Received June 11, 2020, accepted June 25, 2020, date of publication July 13, 2020, date of current version July 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3008719

Hi-End: Hierarchical, Endurance-Aware STT-MRAM-Based Register File for Energy-Efficient GPUs

WON JEON[1], (Graduate Student Member, IEEE), JUN HYUN PARK[1], YOONSOO KIM[2], GUNJAE KOO[3], (Member, IEEE), AND WON WOO RO[1], (Senior Member, IEEE)

[1]Electrical and Electronic Engineering Department, Yonsei University, Seoul 03722, South Korea
[2]SSD System Engineering Team, NAND Solution Division, SK Hynix, Seongnam 13558, South Korea
[3]Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Won Woo Ro (wro@yonsei.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Korea Government (MSIT) under Grant NRF-2018R1A2A2A05018941, and in part by the Institute of Information and Communication Technology Planning and Evaluation (IITP) funded by the Korea Government (MSIT) under Grant 2019-0-00533, Research on CPU vulnerability detection and validation.

ABSTRACT Modern Graphics Processing Units (GPUs) require large hardware resources for massive parallel thread executions. In particular, modern GPUs have a large register file composed of Static Random Access Memory (SRAM). Due to the high leakage current of SRAM, the register file consumes approximately 20% of the total GPU energy. The energy efficiency of the register file becomes more critical as the throughput of GPUs increases. For more energy-efficient GPUs, the usage of non-volatile memory such as Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) as the GPU register file has been studied extensively. STT-MRAM requires a lower leakage current compared to SRAM and provides an appropriate read performance. However, using STT-MRAM directly in the GPU register file causes problems in performance and endurance because of complicated write procedures and material characteristics. To overcome these challenges, we propose a novel register file architecture and its management system for GPUs, named Hi-End, which exploits the data locality and compressibility of the register file. For STT-MRAM-based GPU register files, Hi-End increases the data write performance and endurance by caching and data compression, respectively. In our evaluation, Hi-End enhances the energy efficiency of a GPU register file by 70.02% and reduces the write operations by up to 95.98% with negligible performance degradation compared to SRAM-based register files.

INDEX TERMS Graphics processing unit, register file, spin-transfer torque magnetic random access memory, data compression, energy efficiency, endurance, chip area.

I. INTRODUCTION

Graphics Processing Units (GPUs) have emerged as the most important computing platform since throughput applications, such as artificial intelligence workloads, became popular. Many emerging applications, such as image classification, audio synthesis, and recommender systems, are accelerated using GPUs. Because modern GPUs are deployed from mobile systems to high-performance data centers, the power consumption of GPUs becomes a critical issue. A GPU core, called a streaming multiprocessor (SM), provides a large register file for massive thread-level parallelism and fast context switching. For instance, the recent NVIDIA Ampere architecture includes a 27,648 KB register file whereas the previous Fermi architecture has a 2,048 KB register file [1], [2]. To provide sufficient on-chip memory performance, the register file in a GPU is implemented with Static Random Access Memory (SRAM) which requires a significant amount of power and area. Previous studies revealed that the GPU register file consumes 15–20% of the total energy consumption on NVIDIA GPU devices [3], [4]. As a modern GPU provides a larger register file, the power consumption and the area overhead by the register file become more critical.

As Complementary Metal–Oxide–Semiconductor (CMOS) scales down, the static power consumption by leakage current occupies a considerable proportion of the

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

entire power budget. To reduce the static power of memory cells, researchers have investigated the Non-Volatile Memory (NVM) technologies such as Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM). The STT-MRAM cells provide similar read performance and a significantly low leakage power compared to the conventional SRAM cells. However, the STT-MRAM cells are affected by inferior write performance and endurance. The long write latency and the limited write endurance of STT-MRAM are critical obstacles preventing it from directly substituting SRAM cells despite its leakage power advantage. Previous studies proposed architectural solutions to overcome these performance hurdles of STT-MRAM [4]–[13]. For instance, SRAM-based *write buffers* have been utilized to minimize the performance degradation from STT-MRAM's long write latency [4], [5]. However, despite their importance, none of the previous studies have considered the low endurance issues of STT-MRAM cells used in place of SRAM cells.

In this paper, we propose a novel STT-MRAM-based register file architecture for GPUs, named Hierarchical, Endurance-aware STT-MRAM-based register file (Hi-End). Hi-End employs a hierarchical structure and a compression technique to solve the problems of STT-MRAM. We designed Hi-End based on two observations. The first observation is that most read/write accesses to the GPU register file are concentrated to a small number of registers. Hence, the GPU register file accesses exhibit a high locality. The second observation is that most write accesses to the GPU register file are compressible. Hence, the number of register file bank activation can be effectively reduced if the data can be compressed.

To fully exploit the locality of the GPU register file, we adopted a *register cache* and a *delay buffer*. The register cache works as a write cache for the slow STT-MRAM register file. The operation of the register cache is similar to that of the conventional cache except it only caches write-back data. When the cache block is evicted from the register cache, the delay buffer operates as a station-like buffer that maintains the evicted block until it is written to the STT-MRAM register file. When evicted data are written to register files, data are compressed to minimize the dynamic power and increase the endurance of STT-MRAM. Since the data are compressed, we can store the same data with a small number of register banks. Hi-End utilizes unused register banks to enhance the endurance of STT-MRAM cells. Parts of the STT-MRAM register banks are accessed sequentially to write data. Consequently, the number of write operations for each STT-MRAM cell is decreased owing to the compression ratio; hence, endurance is improved accordingly.

The contributions of this paper are summarized as follows:

- By exploiting the locality of the GPU register file, we propose a *register cache* and a *delay buffer* architecture to minimize direct access to the STT-MRAM-based register file. Using the register cache and the delay buffer architecture, we can reduce the STT-MRAM write overhead and take advantage of the STT-MRAM register file.

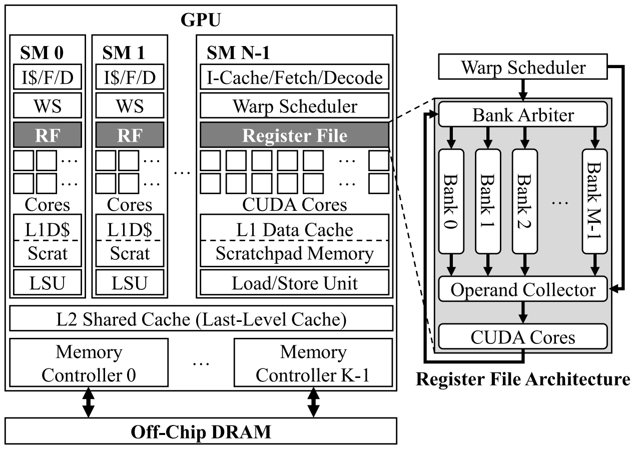


FIGURE 1. Baseline GPU and register file architecture.

- We propose a compression technique to employ the value similarity of GPU register file and decrease the high dynamic power of the STT-MRAM register file. Furthermore, bank-level wear-leveling utilizes concentrated bank access to increase endurance with low-cost architecture.
- With cycle-accurate simulations, we show that Hi-End reduces the energy consumption of GPU register files by 70.02% with a 0.86% of performance drop compared to SRAM-based register file GPUs. Hi-End reduces the number of write operations to the most frequently written register file bank by 95.98%.

