%–12.5% partition of CMOS and TFET because it is performance friendly and generally preferred by computation-intensive benchmarks. We also investigate the 62.5%–37.5% partition which saves more energy and preferred by memory-intensive benchmarks. We normalize the results to our baseline case. As it shows, in general, when the number of TFET-based register increases (the number of CMOS-based register decreases accordingly), the energy saving increases while the performance degrades. Especially, the performance degradation in CP is significant when the percentage of CMOS registers decreases from 78% to 62.5%. This is because the 62.5% CMOS-based registers are not enough for the CMOS register requirements, instead, some TFET registers are used to satisfy those requirements which causes extra delay and hurts the performance. Even though increasing the number of TFET registers can be considered as a simple approach for energy saving, its impact on the performance degradation cannot be ignored. Therefore, we set the 78% of total register file as CMOS-based registers and the rest of 22% as TFET in our default MEM_RA which is the optimal partition across various types of benchmarks.

6.1.3. Prediction model validation

In this subsection, we further analyze the prediction accuracy of the stall cycles in MEM_RA. We record the predicted memory stall cycles after each memory access, and compare each predicted value with next sampled memory access cycle to calculate the prediction error. For each benchmark, we collect the prediction error for each memory access and calculate the mean across all errors.

Fig. 12 shows the prediction errors of the stall cycles in MEM_RA for various investigated benchmarks For the memory-intensive benchmarks (e.g. BFS, BP, MT and NE), our prediction model achieves high accuracy (i.e., low prediction error as shown

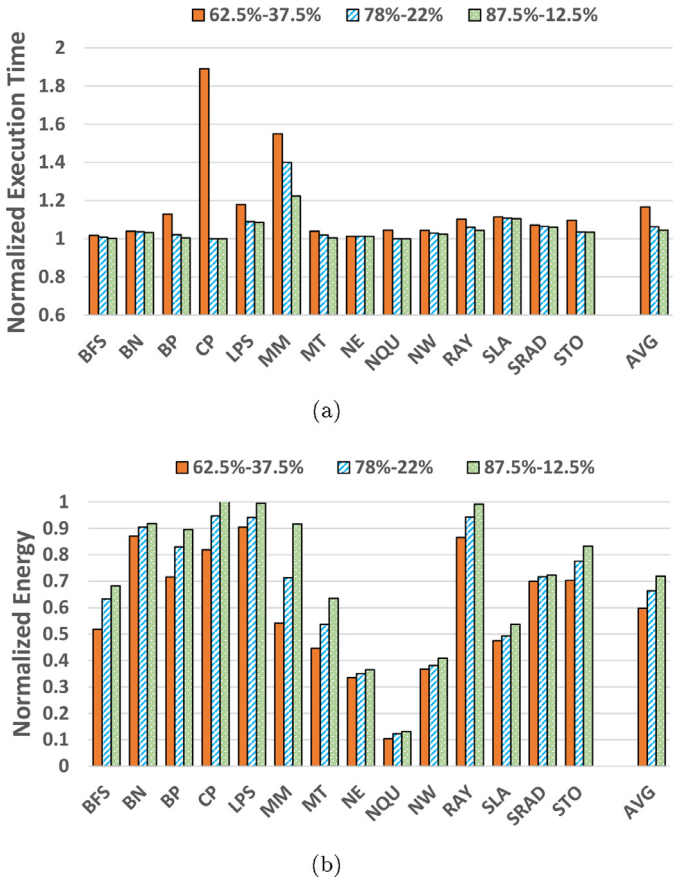


Fig. 11. The (a) normalized execution time (b) normalized energy consumption when running benchmarks under different CMOS-TFET partition percentages.

