A STT-RAM-based Low-Power Hybrid Register File for GPGPUs

Gushu Li[1], Xiaoming Chen[2], Guangyu Sun[3], Henry Hoffmann[4], Yongpan Liu[1], Yu Wang[1], Huazhong Yang[1]
[1]Deparment of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList),
Centre for Brain Inspired Computing Research (CBICR), Tsinghua University, Beijing China
[2]Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[3]Center for Energy-efficient Computing and Applications, School of EECS, Peking University, Beijing, China
[4]Deparment of Computer Science, University of Chicago, Chicago, IL 60637, USA
lgs11@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn

ABSTRACT

Recently, general-purpose graphics processing units (GPGPUs) have been widely used to accelerate computing in various applications. To store the contexts of thousands of concurrent threads on a GPU, a large static random-access memory (SRAM)-based register file is employed. Due to high leakage power of SRAM, the register file consumes 20% to 40% of the total GPU power consumption. Thus, hybrid memory system, which combines SRAM and the emerging non-volatile memory (NVM), has been employed for register file design on GPUs. Although it has shown strong potential to alleviate the power issue of GPUs, existing hybrid memory solutions might not exploit the intrinsic feature of GPU register file. By leveraging the warp schedule on GPU, this paper proposes a hybrid register architecture which consists of a NVM-based register file and mixed SRAM-based write buffers with a warp-aware write back strategy. Simulation results show that our design can eliminate 64% of write accesses to NVM and reduce power of register file by 66% on average, with only 4.2% performance degradation. After we apply the power gating technique, the register power is further reduced to 25% of SRAM counterpart on average.

Categories and Subject Descriptors

EDA3.3 [**Cross-Layer Power Analysis and Low-Power Design**]: [Architectural low-power techniques: partitioning, scheduling, and resource management]

General Terms

Power, Performance

Keywords

General-purpose graphics processing unit (GPGPU); Spin-torque-transfer random-access memory (STT-RAM); Hybrid register file;

1. INTRODUCTION

In recent years, general purpose graphics processing units (GPGPUs) have been widely used to accelerate applications from many areas, such as high-performance computing [1], numeric algorithms [2], EDA [3], multimedia [4], etc. Compared with traditional CPUs, modern GPGPUs support massive thread-level parallelism with hundreds of cores, large register files, and high memory bandwidth. For example, NVIDIA K80 has 5760 CUDA cores, 256KB register files on each streaming multiprocessor (SM), and 480GB/s bandwidth, providing a peak computing capability of 1870/5600 GFlops (double/single precision) [5]. However, such a high computing capability also results in a high power consumption. Modern high-performance GPGPUs usually have a peak power of more than 200W. For example, the peak power of NVIDIA K80 is 375W [5].

It has been noticed that about 20% to 40% of the total GPU power consumption comes from the large register file [6], which is employed to store the contexts of thousands of threads launched by the programs. Current register files are implemented with the traditional static random-access memory (SRAM) technology that has high leakage power. Thus, the SRAM-based register file has high energy consumption even when supporting inactive threads.

Non-volatile memory (NVM) technology, which provides fast random access, high storage, and non-volatility, is a potential replacement for traditional SRAM technology. Several types of emerging NVMs have been extensively investigated recently. They include resistive random-access memory (ReRAM), spin-torque-transfer random-access memory (STT-RAM or MRAM), and phase-change random-access memory (PCRAM). However, due to their long write latency and high write energy, they are usually employed in the hybrid memory system, which takes advantages of both NVM and traditional memory. Hybrid memory system has shown strong potential to alleviate the power issue of GPUs. Several studies have investigated hybrid memory architecture for global memory on GPU [7, 8, 9, 10]. Satyamoorthy has studied STT-RAM-based and MRAM-based shared memory for GPUs [11, 12]. Goswami has proposed STT-RAM-based register file with write buffer on GPU [6].

However, previous studies do not exploit the intrinsic feature of a GPU register file. In fact, threads in NVIDIA GPUs are scheduled in a smallest unit called warp (the corresponding terminology for AMD GPUs is wavefront). Each warp contains 32 threads executed in parallel in a single-instruction-multiple-data (SIMD) mode. Because of the long latency of main memory accesses, not all the warps are active all the time. When a warp invokes a main memory access, this warp switches to pending (waiting for the main memory data) and another warp is selected to execute. The large register file enables such context switch with trivial overhead. For an SRAM based register file, when a warp is inactive, its registers have to be powered on to store the contexts.