The rest of this paper is organized as follows. Section II presents the baseline GPU architecture, register file organization, and characteristics of STT-MRAM. Section III introduces the motivational data and opportunities for our study. In Section IV, we present the detailed architecture and operation of Hi-End. The experimental results of Hi-End, such as the performance and energy efficiency, are described in Section V. In Section VI, we introduce related studies, and we conclude our paper in Section VII.

II. BACKGROUND

A. BASELINE GPU ARCHITECTURE

In this paper, we use NVIDIA GPU terminologies for consistency [1], [2], [14]–[18]. Our baseline GPU model comprises several SMs, and each SM contains dozens of cores; therefore, the GPU device contains hundreds to thousands of computing cores. An SM can execute thousands of threads at a time (e.g. 2,048 in recent GPUs). To provide sufficient memory bandwidth to the massive number of concurrently executing threads, GPUs exploit a large number of register files. Consequently, the register file occupies a large portion of the entire on-chip memory in GPUs.

Fig. 1 describes the baseline GPU and the register file architecture used in this work. The on-chip memory systems of GPUs have hierarchical structures with a register file, L1 data cache, scratchpad memory (or shared memory), and L2

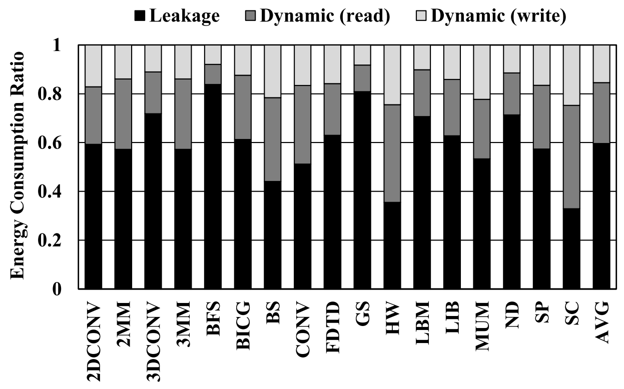


FIGURE 2. Energy consumption of SRAM-based GPU register file.

shared cache (or last-level cache). The register file is used as operands for instructions and additional memory space for context switching. The L1 data cache and the scratchpad memory are located inside of each SM and provide private cache memory space. In particular, the scratchpad memory stores user-defined data for higher throughput. The memory space for the L1 data cache and the scratchpad memory are often exchangeable. Finally, the L2 shared cache is located outside of the SMs and provides shared memory space for all threads operating on a GPU. The L2 shared cache is connected with an off-chip main memory system through memory controllers.

B. GPU REGISTER FILE ARCHITECTURE

Conventional GPUs provide large SRAM-based register files. As shown in Fig. 1, the register file in a single SM is composed of multiple banks in order to provide parallel bank access and higher bandwidth. The baseline GPU model used in this study employs a 128 KB register file composed of 64 banks. Each register file bank contains 256 entries, and each entry is 64-bit wide; hence, a single entry stores two 32-bit data. A GPU hardware executes multiple threads in a group, known as a *warp*, which comprises 32 threads that share the same program counter and execute identical instructions simultaneously. To access multiple registers concurrently, the registers for 32 threads in the same warp are allocated to the same entry location that exists across multiple banks [19]. For instance, assuming the size of a register `r0` is 4 bytes, `r0` of 32 threads in `warp0` can be stored in `entry0` of register banks from 0 to 15. Hence, multiple threads in a single warp access tens of registers simultaneously to minimize clock cycles required to access the register file. However, a large amount of dynamic and static power is dissipated in the register file, as a single warp instruction can activate dozens of register file banks [19].

C. CHARACTERISTICS OF STT-MRAM

High leakage current in SRAM-based memory cells is the major source of large power dissipation in the GPU register file. As shown in Fig. 2, approximately 59.6% of the total GPU register file energy is consumed by the leakage

TABLE 1. Characteristics of SRAM and STT-MRAM [4], [5], [21].

Parameter	SRAM	STT-MRAM
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
Endurance	10^{16}	10^{13}

current in the SRAM cells under 32 nm CMOS technology. Detailed simulation methodologies for the figure are provided in Section V. Using the emerging NVM cells is a prominent approach, as most NVM technologies exhibit extremely low leakage current. For instance, STT-MRAM provides a similar read lartency with a smaller cell size and an extremely low leakage current compared to SRAM-based memory cells. However, NVM technologies still have drawbacks in terms of write performance and endurance. Other NVM technologies such as Phase Change Memory (PCM) and Resistive Random Access Memory (ReRAM) exhibit drawbacks, such as extremely low endurance or long access latency; therefore, they are not suitable for use as fast on-chip memories [20]. In this work, we use STT-MRAM to build an energy-efficient register file for GPUs.

We compare the characteristics of STT-MRAM with the conventional SRAM cells in Table 1. The parameters are measured using NVSim [4], [5], [21]. More detailed experimental methodologies are provided in Section V. The endurance parameter of STT-MRAM is configured based on the actual usage environment [22]. Even though STT-MRAM exhibits the same read latency, the longer write access latency (four times slower) and higher write energy consumption are critical disadvantages of STT-MRAM cells. Another critical drawback of STT-MRAM is its low write endurance, which is 1,000 times lower compared to that of SRAM. In particular, the register file is the memory space that is closest to the computational unit; therefore, data read and write occurs most frequently. Consequently, the low write endurance of STT-MRAM results in the short lifetime of the entire GPU system.

To investigate the effect of STT-MRAM-based register files on GPU performance and lifetime, we configured the STT-MRAM parameters listed in Table 1 into a cycle-accurate GPU simulator [23]. In the study, we replaced the SRAM-based register file in the GPU with an STT-MRAM-based register file without any additional techniques. Our simulation study reveals that the performance decreases by 18% due to slower register file write operations, and the GPU lifetime becomes only 11 months due to the limited register file write endurance. Both the performance drop and short lifetime are impractical for GPUs.

III. MOTIVATION

As introduced in the previous section, low write performance and endurance are critical obstacles hindering the usage of

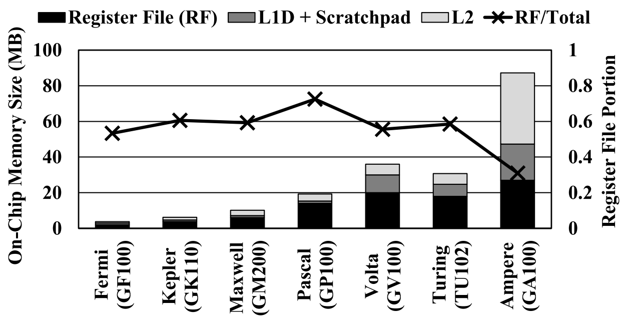


FIGURE 3. On-chip memory size of NVIDIA GPUs by generations.