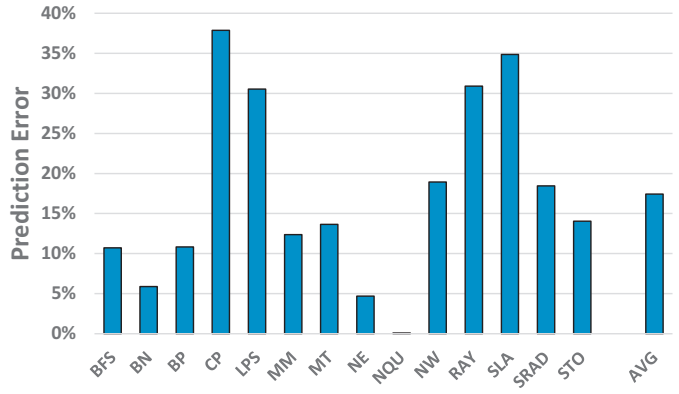


Fig. 12. Prediction errors for each investigated benchmark.

in the figure). It is because these benchmarks have relative constant memory access patterns and our MEM_RA can accurately capture the memory pattern. However, for computation-intensive benchmarks which have few memory accesses, the accuracy of our model is relatively unstable. These benchmarks have many irregular memory access patterns that increase the difficulty for achieving very high accurate prediction via using a simple prediction model as shown in Eq. (1). A more complicated prediction model that can capture the irregular memory access patterns may greatly help to improve the prediction accuracy, but will cause high area and power overhead which largely degrades the benefits achieved by MEM_RA. We thus select the simple but effective prediction model (as shown in Eq. (1)) by taking the complexity, overheads, and overall energy improvement into the consideration.

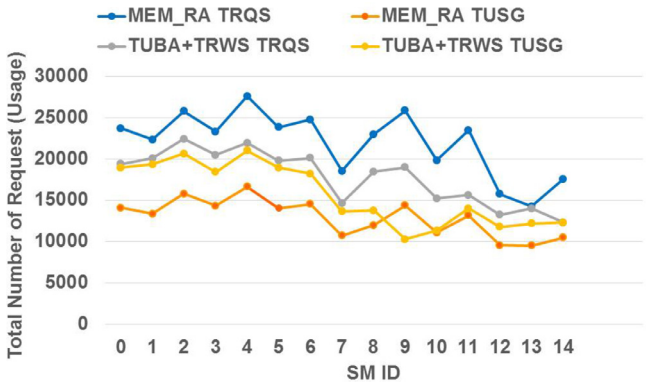


Fig. 13. The total number of requests for TFET-based registers (TRQS) and the total number of TFET-based registers that are used (TUSG) under MEM_RA and MEM_RA+TUBA+TRWS across all SMs when running BFS.

Overall, as shown in Fig. 12, the averaged prediction error across all investigated benchmarks is 17.8%. Considering that the major contributor to the averaged prediction error is the computation-intensive benchmarks which have much fewer memory accesses than the memory-intensive benchmarks, we expect our model is able to achieve high prediction accuracy for major memory accesses.

6.2. The effectiveness of TUBA and TRWS

6.2.1. The TFET-based registers utilization

The major improvement of TUBA and TRWS over MEM_RA is serving more TFET register requests to achieve higher energy savings, which can be translated to the increased TFET based registers utilization. In this subsection, we first evaluate the effectiveness of TUBA and TRWS by investigating the TFET registers utilization. Fig. 13 compares the total number of requests for the TFET-based registers (TRQS) and the total number of TFET-based registers that are used (TUSG) under MEM_RA and the combined TUBA and TRWS on top of MEM_RA, across all SMs when running BFS. As it shows, on average, MEM_RA+TUBA+TRWS is able to reduce the total TRQS around 18% and increase the total TUSG around 16% when compared with MEM_RA. Therefore, the total TUSG is approaching the total TRQS under our proposed MEM_RA+TUBA+TRWS technique. Note that the decreased total requests for TFET registers under the MEM_RA+TUBA+TRWS technique is caused by the limited number of blocks per SM. Since the overall TFET registers utilization increases for all SMs, it has no negative impact on reducing the energy consumption. In addition, the TRQS variations across SMs are becoming smaller which confirms that our TUBA technique effectively balances the TFET registers requests among SMs.