Based on this observation, instead of employing only one block of write buffer for one STT-RAM bank [6], which could not utilize the spatial locality on register file access, we propose a hybrid register file with two SRAM-based write buffers for one STT-RAM block. A new write back strategy is proposed to control when data write back happens following the warp schedule, an intrinsic feature of GPU. The two SRAM-based write buffers take turns to write data back as the warps take turns to execute on GPU, and hide the long write time of STT-RAM in active period of next warp.

DAC '15 , June 07-11 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06$15.00.
http://dx.doi.org/10.1145/2744769.2744785.

Two main contributions of this paper are summarized as follows,

- **Hybrid register file:** We propose a hybrid register file, which is a STT-RAM-based main register file with two SRAM-based mixed write buffers. Different from prior works which employ single buffer for each bank or block, we use two SRAM write buffers for one STT-RAM block. Such an architecture can help to mitigate the problem of STT-RAM's long write latency and high write energy under the control of our write back strategy.
- **Warp-aware write back strategy:** Traditional cache-like write back strategies are not suitable for our write buffer because GPU register file has its unique access patterns. We propose warp-aware write back strategy to leverage the intrinsic features of GPUs and hide the long write latency of STT-RAM during the warp context switch.

Simulation results show that our design can eliminate 64% of write accesses to NVM and reduce power of register file by 66% on average, with only 4.2% performance degradation. In addition, we observe that the utilization of the write buffer varies a lot for different kernels. Thus, we apply power gating on the write buffer and finally reduce 76% of the register power consumption and 19% of the total GPU power. This result is much better than Goswami's work [6], which reduces only about 32% of leakage power and 46% of dynamic power from GPU register file on average.

2. RELATED WORK

Embedded DRAM. The efficiency of using embedded DRAM (eDRAM) as the GPU register files has been investigated recently [13, 14]. Compared with traditional SRAM, eDRAM provides higher density and lower leakage power, but it has limited data retention time. Thus, these two works focus on the refresh strategies of eDRAM. However, eDRAM is intrinsically slower than SRAM. It will inevitably degrade the performance. These two publications have also shown that the performance degradation can be larger than 20% at the 11nm technology node.

Non-Volatile Memory. A resistive memory-based register file is implemented in [6], which uses a pure STT-RAM to replace the original register file with a complex register write back coalescing control strategy. Compared with this design, our design employs two SRAM write buffers for one STT-RAM block instead on only one buffer, and takes into account the warp context switch, which is an intrinsic feature of GPU, to overcome the long write latency of STT-RAM. In addition, we observe the write buffer usage and apply power gating to achieve more power reduction.

Power Gating. Prior research has pointed out that the average inter-access distance to a register is 789 cycles. Thus, aggressively power gating a register file when it is idle can reduce the leakage power significantly with negligible performance degradation [15]. In addition, unallocated register files can be turned off to reduce power too. This approach achieves about 69% of register file power saving with only about 1.02% performance degradation. However, the area overhead for all added circuit is around 4% while our design utilizes the high storage density of STT-RAM. We achieve similar power saving with only 28% of original silicon area.

Register File Cache. Gebhart has proposed register file cache on GPU to achieve more energy-efficiency [16]. This approach employs a small register file cache and a main register file to reduce main register file accesses, the operand delivery energy and the main register file bandwidth requirement. However, this design only saves 36% of the register file access and wire energy, which is much smaller than our 76% power reduction of register file.

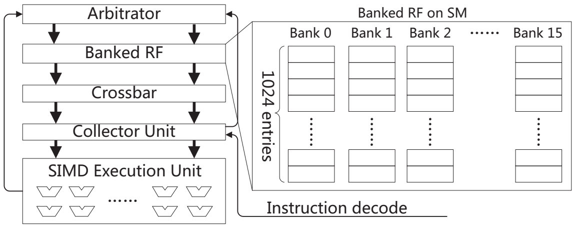


Figure 1: GPU Register File architecture [13]

3. BACKGROUND

3.1 GPGPU and Register File

A modern GPGPU consists of several SMs, and each SM includes execution units, warp schedulers, instruction/data caches, register file, and shared memory. Threads executing on one SM are grouped into warps. A warp which contains 32 threads executing the same instruction on different data, is the smallest schedule unit of NVIDIA GPGPUs. To improve the resource utilization, dozens of warps are invoked and alternated on each SM to hide the long latency of global memory accesses. When a warp is stalled due to a long latency operation, another warp is switched to execute.