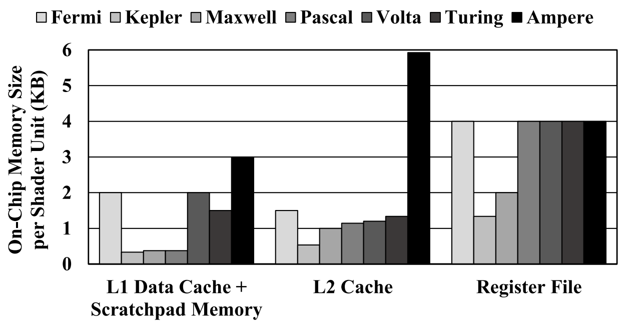


FIGURE 4. On-chip memory size per shader unit.

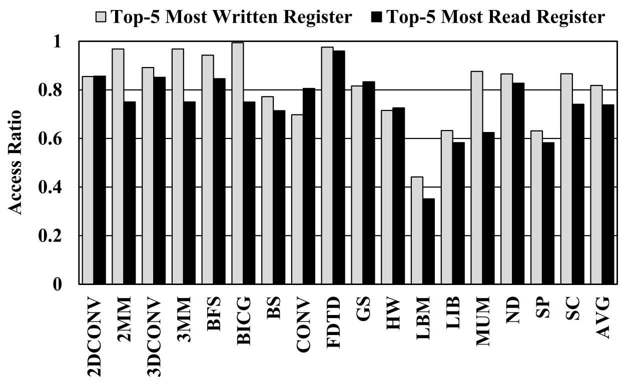


FIGURE 5. Read and write ratios of Top-5 most accessed registers.

STT-MRAM cells as on-chip memory structures, such as register files and L1 caches. In this section, we address the high energy consumption problem of GPU register file, and analyze the unique patterns of register file read and write operations in GPU applications. In addition, we analyze the similarities in the register data accessed by adjacent threads within the same warp. Furthermore, we demonstrate that the register file data can be effectively compressed to a smaller size by exploiting the data similarity observed in the register data. Such observations motivate the development of architectural solutions that can employ STT-MRAM to achieve an energy-efficient register file design.

A. ENERGY CONSUMPTION OF GPU REGISTER FILE

To achieve a high throughput, GPUs execute hundreds of thousands of threads concurrently. For example, the recent NVIDIA Ampere GA100 executes up to 221,184 threads in one GPU device [2]. The large number of register files has been a key enabler for the high throughput of GPUs. We analyzed the on-chip memory size and a portion of the register file for NVIDIA GPUs over the last decade [1], [2], [14]–[18]. As shown in Fig. 3, the register file constituted approximately 60% of the total on-chip memory. Despite SRAM's high energy consumption, the register file occupies a large portion of the on-chip memory system. The relatively low register file portion (approximately 30.1%) of Ampere is due to a substantial increase in the L2 caches. The sizes of L2 caches in NVIDIA GPUs are increased for workloads that have non-cacheable dataset for conventional L2 caches, such as artificial intelligence and high performance computing workloads.

The register file size of NVIDIA GPUs has increased steadily. Fig. 4 shows the on-chip memory size per shader unit (or CUDA core). Except the unique L2 cache in Ampere, the register file size per shader unit is always the largest among various on-chip memory structures. In fact, the register file size per shader unit has remained at 4 KB since Pascal GPUs. The number of computing cores in a GPU device will continue to increase to achieve higher throughput. Hence, the register file size is expected to increase accordingly. However, large SRAM-based register files consume a substantial amount of static energy, as presented in Section II-C. As a result, the energy consumption of register files in GPUs becomes an increasingly important issue.

B. LOCALITY IN GPU REGISTER FILE

We discovered that the register file accesses possess locality based on the register IDs. To reveal this register locality, we measured the read and write access counts by register IDs using a cycle-accurate GPU simulator [23]. Detailed simulation methodology is provided in Section V. For instance, if an instruction requires `r0` and `r1` as operands and writes the execution result to `r2`, we increment the read counts of `r0` and `r1` and the write count of `r2` by 1, respectively. Subsequently, we sorted the register access counts based on the register IDs. This measurement reveals two important characteristics in GPU register file accesses. First, most register file accesses are concentrated on several register IDs. On average, more than 80% of register write accesses are from the Top-5 most written register IDs as shown in Fig. 5. Second, the write access counts from these register IDs are similar to the read counts. This study reveals that high locality and concentration are observed in GPU register file accesses.

C. DATA SIMILARITY IN GPU REGISTER FILE

The other characteristics observed in the GPU register file data is that the data written by the adjacent threads within the same warp have similar values. Hence, this type of data

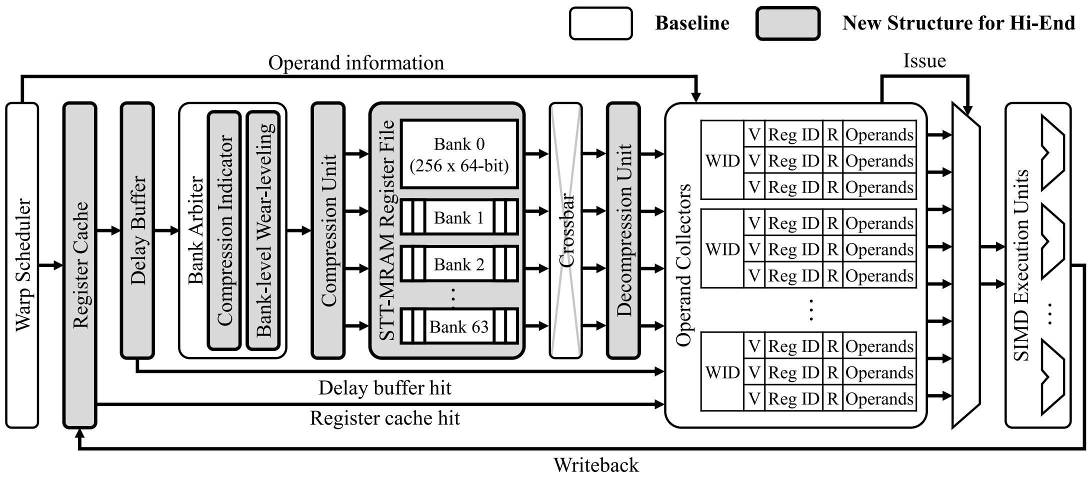


FIGURE 6. Hi-End architecture.

can be easily compressed using simple compression methods, such as the Base-Delta-Immediate (BDI) algorithm [24]. The BDI compression method can perform fast and simple compression and decompression processes. A BDI compressor extracts two parameters, called *base* and *delta*, from multiple sets. The data to be compressed are split into multiple pieces, and the first piece is set as the base. Subsequently, the difference from this base is computed for other pieces. These differences are represented as delta for other values. For instance, 128-byte register data composed of 32 values of 4-byte can be compressed to 35-byte data when BDI compression is applied using 4-byte base and 1-byte delta resolutions. Furthermore, in case all 32 values are the same, the 128-byte register data can be compressed to 4-byte, as the delta is 0-byte. Hence, by applying BDI compression, read/write accesses for multiple memory banks can be reduced.