Fig. 14 shows the ratio of the overall TFET-based registers usage to the overall requests when running memory-intensive benchmarks. Note that in certain memory-intensive benchmarks (e.g., NE and NW), the size of the TFET registers already satisfies the TFET registers requests under MEM_RA. Further triggering TUBA+TRWS will not help with the TFET registers utilization, and Fig. 14 does not include the results for those benchmarks. As Fig. 14 shows, on average, MEM_RA+TUBA+TRWS shows the strong capability to improve the TFET registers utilization around 20% comparing with MEM_RA.

6.2.2. The energy and performance results

To evaluate the effectiveness of TUBA+TRWS, we also compare it with the recently explored block allocation and warp scheduling techniques, which are DYNCTA block allocation and two-level warp scheduling. DYNCTA is proposed by Kayiran et al. [31] which limits

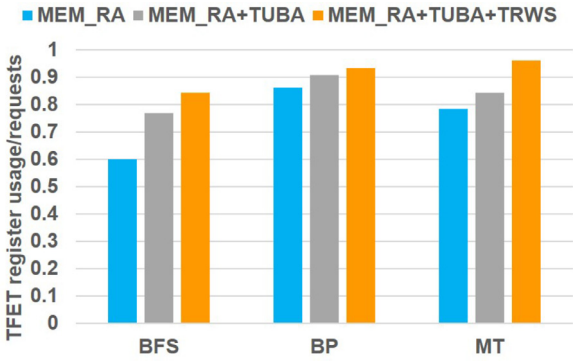


Fig. 14. The TFET-based registers utilization for memory-intensive benchmarks.

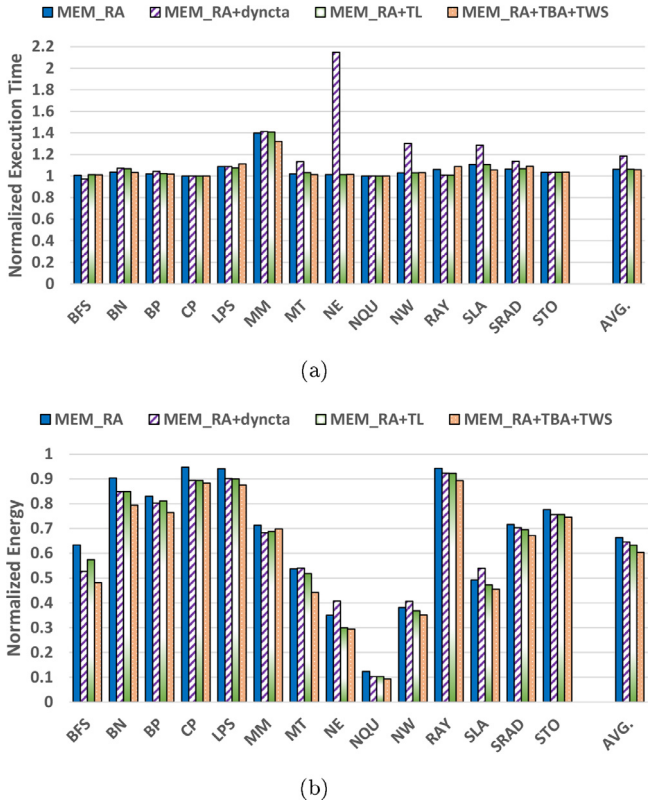


Fig. 15. (a) Normalized execution time and (b) Normalized energy under MEM_RA, MEM_RA with DYNCTA, MEM_RA with two-level warp scheduling, and MEM_RA with TUBA+TRWS.

the number of blocks in the SM to reduce the cache, network and memory contentions. The two-level warp scheduling (TL) is proposed by Narasiman et al. [30] which splits the execution warps into two-level fetch groups to hide the long off-chip memory access latency. Both techniques mainly target on boosting the GPUs performance but ignore the energy savings. We thus apply them on top of MEM_RA to reduce the energy consumption, and conduct the fair comparison with our TUBA+TRWS technique. Fig. 15 shows the execution time and the RF energy when introducing various techniques into MEM_RA, which are MEM_RA combined with DYNCTA, MEM_RA combined with the two-level scheduling, and MEM_RA combined with our proposed TUBA+TRWS. The results are normalized to the baseline case.