Register File: To support zero-cost warp switch, a large SRAM-based register file is employed to maintain the contexts of all the concurrent threads. For example, NVIDIA Kepler architecture employs an 256KB register file which contains 65536 32-bit registers[17]. This size is far beyond that on a traditional CPU. In the NVIDIA parallel thread execution (PTX) standard [18], one instruction can read up to 4 registers and write 1 register at the same time. To implement a wide register file, NVIDIA employs a 16-banked register architecture and each bank contains 1024 entries, as shown in Figure 1. This register file has an 128-byte width which contains 32 32-bit operands, so all the 32 threads in the same warp could fetch one operand at the same time. Each warp has its own register block and each thread inside a warp has its local registers. Registers belonging to the same warp are distributed to multiple banks. This design allows multiple registers to be read or written in one cycle. We propose a distributed hybrid register architecture (shown in Figure 3) which utilizes the spatial locality of the register allocation to support more bandwidth of the register file.

3.2 STT-RAM

Among all emerging non-volatile memory technologies, STT-RAM is most competitive to replace SRAM because of its low access latency, high storage density, and low power [19, 20, 21]. We compare SRAM-based and STT-RAM-based memories of 128KB using NVSim [22], which is a circuit-level memory simulator. The simulation results are shown in Table 1. It can be observed that STT-RAM provides similar performance and very low leakage power with a little more dynamic energy. The critical problem of STT-RAM is that its write latency is 4 times of that of SRAM at 700MHz clock rate, which is a typical GPU core clock rate. This problem may cause substantial performance degradation. In this paper, we propose a mixed write buffer and a new write back strategy to overcome this problem.

Table 1: Parameters of SRAM-based and STT-RAM-based 128KB memories, at 32nm technology node and 700MHz clock rate

	SRAM	STT-RAM
Cell Factor(F^2)	146	57.5
Area(mm^2)	0.194	0.038
Read latency(cycle)	1	1
Write latency(cycle)	1	4
Read energy (pJ/bit)	0.203	0.239
Write energy (pJ/bit)	0.191	0.300
Leakage power(mW)	248.7	16.2

4. HYBRID REGISTER FILE

4.1 Potential of Reducing the Register Power Consumption

Our assumptions and experiments are based on Fermi architecture [23]. To demonstrate the potential of reducing register power, we present some simulation results about the register behavior based on several GPGPU benchmarks. Our benchmarks are from NVIDIA Computing SDK [24] and Rodinia Benchmark suite [25]. It has to be mentioned that Mohammad and Murali have investigated register behavior in [15]. They analyzed the GPU register usage and the average register inter-access distance. It was found that on average 46% of the register file is never allocated and the average register inter-access distance is 789 clock cycles.

Warp inter-active distance: We define the *warp inter-active distance* as the number of idle cycles between two adjacent active periods of one warp. Different from Mohammad's work, our experiments focus on the warp behavior instead of register accesses. In fact, it is more convenient to track the warp behavior rather than the status of register accesses. Moreover, the warp switch mechanism is an intrinsic feature of modern GPUs. Thus, our analysis and method are dedicated to GPUs. The results in Figure 2 show that the average warp inter-active distance for different workloads is about 11000 cycles. It means that the average length of an active period for one warp is about 234 cycles. For NVIDIA Kepler architecture [17], although there are 192 SPs on one SM and one SM can handle 6 warps simultaneously, this conclusion still holds for each individual set of 32 SPs.

These results have revealed that the number of active cycles of a register is significantly fewer than its idle cycles. During the idle cycles, if we store the contexts in NVM instead of SRAM, the leakage power can be significantly reduced. Consequently, there is a huge potential to reduce the register power of GPUs by employing NVM. However, due to the long write latency of NVM (four cycles for STT-RAM as shown in Table 1), directly replacing SRAM with NVM will cause substantial performance degradation. To overcome this challenge, we employ an SRAM-based write buffer for STT-RAM based register file.

SRAM has been used as buffers [11] or caches [6] for NVM to improve the performance. However, the case for register files of GPUs is different. The performance gain obtained from these two studies comes from the high density of NVM and the larger size of NVM-based shared memory or caches. Different from these two researches, the register size in our design is not changed. In the following part of this section, we will illustrate that our new STT-RAM-based register file and its write buffer leverages the long warp inter-active distance and the warp switch mechanism to achieve power reduction with low performance degradation.

4.2 Distributed Hybrid Register File

Since there are no inter-thread register accesses, each SM can have its dedicated registers. Thus, we could distribute the register file close to each core and divide the original 128KB register file into 32 pieces of 4KB small blocks, as shown in Figure 3. Consequently, each core only needs to access a small piece of register and each small register block only needs to maintain the contexts of the thread with the same thread ID from each warp. To deal with the long write latency of STT-RAM, we propose a hybrid register file (HRF) architecture instead of a pure STT-RAM based register file. This hybrid design, with one piece of STT-RAM as the main register file and two small blocks of SRAM as write buffers (WB), could reduce the power consumption without significant performance degradation under our proposed warp-aware write back strategy.