The original BDI algorithm exploits multiple bases and corresponding deltas to achieve optimal compression ratio. However, in this study, we fixed the base resolution to 4-byte to simplify the hardware design. Furthermore, we restricted the deltas to only have 0-byte, 1-byte, and 2-byte resolutions. By applying these limitations we can minimize clock cycles and power consumption of the compression and decompression units. In this work, if the data cannot be compressed using such fixed base and delta resolutions, the data will remain uncompressed. We calculated the ratio of register write that can be compressed using the BDI algorithm to demonstrate the value similarity of the GPU register file. In fact, more than 62% of the register write can be compressed using the BDI algorithm. Furthermore, recent studies revealed that the register file can be effectively compressed using the BDI compression algorithm [4], [19].

IV. HIERARCHICAL, ENDURANCE-AWARE STT-MRAM REGISTER FILE ARCHITECTURE (HI-END)

We propose an energy-efficient STT-MRAM-based register file for GPUs. Our proposed register file architecture exploits STT-MRAM as a primary memory unit for register data to reduce the static power dissipation. To overcome STT-MRAM's poor write performance, we apply a hierarchical structure that employs a small SRAM as a cache for the STT-MRAM register file. It is effective because of the strong locality observed in the GPU's register file data, as mentioned in Section III-B. In addition, we propose an efficient wear-leveling mechanism using register data compression to extend the lifetime of STT-MRAM cells. The detailed architecture will be described in the following subsections.

A. OVERALL ARCHITECTURE OF HI-END

In this section, we describe the detailed architecture of the proposed STT-MRAM-based hierarchical, endurance-aware register file, called Hi-End. The overall architecture of Hi-End is depicted in Fig. 6. Hi-End includes additional components (*register cache*, *delay buffer*, *compression unit*, and *bank-level wear-leveling*) on the baseline GPU register file. The newly added blocks are shown in gray in the figure. Hi-End employs a hierarchical structure that exploits an SRAM-based write cache to hide the long write latency of the STT-MRAM register file and reduce write counts to the STT-MRAM cells. The SRAM-based write cache works as a gateway for register writes to the STT-MRAM register file. Hence, for register writes, the register values are first written to the register cache. The GPU's register data exhibits strong locality (see Section III-B); therefore, the frequently written register data can remain for a long time in the register cache. Once the cached data are evicted from the register cache, the evicted data are compressed using the compression unit. As the size of the register data is reduced by the compression step, the register data can occupy less banks of the STT-MRAM register file. Furthermore, Hi-End applies bank-level wear-leveling (BWL) to increase the lifetime of the STT-MRAM cells. BWL equally distributes the write requests to multiple banks of the STT-MRAM register file. Hi-End employs the delay buffer between the register cache

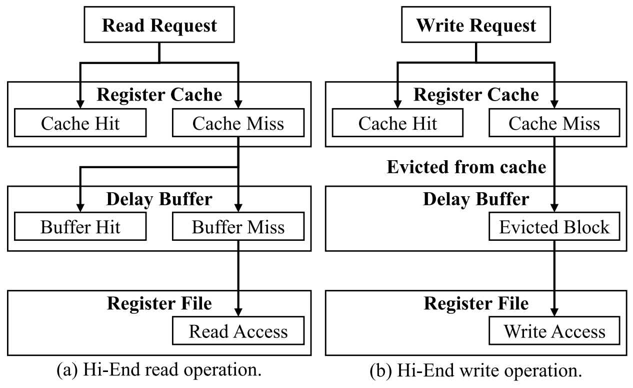


FIGURE 7. Read and write operations in Hi-End.

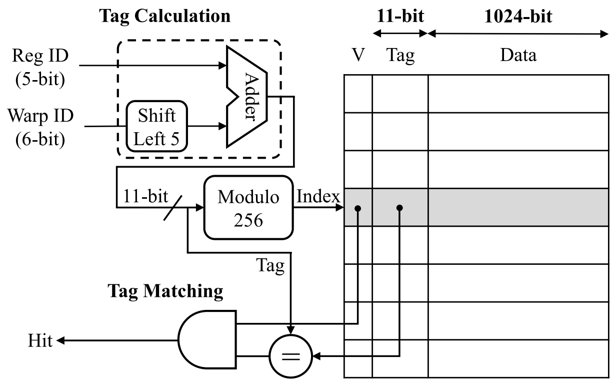


FIGURE 8. Register cache structure.

and the bank arbiter to compensate the additional cycles from the compression unit and slow STT-MRAM write operation. We will describe the detailed structures and mechanisms of Hi-End components in the following sections.

1) HIERARCHICAL REGISTER FILE

Because Hi-End exploits the hierarchical register file, the requested register data can be found in the various levels of the memory units (register cache, delay buffer, and STT-MRAM register file). Fig. 7 summarizes the read and write operations in Hi-End.

The GPU utilizes the operand collector to read register data from the unified register file. When a warp issues an instruction that accesses register data, the warp scheduler passes operand information required for register file accesses (see Fig. 6). Each operand collector contains the operand information, such as warp IDs, valid flags, register IDs, and ready flags. Such operand information is sent to Hi-End, to access register data from the hierarchical register file.

For a read operation, Hi-End first searches the requested data in the register cache. Since the register cache has an SRAM-based high performance cache structure, Hi-End can quickly service the register data to the operand collector if the requested data is found in the register cache. Second, the requested data can be found in the delay buffer if the data is evicted from the register cache and not written completely to the STT-MRAM register file. When the requested data does not exist in both the register cache and the delay buffer, the data is read from the STT-MRAM register file. In this case, if the data stored in the compressed form, the bank arbiter activates and accesses the smaller number of banks. The operand collector obtains the original data form via the decompression unit. Due to the hierarchical structure, the register read latency can degrade the GPU performance when the target data is in the STT-MRAM register file. However, as most read requests are handled in the register cache, the read delay is negligible in Hi-End. A detailed analysis on the effect of read latency on GPU performance is presented in Section V-C.

Once the Single Instruction Multiple Data (SIMD) execution units complete arithmetic operations, a register write request occurs. Hi-End first searches the register cache for the write request, and when a cache hit occurs, the target cache line is updated with the new value. Otherwise, the write-back data is newly allocated in the register cache. The data evicted from the register cache must be stored in the STT-MRAM register file if the register cache is already full with the data that was previously written back; hence, the evicted data enters the delay buffer. Subsequently, the evicted data is compressed by the compression unit and written to the STT-MRAM register file. This mechanism is similar to the conventional cache write-back policy that employs the store buffer except that every evicted data is written to the slower STT-MRAM register file. Once the data is compressed, the bank arbiter selects the banks to be activated for register write operation. Hi-End applies the BWL technique in this step to improve endurance of STT-MRAM cells. The long write latency of STT-MRAM can cause read-after-write hazards and pipeline stalls in SMs. In Hi-End, with the support of the register cache and delay buffer, a register write request instantly obtains a cache line in the register cache; hence, the write delay is negligible.