As Fig. 15 shows, when combined with MEM_RA, the DYNCTA can not effectively improve the performance, and even hurt the performance. On average, MEM_RA+DYNCTA increases the execution time around 18% but achieves the similar energy savings as MEM_RA. Especially, the performance degradation in NE is even 100% under MEM_RA+DYNCTA. This is because the DYNCTA reduces the thread-level parallel based on the memory stall time in each SM, and MEM_RA allocates TFET registers to further slow down the warps with long memory access latency. Combining these two techniques leads to few active warps to hide the long execution time of slow warps, thus, degrades the performance. The two-level warp scheduling does not induce extra energy benefits when combined with MEM_RA either. This is because the two-level technique also tries to separate the warps' execution progress to alleviate the memory contentions, which has similar effect as MEM_RA. Finally, as shown in Fig. 15, our TUBA+TRWS technique is able to maintain the performance and meanwhile, achieve additional 4% RF energy savings on average across all benchmarks comparing with MEM_RA. In other words, our MEM_RA+TUBA+TRWS obtains 40% RF energy reduction with small performance loss. For the memory-intensive benchmarks that require numerous TFET registers (e.g., BFS, BP, MT), it saves additional 7% RF energy; especially for BFS, it gains additional 15% RF energy savings comparing with MEM_RA.

6.3. Discussions

To further explain the benefit of our technology, we assume the register file consumes 15% of SM power [36,39]. Our evaluation result shows that our technology achieves 40% and 54% register file energy saving across all benchmark and memory-intensive benchmarks, respectively. Such a mount of energy saving contributes to around 7% energy savings to the entire SM which is already considered as the noticeable optimization as discussed in [4]. Even though our technology exhibits slight performance penalty (e.g. 6%), making the integrated system more power-friendly not only reduces the working temperature but also breaks the power-wall to accompany the high performance architecture design in the future. Meanwhile, our evaluation shows that our technology causes negligible performance loss for memory-intensive benchmarks (e.g. 1.5%). Due to the ever-increasing of data size for modern applications, we expect our technology can provide more energy reduction and lower performance penalty for next-generate applications.

7. Related work

There have been several studies on building hybrid storage-cell based structure and furthermore, heterogeneous multi-core processors based on CMOS and TFETs to achieve the good trade-off between performance and power [13,19,26,27]. For instance, Narayanan et al. [19] developed the hybrid cache architecture that uses a mix of TFET and the non-volatile memory. Swaminathan et al. [13] proposed to replace some of the CMOS cores with TFET alternatives, and dynamically migrate threads between CMOS and TFET cores to achieve significant energy savings with negligible performance loss. We build the hybrid registers in GPGPU and leverage its unique characteristics to fully explore the benefit of TFETs for the energy-efficient GPGPU design.

Many methodologies have been proposed recently to reduce the GPGPU registers dynamic power. Gebhart et al. [4] proposed register file caching and two-level thread scheduler to reduce the number of reads and writes to the large main register file and save its dynamic energy. The authors further extended their work to the compiler level and explored register allocation algorithms to improve register energy efficiency [5]. Yu et al. integrated embedded DRAM and SRAM cells to reduce area and energy [3]. In addition, several works have been done on GPGPU register leakage power optimization. Chu et al. [6] explored the fine granularity clock gating scheme for registers. Wang et al. [2] adopted the power gating technique at architecture level for leakage reduction on GPGPUs.

Mohammad et al. presented a gating-aware two-level warp scheduler(GATES) which prioritizes warps based on the instruction type. By doing that, it increases the power gating windows and save the leakage power [33]. Qiumin et al. proposed a new divergence pattern aware warp scheduler. it prioritizes warps with the same divergence pattern so as to create long idle window and provide opportunities to adopt power gating at the lane level [34]. There are also several works to reduce both dynamic and leakage power. For instance, Mohammad et al. [32] proposed a tri-modal register file which can switch between ON, OFF and drowsy stages to reduce leakage power, and utilize the active mask during branch divergence to disable unnecessary register activity to reduce the dynamic power. Our technique is orthogonal to the techniques discussed above.