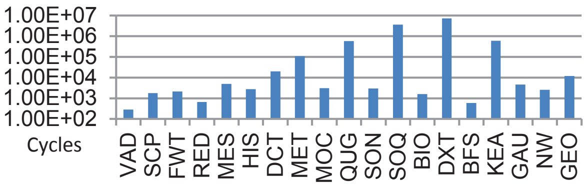


Figure 2: Warp inter-active distance

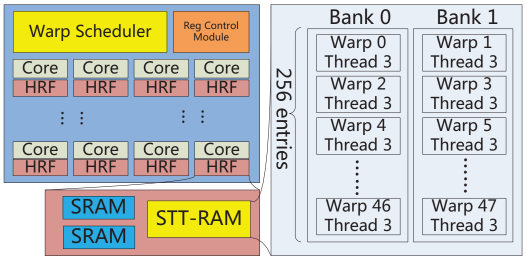


Figure 3: Distributed Hybrid Register File for GPGPU

4.3 Warp-aware Write Back Strategy

This strategy leverages the intrinsic feature of GPU warp scheduling, which is explained as follows. When warp N is active, we use one block of SRAM to maintain the write back registers. The next active warp, warp $N + 1$, should save its write back registers to the other block of SRAM. During the active period of warp $N + 1$, the SRAM block that keeps the write back registers of warp N writes back all the data to the STT-RAM block. After that, when warp $N + 1$ switches to pending and warp $N + 2$ becomes active, it should store the write back data from warp $N + 2$. To support the data read and write at the same time, the STT-RAM block is designed with two banks. We re-map the registers of the warps with an even warp ID to bank 0, and those with an odd warp ID to bank 1, to avoid bank conflicts during read and write operation. When a bank is serving write accesses, the other bank will always handle read access requests from another warp. It is also possible that, due to the data transmission rate limitation, the active period of one warp is not long enough for the WB to write back all the data of the last warp. In this case, the whole pipeline needs to stall. Actually, our strategy hides the long write back time of STT-RAM of one warp in the active period of its next warp.

4.4 Implementation

Since the original banked register file on Fermi architecture is 128KB, 16 banks with 1024 entries in each bank, one HRF should maintain 4KB STT-RAM to keep the same register size in total. One STT-RAM block has two banks. Each of them contains 256 entries.

Our design employs two cache-style data WBs. Each SRAM block has 64 entries with 38 bits in each entry. 32 bits are used to save the data in one register and the other 6 bits are used to record the address (register number) of this register. To control the data flow and guarantee the program accuracy, we add a Reg Control Module (RCM) based on SRAM in each SM, as shown in Figure 5. This RCM shares

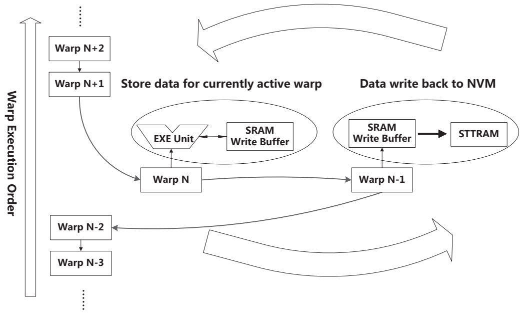


Figure 4: Warp-aware Write Back Strategy

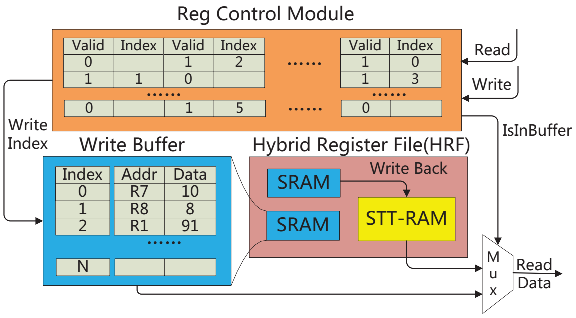


Figure 5: Write Buffer Details

a very similar structure with random access memory(RAM). There are 64 entries and their physical positions represent 64 register addresses. Each entry has one valid bit to show whether the register of the corresponding address is stored in the WB. Another 6 bits are needed to record the index of the entry, which stores the data content in the WB.