B. DETAILED ARCHITECTURE OF HI-END COMPONENT

In this subsection, we discuss the detailed architecture and operations of each structure in Hi-End, such as the *register cache*, *delay buffer*, *compression unit*, and *bank-level wear-leveling.*

1) REGISTER CACHE

The register cache is an SRAM-based, fast, and temporary cache memory for write-back data. The register cache employs a direct-mapped cache structure to simplify the hardware complexity. While the write performance of STT-MRAM is much slower than that of SRAM, the read performance of STT-MRAM is similar to that of SRAM. Using this characteristic, the register cache in Hi-End only allocates write data, unlike the general cache memory. A read request can be quickly serviced from the STT-MRAM register file due to its read performance. Therefore, the read request does not require the support of the register cache. Furthermore, requesting the previously written-back register data is more probable than reusing the data previously read

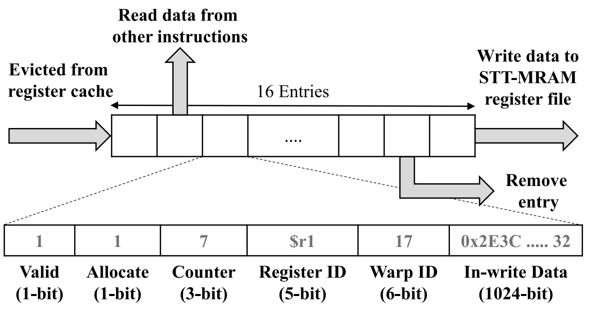


FIGURE 9. Delay buffer structure.

from the register file. Hence, the register cache effectively increases the write hit ratio by allocating only write data. Using this approach, Hi-End can reduce the number of write requests to the STT-MRAM register file, thereby increasing the lifetime of the STT-MRAM cells. In this work, the register cache contains 256 blocks with 1024-bit sized cache lines. As shown in Fig. 8, a single cache line includes a 1-bit valid flag and a 11-bit tag field. Unlike conventional cache tags that use part of an address field, the tags in the register cache utilizes the combination of a 6-bit warp ID and a 5-bit register ID. The tag calculation logic is implemented with a logical shifter and an adder.

2) DELAY BUFFER

The role of the delay buffer is similar to that of the store buffer in the conventional cache architecture. The delay buffer temporarily stores the dataset evicted from the register cache before the dataset are written to the STT-MRAM register file. This buffer space is required for the evicted data since several cycles are taken for data compression and write operation to STT-MRAM cells. Note that the write performance of STT-MRAM used in this work is four times slower than that of SRAM cells, as explained in Section II-C. In this work, we configure the delay buffer to include 16 entries, as shown in Fig. 9. Our simulation study reveals that this depth is sufficient to minimize the structural hazards in the delay buffer considering the cycles required by the compression unit and the STT-MRAM writes. As mentioned previously, the data read requests can be read from the delay buffer. This feature allows Hi-End to read the data while they are being compressed, thereby hiding compression latency. In addition, if the delay buffer does not support the read operation, the operand collector can obtain incorrect data from the STT-MRAM register file due to the redundant write delay. This is because, during the time between data leaves the register cache and is written to the STT-MRAM register, no corresponding register exists in the register cache, and only old data exists in the STT-MRAM register file.

To support the read operation of the delay buffer, each entry of the delay buffer has additional control fields as shown in Fig. 9. These control fields include a 1-bit valid flag, 1-bit allocate flag, 3-bit counter, 5-bit register ID, 6-bit warp ID, and 1024-bit register data. While other fields contain general

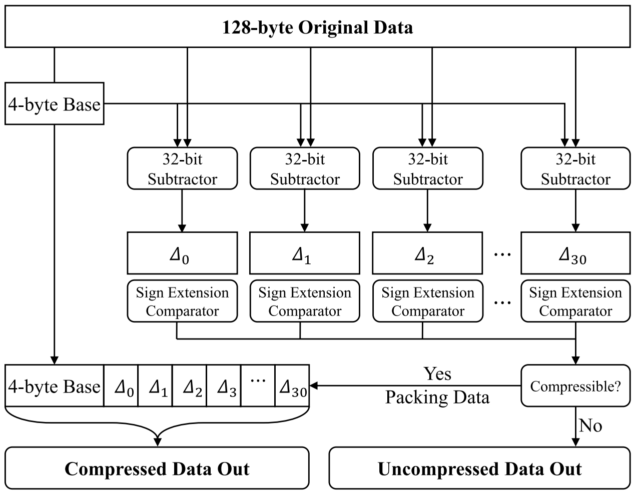


FIGURE 10. Implementation of the compression unit [19].

information, the allocate flag and the counter contain unique information for STT-MRAM. These two fields are used to validate the completions of write operations. In case the polling-based approach is used for the validation, every entry of the delay buffer and STT-MRAM register bank should communicate with each other, and the completion of a write process must be verified in every cycle. Hi-End employs the counter-based approach to avoid such an overhead. The allocate flag indicates whether the corresponding entry is currently writing data into the STT-MRAM register file. The counter field contains the number of cycles remaining to complete the write process of the corresponding entry. The counter is set to 6 when a write process is initiated and decreased by 1 while the allocate flag is 1. In detail, writing data to the STT-MRAM register file requires two cycles for compression and four cycles of STT-MRAM write latency; hence, six cycles are required in total, and the counter must be set accordingly.

3) COMPRESSION UNIT

We implement a compression unit using the BDI algorithm used in *warped-compression* [19]. In the BDI algorithm, the compression ratio can be improved by supporting various configurations for the *base* and *delta*. However, a higher number of options for the base and delta results in a higher hardware overhead; hence, we limit the number of possible sizes of the base and delta. The size of base is fixed to 4-byte, whereas the size of delta can be 0-byte, 1-byte, or 2-byte. Based on our observations shown in Section III, most GPU register file data can be efficiently compressed using these three options for delta. Therefore, the compression unit in Hi-End supports only three compression options to simplify the hardware design. The hardware architecture of the compression unit is described in Fig. 10. Hi-End can reduce the dynamic power for accessing the STT-MRAM register file by compressing the data size. In addition, we can reduce the number of activated banks in the STT-MRAM register file

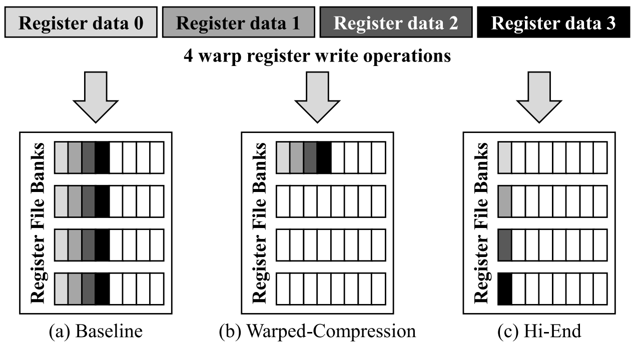


FIGURE 11. Example of bank-level wear-leveling operation in Hi-End compared to the baseline register file and *warped-compression* [19].

when storing compressed data. For instance, the compressed data will activate only 5 banks if the data size is reduced by a 4-byte base and a 1-byte delta, whereas the uncompressed 128-byte data accesses 16 banks in the register file. Hence, Hi-End can effectively reduce the access counts to STT-MRAM cells, and prolong the lifetime of the register file based on BWL.