8. Conclusions

Modern GPGPU employs the fine-grained multi-threading among numerous active threads which leads to the large register files consuming massive dynamic and leakage energy. Exploring the optimal energy savings in the register file becomes the critical and first step towards the energy-efficient GPGPUs. The conventional method to reduce dynamic energy is to scale down the supply voltage which causes substantial leakage in CMOS circuits. The TFETs are the promising candidates for low voltage operations regarding to both leakage and performance. However, always executing at the low voltage (so that low frequency) will result in significant performance degradation. In this study, we propose the hybrid CMOS-TFET based register files. We leverage the unique characteristics of GPUs during the off-chip memory accesses, and explore the memory contention-aware TFET register allocation (MEM_RA) to make use of TFET registers in alleviating the memory contentions, and meanwhile gaining the attractive energy optimization. We further observe the insufficient TFET-based registers for the memory-intensive benchmarks when applying MEM_RA, and propose the TFET-register-Utilization-aware Block Allocation (TUBA) and TFET-register-Request-aware Warp Scheduling (TRWS) on top of MEM_RA to efficiently utilize the limited TFET register resources in achieving the maximal energy savings. Our experiment results show that MEM_RA+TUBA+TRWS obtains 40% energy (including both dynamic and leakage) reduction in register file compared to the baseline case with naive energy optimization technique. Especially, it achieves 54% RF energy savings for memory-intensive benchmarks with only 1.5% performance loss.

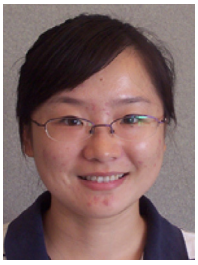
Acknowledgments

This work is supported in part by NSF grants CCF-1537062, CCF-1619243, and CCF-1537085 (CAREER). This work is also co-funded by NSF of China grants 91418203, 61672230. The authors gratefully acknowledge use of the services and facilities of the Center for Advanced Computing & Data Systems (CACDS) at the University of Houston.