When a warp is activated, RCM and one SRAM block are reset. The mechanism of this design is explained as follows:

(1) If a write access comes, RCM will directly access the entry corresponding to the write address, and check the valid bit to decide if the register have already been stored in the WB. If so, RCM will send the index to the WB so that it could overwrite the same entry. Otherwise, the WB will store the data and its address in a new entry, then let RCM record a new index and set the valid bit.

(2) If a read access comes, RCM will check the valid bit of the read address to determine if register in the read address is stored in the WB. At the same time, the STT-RAM block will also receive the read address and send data out. If the register is not in the WB, the core should receive data from the STT-RAM. Otherwise, RCM will send the entry index to WB and the WB would read the entry of that index and send data back to the execution units.

(3) When the active warp becomes waiting for a long global memory access operation, another warp will be activated. One of the SRAM blocks which maintain the register data of this warp begins to write all entries with valid data back to the STT-RAM. The other SRAM block should reset and become the WB of the next active warp.

When the WB is holding data for an active warp, our HRF checks an entry in RCM with the read/write address to decide whether the operand is in the WB. When the WB writes data back to STT-RAM block, as the cache-style WB stores data in the first few successive entries, we do not need to traverse the whole WB. We just read the data and its address entry by entry from the WB and write back to the STT-RAM block. Since the read and write latency of SRAM is fairly small, the access to RCM and the access to WB are directly based on the register address and the index without any search operation. For the read operation, our HRF will access WB and STT-RAM at the same time and determine the output data with a multiplexer. RCM access could be triggered by the rising edge of the GPU core clock signal and WB access could be trigged by the falling edge in the same clock cycle.

5. POWER GATING ON WRITE BUFFER

The overall results of our design are in Section 6 and it is shown that our HRF and warp-aware write back strategy could significantly reduce the register power consumption. However, there are still some opportunities to achieve more power reduction. During the simulation, we record the WB usage information and the components of HRF. In the original HRF design, each WB contains 64 entries, which could save 64 32-bit registers. Although 64 entries in the WB would guarantee that all the registers of one thread could be stored without overflow, the number of entries in use is usually less than 16 in real cases. That means most of the entries would not be used most of the time. If we switch off the latter entries, we could further reduce HRF power since our simulation results shows that the static leakage power of the SRAM blocks in HRF still contributes a lot to the whole power consumption of RF.

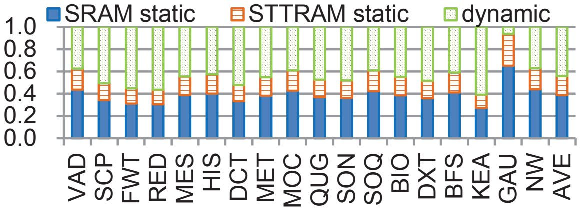
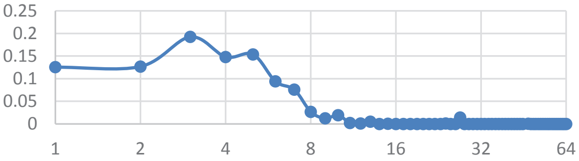


Figure 6: Normalized power components of HRF



X-axis: the number of entries used during one whole active period
Y-axis: the frequency of different write buffer usages

Figure 7: Statistical Write Buffer Usage Distribution

5.1 WB Usage and HRF Power Components

Figure 6 shows power breakdown including dynamic and static RF power consumption parts. The SRAM static power is about 39% of the whole HRF consumption on average. Figure 7 shows the WB usage information for different workloads. It is interesting that in most cases, the number of used entries is smaller than 16. This simulation result shows that about three quarters of the WB is rarely used during the kernel execution. That means about three quarters of the SRAM static power is wasted.

Table 2: Normalized Power Gated WB Parameters

	Non PG	PG
Power Gated area	0	0.75
Dynamic Power	1	1.04
Leakage Power	1	0.3
Silicon Area	1	1.16

5.2 Power Gated WB

Power Gating (PG) technology has been widely used to save SRAM power consumption [26, 27]. In this specific case of SRAM-based WB, we propose the following design according to our previous simulation results. Each WB is divided into two blocks, the first 16 entries, which always have power, and the latter 48 entries power gated. Every time a register not in the WB is written back, the WB will use a new entry and increase the counter representing the number of used lines in WB. In our power gated WB, the power gating control circuit will check the counter at this time. If the counter is over 16, then the latter 48 entries will be switched on. After this WB write all data back to the STT-RAM block, the latter entries will be switched off. Table 2 shows the overhead for the PG circuit [28].