4) BANK-LEVEL WEAR-LEVELING

To prolong the lifetime of the STT-MRAM cells, Hi-End applies BWL by exploiting the advantages of the compressed data. Hi-End can reduce the overall write counts for the STT-MRAM banks using a hierarchical structure. However, some register banks may have a shorter lifetime if the compressed data is written more frequently to several specific banks. To avoid such cases, Hi-End applies the wear-leveling mechanism, which can balance the write accesses among multiple register file banks.

Fig. 11 presents an illustrative example of BWL of Hi-End compared to the baseline register file without compression and *warped-compression* [19]. Without using the compression technique, each register data is written across multiple banks, as shown in Fig. 11(a). In the baseline model, all banks are affected by the high write count, which results in a short lifetime when the register file is implemented with STT-MRAM. Warped-compression compresses register data, uses less number of banks, and power gates the unused banks to reduce leakage power, as shown in Fig. 11(b). However, in case of STT-MRAM, power gating has a less prominent effect compared to the case of SRAM because the leakage power of STT-MRAM is extremely small. Furthermore, although the lifetime of the unused banks can be prolonged, the banks that frequently store compressed data are still affected by high write count and short lifetimes. The lifetime of a device is determined by the structure with the shortest lifetime. Consequently, the lifetime of the STT-MRAM-based register file with warped-compression is equivalent to that of the baseline register file. As shown in Fig. 11(c), Hi-End compresses register data and stores them into multiple banks to avoid write concentration which occurs in warped-compression. Using the wear-leveling technique, Hi-End can decrease the write concentration and improve the lifetime of the GPU device.

TABLE 2. Simulation Parameter.

Parameters	Value
GPU Microarchitectural Parameter	
Process	32 nm
Core clock frequency	700 MHz
SMs/GPU	15
Warp size	32-threads
Warp scheduling policy	Greedy-Then-Oldest
Register file size	128 KB
Number of register banks/SM	64
Maximum number of warps/SM	48
Maximum number of threads/SM	1,536
Maximum number of registers/SM	32,786
Bit width/bank	64-bit
Number of entries/bank	256
Hi-End Architecture Parameter	
Number of blocks in register cache	256
Register cache size	32.375 KB
Number of entries in delay buffer	16
Delay buffer size	2.03 KB
Compression unit energy/activation (pJ)	23
Compression unit leakage power (mW)	0.12
Decompression unit energy/activation (pJ)	21
Decompression unit leakage power (mW)	0.08

The Compression Indicator Table (CIT) in the bank arbiter contains a compression range that indicates the compression method used in the previous compression operation. The arbiter determines the number of banks to access by referring to the compression range because the number of banks to access changes based on the compression method. Furthermore, the CIT contains the Bank Point Value (BPV), which indicates the next bank ID of previously compressed data accessed. In the write bank allocation process, the bank arbiter sends a BPV to the compression unit with the data. The compression unit identifies the bank ID to start a write operation using the BPV. While the compression unit writes data to the register file, it concurrently updates the CIT with a new BPV and compression range. In the read operation, the decompression unit can identify the number of banks to read by referring to the compression range as well as identify the bank ID to start the access process by subtracting the numbers of banks to access from the BPV.

V. EVALUATION

A. METHODOLOGY

We evaluated our proposed architecture using cycle-accurate simulator, GPGPU-Sim version 3.2.2 [23]. The detailed GPU hardware parameters are shown in Table 2. The parameters of the STT-MRAM and SRAM register files shown in Table 1 are evaluated using circuit-level simulator, NVSim, and configured from previous studies [4], [21]. The parameters of the compression/decompression unit are configured from a previous study [19]. The detailed parameters of Hi-End, such as the register cache, delay buffer, and compression/decompression unit are shown in Table 2.

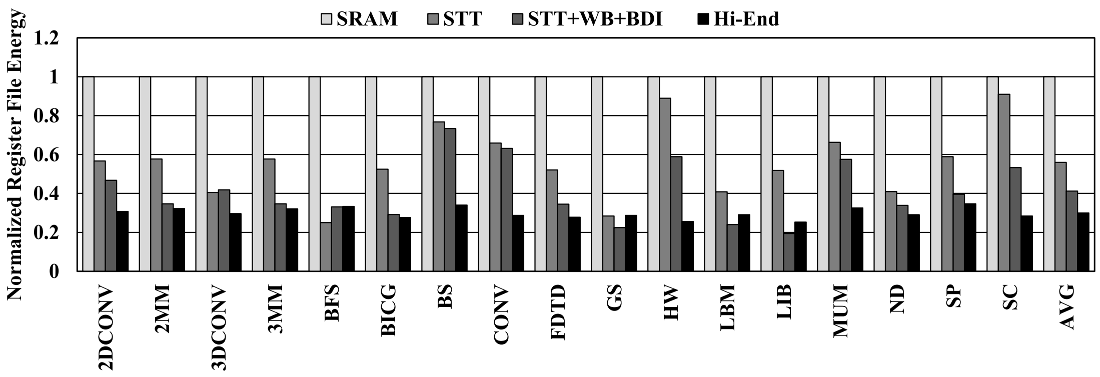


FIGURE 12. Normalized register file energy consumption.

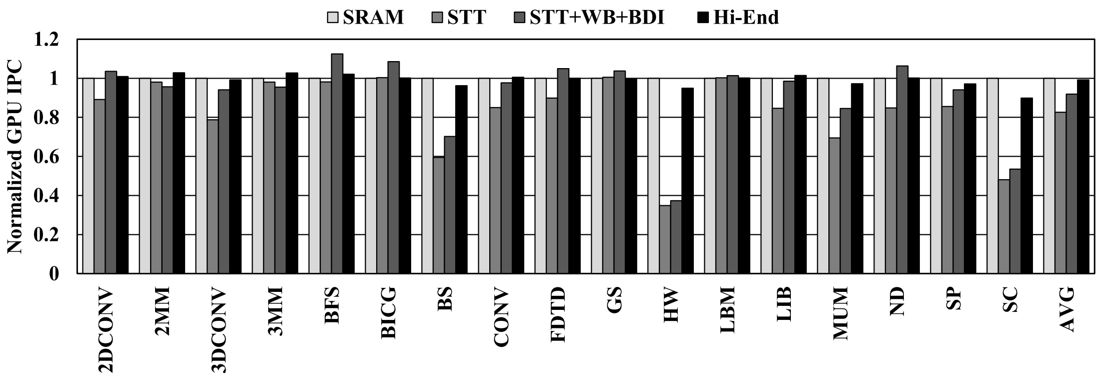


FIGURE 13. Normalized GPU IPC.

We selected 17 benchmarks from Parboil [25], CUDA Software Development Kit [26], Rodinia [27], PolyBench [28], and GPGPU-Sim benchmark suite [23]. Using the benchmarks and architectural configuration above, we compared the following four different architectures:

- SRAM: GPU architecture with SRAM register file.
- STT: GPU architecture with STT-MRAM register file without additional techniques.
- STT+WB(Write Buffer)+BDI: STT-MRAM-based register file which includes a *centralized write buffer* and BDI compression from the previous study [4].
- Hi-End: The proposed architecture which includes a register cache, a delay buffer and a compression/decompression unit with the BWL technique.