References

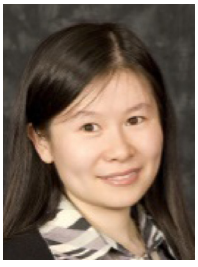
[1] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, D. Glasco, GPUs and the future of parallel computing, IEEE Micro 31 (September) (2011) 7–17.
[2] P. Wang, C. Yang, Y. Chen, Y. Cheng, Power gating strageties on GPUs, ACM Trans. Archit. Code Optim. (TACO) 8 (3 (October)) (2011).
[3] W. Yu, R. Huang, S. Xu, S.-E. Wang, E. Kan, G.E. Suh, SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading, in: Proceedings of ISCA, 2011.
[4] M. Gebhart, D.R. Johnson, D. Tarjan, S.W. Keckler, W.J. Dally, E. Lindholm, K. Skadron, Energy-efficient mechanisms for managing thread context in throughput processors, in: Proceedings of ISCA, 2011.
[5] M. Gebhart, S.W. Keckler, W.J. Dally, A compile-time managed multi-level register file hierarchy, in: Proceedings of MICRO, 2011.
[6] S. Chu, C. Hsiao, C. Hsieh, An energy-efficient unified register file for mobile GPUs, in: Proceedings of IFIP 9th International Conference on Embedded and Ubiquitous Computing, October, 2011.
[7] D. Kanter, Inside fermi: nvidia's HPC push, 2009. http://www.realworldtech.com/page.cfm?ArticleID=RWT093009110932
[8] S. Hong, H. Kim, An integrated GPU power and performance model, in: Proceedings of ISCA, 2010.
[9] S. Mookerjea, D. Mohata, R. Krishnan, J. Singh, A. Vallett, A. Ali, T. Mayer, V. Narayanan, D. Schlom, A. Liu, S. Datta, Experimental demonstration of 100nm channel length in0.53ga0.47as-based vertical inter-band tunnel field effect transistors (TFETs) for ultra low-power logic and sram applications, in: Proceedings IEEE International Electron Devices Meeting (IEDM), 2009, pp. 1–3.
[10] N.N. Mojumder, K. Roy, Band-to-band tunneling ballistic nanowire FET: circuit–compatible device modeling and design of ultra-low-power digital circuits and memories, IEEE Trans. Electron. Devices 56 (2009) 2193–2201.
[11] NVIDIA, CUDA programming guide version 3.0., nvidia corporation, 2010.
[12] Y. Taur, T.H. Ning, Fundamentals of Modern VLSI Devices, Cambridge University Press, 2009.
[13] K. Swaminathan, E. Kultursay, V. Saripalli, V. Narayanan, M. Kandemir, S. Datta, Improving energy efficienty of multi-threaded applications using heterogeneous CMOS-TFET multicores, in: Proceedings of ISLPED, 2011.
[14] J. Singh, K. Ramakrishnan, S. Mookerjea, S. Datta, N. Vijaykrishman, D. Pradhan, A novel si tunnel FET based SRAM design for ultra low-power 0.3v vdd applications, in: Proceedings of ASPDAC, 2010.
[15] V. Saripalli, S. Datta, V. Narayanan, J.P. Kulkarni, Variation-tolerant ultra low-power heterojunction tunnel FET SRAM design, in: Proceedings of International Symposium on Nanoscal Architectures, 2011.
[16] D. Kim, Y. Lee, J. Cai, I. Lauer, L. Chang, S.J. Koester, D. Sylvester, D. Blaauw, Low power circuit design based on heterojunction tunneling transistors (HETTs), in: Proceedings of ISLPED, 2009.
[17] X. Yang, K. Mohanram, Robust 6t si tunneling transistor SRAM design, in: Proceedings of DATE, 2011.
[18] H. Cheng, C. Lin, J. Li, C. Yang, Memory latency reduction via thread throttling, in: Proceedings of MICRO, 2010.
[19] V. Narayanan, V. Saripalli, K. Swaminathan, R. Mukundrajan, G. Sun, Y. Xie, S. Datta, Enabling architectual innovations using non-volatile memory, in: Proceedings of GLSVLSI, 2011.
[20] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, T.M. Aamodt, Analyzing CUDA workloads using a detailed GPU simulator, in: Proceedings of ISPASS, 2009.
[21] Muralimanohar, Naveen, R. Balasubramonian, N.P. Jouppi, CACTI 6.0: A Tool to Model Large Caches, HP Laboratories, 2009, pp. 22–31.
[22] http://www.nvidia.com/object/cuda_sdks.html
[23] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, K. Skadron, Rodinia: a benchmark suite for heterogeneous computing, in: Proceedings of IISWC, 2009.
[24] P.B. suite, http://impact.crhc.illinois.edu/parboil.php.
[25] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, T. Mudge, Drowsy caches: Simple techniques for reducing leakage power, in: Proceedings of ISCA, 2002.
[26] V. Saripalli, A. Mishra, S. Datta, V. Narayanan, An energy-efficient heterogeneous CMP based on hybrid TFET-CMOS cores, in: Proceedings of DAC, 2011.
[27] V. Saripalli, G. Sun, A. Misha, Y. Xie, S. Datta, V. Narayanan, Exploiting heterogeneity for energy efficiency in chip multiprocessors, IEEE Trans. Emerg. Sel. Topics Circuits Syst. 1 (June) (2011) 109–119.
[28] A. Pal, A.B. Sachid, H. Gossner, V.R. Rao, Insights into the design and optimization of tunnel-FET devices and circuits, IEEE Trans. Electron. Devices 58 (4 (April)) (2011).
[29] V. Sathish, M.J. Schulte, N.S. Kim, Lossless and lossy memory i/o link compression for improving performance of GPGPU workloads in Proceedings of PACT, 2012.
[30] V. Narasiman, M. Shebanow, C.J. Lee, R. Miftakhutdinov, O. Mutlu, Y.N. Patt, Improving GPU performance via large warps and two-level warp scheduling, in: Proceedings of MICRO, 2011.
[31] O. Kayiran, A. Jog, M.T. Kandemir, C.R. Das, Neither more nor less: optimizing thread-level parallelism for GPGPUs, in: Proceedings of PACT, 2013.
[32] M. Abdel-Majeed, M. Annavaram, Warped register file: a power efficient register file for GPGPUs, in: Proceedings of HPCA, 2013.
[33] M. Abdel-Majeed, D. Wong, Murali annavaram. warped gates:gating aware scheduling and power gating for GPGPUs, in: Proceedings of MICRO, 2013.
[34] Q. Xu, M. Annavaram, PATS: pattern aware scheduling and power gating for GPGPUs, in: Proceedings of PACT, 2014.
[35] J. Lim, B. Nagesh, et al., Power modeling for GPU architecturesusing mcPAT, ACM Trans. Design Autom. Electron. Syst. 19 (3 (June)) (2014). Art. no. 26
[36] J. Leng, S. Gilani, T. Hetherington, A. ElTantawy, N.S. Kim, T.M. Aamodt, V.J. Reddi, Gpuwattch: enabling energy optimizations in GPGPUS, ISCA 2013: International Symposium on Computer Architecture, 2013.
[37] M. Giles, Jacobi iteration for a laplace discretisation on a 3d structured grid, April, 2008. http://people.maths.ox.ac.uk/gilesm/hpc/NVIDIA/laplace3d.pdf.
[38] U.E. Avci, D.H. Morris, I.A. Young, Tunnel field-effect transistors: prospects and challenges, IEEE J. Electron Devices Soc. 3 (3)(2015) 88–95.
[39] L. Sangpil, et al., Warped-compression: enabling power efficient gpus through register compression, ACM SIGARCH Comput. Archit. News, ACM 43 (3) (2015).
[40] J. Naifeng, et al., An energy-efficient and scalable eDRAM-based register file architecture for GPGPU, ACM SIGARCH Comput. Archit. News, ACM 41 (3) (2013).