6. EVALUATION

6.1 Simulation Settings and Workloads

Our evaluation focuses on the RF power consumption, overall GPU power, performance (IPC) and the STT-RAM write times. We use a modified version of GPGPU-Sim [29], a detailed GPGPU simulator which shows the warp and RF's behavior accurately in cycle level. We configure the simulator similar to NVIDIA Fermi GTX480 GPU (shown in Table 3). The warp schedule strategy is enforced that two consecutively activated warps are from two different banks, which may cause performance degradation. As there exist many different warp scheduler oriented to different aspects, such as memory access [16, 30, 31], branch divergence [30, 32], cache efficiency [33, 31], etc, further research is necessary for warp scheduling in our hybrid register file. Parameters of the SRAM-based RF, STT-RAM based RF, and our HRF, are simulated by NVSim [22].

The following 18 benchmarks which cover different scientific and computation domains are from NVIDIA Computing SDK [24] and Rodinia Benchmark suite [25]: vectoradd (VAD), scalarProd (SCP), fastWalshTransformation (FWT), reduction (RED), mergeSort(MES), histogram (HIS), dct8x8 (DCT), MersenneTwister (MET), MonteCarlo (MOC), quasirandomGenerator (QUG), sortingNetworks (SON), SobolQRNG (SOQ), binomialOption(BIO), dxtc (DXT), bfs (BFS), kmeans (KEA), gaussian (GAU), nw (NW). AVG represents the geometrical mean.

In our simulation, we observe the RF access inside each SM, and simulate the RF behavior in our customized module and send feedback to the simulator. The original SRAM-based RF in Fermi GPU is the baseline in our comparison.

Table 3: GPGPU-Sim Configuration

GPU Architecture	Fermi
Core (SM) Frequency (MHz)	700MHz
No. of SM	15
Cores per SM	32
RF size	128 KB
Register Width	128 Bytes
Number of Banks	16
Maximux Warps per SM	48
Warp Scheduler	Round Robin
PTXPLUS	Enabled

6.2 Reduction of Performance Degradation

Our experiments test three conditions for performance evaluation, which are SRAM baseline, pure STT-RAM, and our HRF. For the pure STT-RAM, the GPU pipeline would stall for every RF write access. For the HRF design, as the warp-aware write back strategy has explained before, stalls happen when the active period of the next warp is too short to write data from the current warp back to STT-RAM. The simulation result is shown in Figure 9. Our HRF causes only 4.2% performance degradation on average which is much better than pure STT-RAM (33%) and very close to the original SRAM-based RF. Such performance comes from our warp-aware write back strategy which eliminates 64% of the write operations on STT-RAM. (shown in Figure 8)

Potential Compiler Optimization: Warps are scheduled by hardware instead of the compiler. However, if we consider warp inter-active distance during the compiling optimization, it is possible to control the warp inter-active distance in some way. The compiler could rearrange the memory access instructions and the register access instructions to determine the number of instructions between two memory accesses. However, this would not have much side effect on the performance or the power reduction. The shorter the warp inter-active distance, the fewer the instructions between the two memory accesses. That also means fewer register access which will reduce the impact on our performance. Although our power savings come from the long idle period, it is actually the ratio between the idle time and the total which determines the effect of our design.

6.3 Reduction of Power Consumption

For the power consumption evaluation, we compare three conditions, SRAM baseline, HRF, and HRF+PG. Our pure STT-RAM design significantly reduces the leakage power. It is sure that employing WB and RCM will incur more dynamic power. However, the additional dynamic power is much smaller than the static power reduction. The analysis in Section 5 shows that the static leakage power of SRAM-based WB is still the dominant component in the HRF, and PG could reduce the leakage power in the WB further.

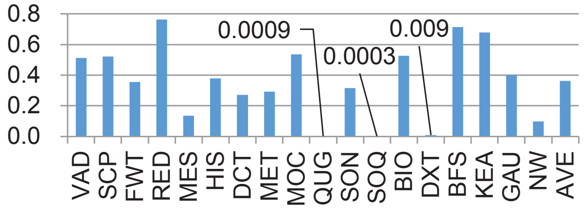


Figure 8: Normalized Write Back Times of STT-RAM

The normalized RF power is shown in Figure 10 and the normalized total GPU power in shown in Figure 11. Our final design HRF+PG, reduced 76% RF power and 19% total GPU power on average.

Goswami's work [6] mentioned in Section 2 is similar to ours. Their proposal reduced 32% of leakage power, 46% of dynamic power and 46% area on average. In comparision, our hybrid register file with warp-aware write back strategy reduces 76% of the register file power and 19% of the total GPU power on average, which is more significant than Goswami's work with only about 28.3% of the area of baseline. Our design takes into account the warp context switch of GPU and employs two buffers so we can achieve more power reduction without significant performance loss, and our method is more suitable for GPUs.