B. ENERGY EFFICIENCY

The normalized energy consumption of the register file is shown in Fig. 12. Each data is normalized to the energy consumption of the baseline SRAM-based register file. Using the STT-MRAM directly with the register file instead of using SRAM can reduce the register file energy consumption by 43.98%, but it causes a 17.36% degradation in the GPU Instructions per Cycle (IPC), as shown in Fig. 13. In several benchmarks such as *3DCONV*, *BFS*, *GS*, *LBM*, and *ND*, STT shows relatively low energy consumption compared to other benchmarks. As presented in Fig. 2 in Section II, the SRAM-based register file energy in such benchmarks are largely consumed by the SRAM leakage energy. In particular, the leakage energy in *BFS* and *GS* are over 80% of the total register file energy consumption. High leakage energy consumption can be effectively decreased by directly using the STT-MRAM without additional techniques. Furthermore, as the leakage energy ratio increases, the read and write energy ratio decreases, which results in the lower effect of the additional techniques in Hi-End. Consequently, STT shows the similar or the lowest energy consumption in such benchmarks.

In the previous work (STT+WB+BDI), both the write buffer and the STT-MRAM register file handle read requests simultaneously, thereby requiring additional dynamic energy. Hi-End accesses the STT-MRAM register file only if the read request cannot obtain the data in the upper hierarchy, such as the register cache or the delay buffer; hence, the process is more energy-efficient. Hi-End obtains register files in an energy-efficient manner owing to the following two factors. The first involves the lower leakage current with STT-MRAM. Since STT-MRAM is an NVM and does not require a constant power supply, it affords almost no

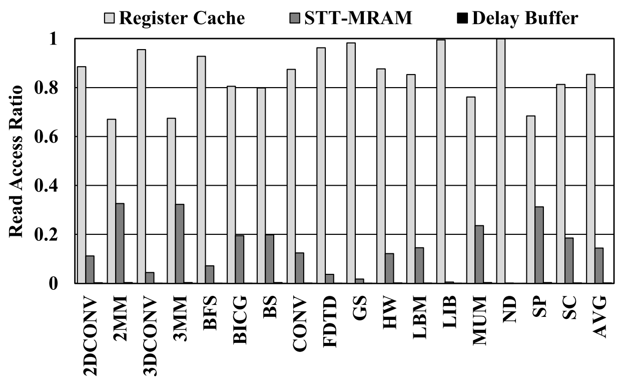


FIGURE 14. Read access ratio of Hi-End memory structures.

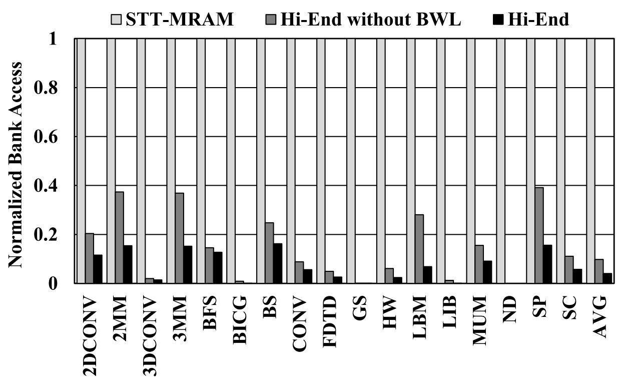


FIGURE 15. Normalized access count of the most accessed register bank.

leakage current compared to SRAM. The second involves the usage of the compression unit. STT-MRAM consumes high dynamic power for read and write operations, as presented in Table 1. Meanwhile, Hi-End reduces the number of bits for read and write operations by compressing data for STT-MRAM, thereby reducing the dynamic power. As a result, Hi-End reduces 70.02% of the register file energy, whereas STT+WB+BDI reduces only reduces 58.79% of the energy compared to the SRAM-based register file.

C. IPC PERFORMANCE

Fig. 13 shows the GPU IPC performance normalized to the baseline SRAM-based register file. Directly using STT-MRAM with the register file instead of using SRAM decreases the performance by 17.36% due to the long write latency of STT-MRAM. In particular, *BS*, *HW*, *MUM*, and *SC* are affected more in STT in terms of both energy and performance. As presented in Fig. 2 in Section II, these applications have a large portion of read and write energy consumption, which means the number of read and write requests are larger than other benchmarks. As the number of read and write increases, STT cannot fully cover the in-flight requests due to the low write performance; hence, results in performance degradation. The previous work that adopts WB covers some of the problems and improves the IPC compared to STT. However, merely using WB cannot fully hide the long write latency of STT-MRAM; therefore, a performance degradation of 8.12% is observed.

Fig. 14 shows the ratio of read access to the different memory structures in Hi-End, such as the register cache, STT-MRAM, and delay buffer. In Hi-End, 85.40% of the read operations are served from the register cache, 14.45% from the STT-MRAM-based register file, and 0.15% from the delay buffer. Despite its low read access ratio, the delay buffer is an essential component, as it prevents incorrect data reads as explained in Section IV-B. As reads from the delay buffer and the STT-MRAM with the decompression unit require two and four cycles, respectively, the average read latency is 1.43 cycles. By employing the register cache, Hi-End has one cycle write latency, as all write accesses are performed in the register cache. As a result, the overall latency increase is only 0.43 read latency. This latency can be hidden by warp scheduling or using other techniques to hide short latency. Consequently, the evaluation results demonstrate that Hi-End can achieve 99.14% in terms of performance compared with the SRAM-based register file.

D. ENDURANCE

Despite the endurance of STT-MRAM being one of the most critical issues in using STT-MRAM, it was considered insignificant in previous studies [4]–[6]. In Hi-End, we improve the endurance of STT-MRAM using two architectural techniques, the register cache and BWL. Fig. 15 shows the access count of the most accessed register bank normalized to the directly used STT-MRAM register file. By reducing the write access to the STT-MRAM register file using the register cache, Hi-End decreases 90.25% of the write access to the STT-MRAM register file compared to directly using the STT-MRAM register file. However, the concentrated bank access provides additional opportunities to improve the endurance of STT-MRAM. Using BWL technique, Hi-End distributes concentrated write accesses of compressed data, thereby deceasing an additional 58.78% of bank accesses to the most written bank compared to Hi-End without BWL. As a result, Hi-End decreases 95.98% of the write accesses in the most written bank compared to directly using the STT-MRAM register file, thereby enhancing the endurance considerably.

E. AREA ANALYSIS

To analyze the area of Hi-End, we used NVSim for the area simulation of the STT-MRAM-based register file, register cache, and delay buffer [4], [21]. The compression/decompression unit are is evaluated based on a previous study [19]. Table 3 shows the relative area of Hi-End by structures compared to the baseline SRAM-based register file. Hi-End reduces 8.19% of the area compared to the baseline SRAM-based register file despite the additional structures in Hi-End. The main reason for area saving of Hi-End is the high density of STT-MRAM compared to that of SRAM; 128 KB of the STT-MRAM register file only occupies 19.59% of the area compared to 128 KB of

TABLE 3. Relative Area of Hi-End Compared to SRAM-based Register File.