Chenhao Xie received the B.S degree in electrical engineering from Beijing University of post and telecommunication, Beijing, China in 2012, and the M.S degree in electrical engineering from Washington University in St.louis, Saint Louis, MO, USA in 2013. He is currently working toward his Ph.D degree in the University of Houston, Houston, TX, USA since 2014. He research interest on energy-efficient computing on HPC platform, memory systems for heterogeneous CPU-GPU processor.



Jingweijia Tan received the B.S. degree in computer science from Jilin University, China, and the M.S. degree in computer science from University of Kansas. She is currently working toward the Ph.D. degree in electrical engineering in University of Houston. Her research interests include computer architecture, high performance computing, General-Purpose computing on GPUs (GPGPUs), hardware reliability and variability, energy-efficient processor design, emerging technologies based architecture design.

Mingsong Chen received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. 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 is an assistant professor in the department of electrical engineering and computer science (EECS) at the University of Kansas (KU). She received the B.S. and M.S. degrees in electronic engineering at Shanghai Jiao Tong University, and the Ph.D. degree in electrical and computer engineering at Texas A&M University. 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.

Lu Peng is currently an Associate Professor with the Division of Electrical and Computer Engineering at Louisiana State University. He holds the Gerard L. "Jerry" Rispone Professorship in Electrical and Computer Engineering. He received the Bachelor's and Master's degrees in Computer Science and Engineering from Shanghai JiaoTong University, China. He obtained his Ph.D. degree in Computer Engineering from the University of Florida in April 2005. His research focus on memory hierarchy system, reliability, power efficiency and other issues in processor design. He received an ORAU Ralph E. Powe Junior Faculty Enhancement Award in 2007 and a Best Paper Award from IEEE International Conference on Computer Design in2001. Dr.Peng is a member of the ACM and the IEEE Computer Society.

Xin Fu received the Ph.D. degree in Computer Engineering from the University of Florida, Gainesville, in 2009. She was a NSF Computing Innovation Fellow with the Computer Science Department, the University of Illinois at Urbana-Champaign, Urbana, from 2009 to 2010. From 2010 to 2014, she was an Assistant Professor at the Department of Electrical Engineering and Computer Science, the University of Kansas, Lawrence. Currently, she is 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. Dr. Fu is a recipient of 2014 NSF Faculty Early CAREER Award, 2012 Kansas NSF EPSCoR First Award, and 2009 NSF Computing Innovation Fellow.