All our assumptions and experiments are based on Fermi architecture [23]. In the later Kepler architecture [17], one SM could handle 6 active warps at the same time. While in Fermi, there is only one active warp. This would have a side effect on our design. First, the dynamic power will contribute more to the RF power while our HRF is to save leakage power. More active warps mean more SRAM blocks are required to maintain the context of more active threads which will increase the leakage power.

6.4 Discussion About Extreme Cases

The total GPU power of the benchmark GAU is only reduced by 3.6%, the smallest result among all results. However, its RF power is reduced by 88%, the most among all selected benchmarks. The reason for this case is that register operation is comparatively infrequent in GAU such that RF power only contributes a little in the total GPU power and the RF power reduction is covered by other factors.

For some extreme cases, like QUG, SOQ and DXT, these benchmarks showed very high STT-RAM write time reduction by keeping writing to a few registers so that most of the RF write access is in the WB. Figure 2 shows that the average warp inter-active distance in these cases are apparently longer than others and verifies that warp context switch is relatively infrequent in these benchmark. The warp active period of these benchmarks is also long enough for our WB to write data back to the STT-RAM block such that the performance is not impacted (in Figure 9)

The benchmark RED and BFS obtain the least two write times reduction in Figure 8 and they both have comparatively low performance (shown in Figure 9). That is because these two benchmarks write back to different registers with frequent warp switch, and they have small warp-inter active distances which could also be verified in Figure 2.

6.5 Reduction of Silicon Area

Additionally, our HRF could also save the silicon area on chip, since STT-RAM provides high storage density. The area of a 128KB STT-RAM-based RF is only 19.6% of that occupied by a SRAM-based RF (shown in Table 1). Considering the HRF and the additional PG circuits, it takes about 28.3% of the area of the baseline. This result, which is estimated in circuit level, makes our HRF a more attractive substitution of the traditional SRAM-based RF for GPU.

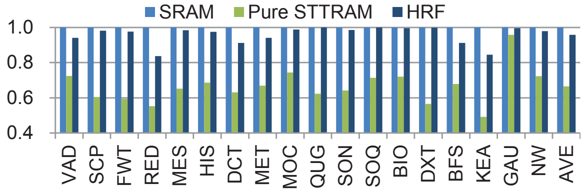


Figure 9: Normalized IPC

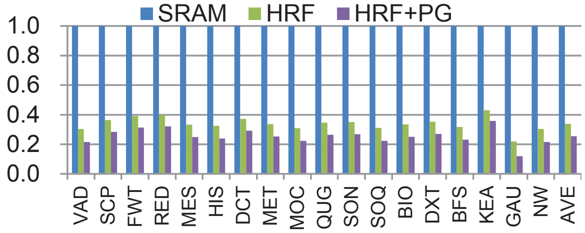


Figure 10: Normalized Power Consumption of RF

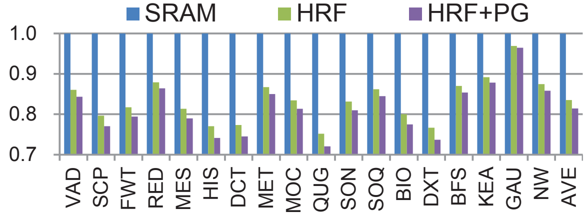


Figure 11: Normalized Power Consumption of Total GPU

7. CONCLUSIONS

The large RF of GPUs are employed to support zero-overhead warp context switch. Traditional SRAM-based RF consumes high leakage power. STT-RAM is a potential substitution of SRAM for its latency, power and high density. Pure STT-RAM-based RF suffers from significant performance degradation for its long write latency. Thus, hybrid memory system consisting of NVM and traditional memory, has been employed on GPU RF. However, existing hybrid memory solutions may not exploit the intrinsic feature of GPU RF. In this paper, we propose a hybrid RF and a warp-aware write back strategy to leverage the warp schedule feature of GPU. Our simulation results show that our hybrid RF could achieve 66% register file power reduction, eliminate 64% write access to NVM with only 4.2% performance degradation on average. Furthermore, by observing the utilization of the WB, we apply PG technique on the W-B and finally reduce 74% of register file power consumption and 19% of total GPU power on average.

8. ACKNOWLEDGMENTS

This work was supported by 973 project 2013CB329000, National Natural Science Foundation of China (61373026), The Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions, Brain Inspired Computing Research, Tsinghua university (20141080934), Tsinghua University Initiative Scientific Research Program, National Natural Science Foundation of China (61202072), National High-tech R&D Program of China (2013AA013201).

9. REFERENCES

[1] Volodymyr V Kindratenko et al. GPU clusters for high-performance computing. In *CLUSTR*, pages 1–8, Aug 2009.

[2] Piyush Sao et al. A Distributed CPU-GPU Sparse Direct Solver. In *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 487–498, 2014.

[3] Xiaoming Chen et al. GPU-Accelerated Sparse LU Factorization for Circuit Simulation with Performance Modeling. *TPDS*, 26(3):786–795, 2015.

[4] Yun Liang et al. Real-time implementation and performance optimization of 3D sound localization on GPUs. In *DATE*, pages 832–835, March 2012.

[5] NVIDIA Corporation. NVIDIA Tesla K80.

[6] Goswami Nilanjan et al. Power-performance co-optimization of throughput core architecture using resistive memory. In *HPCA*, pages 342–353, Feb 2013.

[7] Jishen Zhao and Yuan Xie. Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration. In *ICCAD*, pages 81–87, Nov 2012.

[8] Jishen Zhao et al. Energy-efficient GPU Design with Reconfigurable In-package Graphics Memory. In *ISLPED*, pages 403–408, 2012.

[9] Bin Wang et al. Exploring hybrid memory for GPU energy efficiency through software-hardware co-design. In *PACT*, pages 93–102, Sept 2013.

[10] Dongki Kim et al. Hybrid DRAM/PRAM-based main memory for single-chip CPU/GPU. In *DAC*, pages 888–896, June 2012.

[11] Prateeksha Satyamoorthy and Sonali Parthasarathy. MRAM for Shared Memory in GPGPUs. Technical report, University of Virginia, 2011.

[12] Prateeksha Satyamoorthy. STT-RAM for Shared Memory in GPUs. Master's thesis, University of Virginia, 2011.

[13] Naifeng Jing et al. An Energy-efficient and Scalable eDRAM-based Register File Architecture for GPGPU. In *ISCA*, pages 344–355, 2013.

[14] Naifeng Jing et al. Compiler assisted dynamic register file in GPGPU. In *ISLPED*, pages 3–8, Sept 2013.

[15] Mohammad Abdel-Majeed et al. Warped Register File: A Power Efficient Register File for GPGPUs. In *HPCA*, pages 412–423, 2013.

[16] Mark Gebhart et al. Energy-efficient mechanisms for managing thread context in throughput processors. In *ISCA*, pages 235–246. IEEE, 2011.

[17] NVIDIA Corporation. NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110 (White Paper), 2012.

[18] NVIDIA Corporation. Parallel thread execution.

[19] Cong Xu et al. Device-architecture co-optimization of stt-ram based memory for low power embedded systems. In *ICCAD*, pages 463–470. IEEE Press, 2011.

[20] Clinton W Smullen et al. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *HPCA*, pages 50–61. IEEE, 2011.

[21] Adwait Jog et al. Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps. In *DAC*, pages 243–252. ACM, 2012.

[22] Xiangyu Dong et al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *TCAD*, 31(7):994–1007, 2012.

[23] NVIDIA Corporation. NVIDIA's Fermi: The First Complete GPU Computing Architecture, 2009.

[24] NVIDIA. Computing sdk. *Gpu computing sdk, Avaliable at: https://developer.nvidia.com /gpu-computing-sdk*, 22(07):2013, 2013.

[25] Shuai Che et al. A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads. In *IISWC*, pages 1–11. IEEE, 2010.

[26] Yasuhisa Takeyama et al. A low leakage sram macro with replica cell biasing scheme. *Solid-State Circuits, IEEE Journal of*, 41(4):815–822, 2006.

[27] Pradeep Nair et al. A quasi-power-gated low-leakage stable sram cell. In *MWSCAS*, pages 761–764. IEEE, 2010.

[28] Hailin Jiang et al. Benefits and costs of power-gating technique. In *ICCD*, pages 559–566. IEEE, 2005.

[29] Ali Bakhoda et al. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*, pages 163–174, April 2009.

[30] Veynu Narasiman et al. Improving gpu performance via large warps and two-level warp scheduling. In *MICRO*, pages 308–317. ACM, 2011.

[31] Adwait Jog et al. Owl: cooperative thread array aware scheduling techniques for improving gpgpu performance. *ACM SIGARCH*, 41(1):395–406, 2013.

[32] Wilson WL Fung et al. Dynamic warp formation and scheduling for efficient gpu control flow. In *MICRO*, pages 407–420. IEEE Computer Society, 2007.

[33] Timothy G Rogers et al. Cache-conscious wavefront scheduling. In *MICRO*, pages 72–83. ACM, 2012.