Hi-End Component	Area Ratio (%)
Compression/decompression unit	36.08
Register cache	30.55
STT-MRAM register file	19.59
Delay buffer	5.59
Area saving	8.19
Total	100

the SRAM register file. Each register cache, delay buffer, compression/decompression unit, and STT-MRAM-based register file occupies 30.55%, 5.59%, 36.08%, and 19.59% of the SRAM-based register file, respectively, thereby resulting in an 8.19% area reduction.

VI. RELATED WORK

Hi-End aims to provide higher energy efficiency, lower performance degradation, and higher endurance for GPUs with STT-MRAM-based register file. To the best of our knowledge, this work is the first that investigated compositive optimization techniques of register file cache, compression, and wear-leveling for STT-MRAM-based register file. In this section, we discuss previous techniques that are related to our study, such as optimization techniques for GPUs with STT-MRAM register file and GPU register files with compression techniques.

A. REGISTER FILE ADOPTING STT-MRAM

Recent studies attempted to solve the power consumption of the GPU register file by substituting SRAM to STT-MRAM. However, directly using STT-MRAM in a GPU register file instead of SRAM is challenging due to the drawbacks. Several previous works attempted to solve the challenges of STT-MRAM with architectural approaches.

Li *et al.* proposed a hybrid STT-MRAM register file architecture [5]. They adopted *bank-level write buffer* and *warp-aware write back* techniques to overcome the performance drop of STT-MRAM-based register files. They employed two SRAM buffers for every STT-MRAM register file. When a warp is active, one SRAM buffer is used to write back registers, and the next active warp uses the other SRAM buffer. The two buffers provide services alternately between the two warps, thereby hiding the long latency stall of STT-MRAM. However, in this approach, the entire pipeline of the GPU can be stalled when the active period of one warp is not adequate to write back data from the SRAM write buffer to the STT-MRAM register file. Furthermore, such per bank write buffer designs cannot fully utilize SRAM resources when the write requests are unbalanced.

To address the limitations of the previous approach [5], Zhang *et al.* proposed *centralized write buffer* [4]. They employed an SRAM buffer in the bank arbiter, and the buffer is shared by all register banks. Furthermore, they adopted the compression technique to decrease the dynamic power consumption of STT-MRAM. However, in this work, the centralized write buffer and the STT-MRAM register file are concurrently accessed to minimize read latency. This approach decreases the energy efficiency of the GPU register file due to the redundant memory accesses. We compared the effect of the centralized write buffer on the energy consumption and performance with Hi-End in Section V. In addition, unlike Hi-End, the endurance problem of STT-MRAM is not considered in the previous studies.

Liu *et al.* proposed a Multi-Level Cell (MLC) STT-MRAM register file for GPUs [6]. Using the MLC STT-MRAM design, they achieve a larger storage density of register file. Furthermore, they proposed an *MLC-aware register file remapping* strategy and a *warp rescheduling* scheme to optimize the different read and write performance between soft and hard-bit operations in MLC STT-MRAM. This work focused on the optimization techniques for MLC STT-MRAM, whereas our work uses hierarchical approaches and the wear-leveling technique. Consequently, the MLC-aware technique proposed herein is orthogonal to Hi-End; hence, it can be combined.

B. GPU REGISTER FILE WITH COMPRESSION

Previous works utilized several compression techniques to minimize the static or dynamic power consumption of the SRAM-based register file in GPUs.

Lee *et al.* proposed *warped-compression*, an SRAM-based GPU register file with BDI compression algorithm [19], [29]. They restricted base and delta parameters and adopted BDI compression directly to a warp register in an SRAM-based register file GPU. After values are compressed, a smaller number of register banks is used, and the unused register banks are power gated; hence, the technique reduces the power consumption of GPUs. In addition to adopting the compression technique to an STT-MRAM-based GPU register file, we propose additional optimization techniques specific to STT-MRAM. Unlike SRAM that merely benefits from power saving, the compression technique for STT-MRAM additionally benefits from endurance with the support of BWL in Hi-End. In addition, the effect of power saving due to the compression technique of STT-MRAM is more prominent than that of SRAM, since STT-MRAM requires a higher dynamic power consumption.

Wong *et al.* proposed *warp approximation*, which stores a representative value for a warp using an approximation technique [30]. They demonstrated effects similar to those of warped-compression and reduced the number of read or write bits, thereby reducing the dynamic and static power of the SRAM-based register file. Their approach can further increase the compression ratio compared to warped-compression, as values that differ slightly from the representative value can be additionally removed. Since the approximation-based compression technique is orthogonal to Hi-End, it can be applied to our work.

VII. CONCLUSION

With the scaling down of CMOS and adaption of larger register files by the latest GPU devices, the leakage current of conventional SRAM-based register files becomes a critical problem. Recent studies proposed the usage of STT-MRAM as a register file because of its low leakage current and acceptable read performance. However, STT-MRAM-based GPU register files must be carefully designed due to write overheads and endurance problems. These challenges have been addressed in various existing techniques. However, write latency and endurance issues were not completely addressed. Hence, addressing these issues, we propose Hi-End register file architecture for GPUs. In Hi-End, we adopt a register cache to exploit the locality in the register file and a delay buffer to cover evicted data. Moreover, we implement a compression unit to reduce high dynamic power and employ BWL to address concentrated accesses of compressed data. Our experimental results demonstrate that Hi-End reduces energy consumption by 70.02% with only a 0.86% performance degradation compared to an SRAM-based register file. Moreover, Hi-End reduces 95.98% of the write access to the register bank by distributing concentrated bank accesses and reducing write accesses with register caches.

ACKNOWLEDGMENT

(Won Jeon and Jun Hyun Park contributed equally to this work.)

REFERENCES

[1] *NVIDIA's Next Generation CUDA Computer Architecture: Fermi*, NVIDIA, Santa Clara, CA, USA, 2009.

[2] *NVIDIA A100 Tensor Core GPU Architecture: Unprecedented Acceleration at Every Scale*, NVIDIA, Santa Clara, CA, USA, 2020.

[3] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 487–498, 2013.

[4] H. Zhang, X. Chen, N. Xiao, and F. Liu, "Architecting energy-efficient STT-RAM based register file on GPGPUs via delta compression," in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, 2016, p. 119.

[5] G. Li, X. Chen, G. Sun, H. Hoffmann, Y. Liu, Y. Wang, and H. Yang, "A STT-RAM-based low-power hybrid register file for GPGPUs," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, pp. 1–6.

[6] X. Liu, M. Mao, X. Bi, H. Li, and Y. Chen, "An efficient STT-RAM-based register file in GPU architectures," in *Proc. 20th Asia South Pacific Design Autom. Conf.*, Jan. 2015, pp. 490–495.

[7] S. Mittal, "A survey of techniques for architecting and managing GPU register file," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 16–28, Jan. 2017.

[8] H. Zhang, X. Chen, N. Xiao, L. Wang, F. Liu, W. Chen, and Z. Chen, "Shielding STT-RAM based register files on GPUs against read disturbance," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 2, pp. 1–17, Mar. 2017.

[9] Z. Gong, K. Qiu, W. Chen, Y. Ni, Y. Xu, and J. Yang, "Redesigning pipeline when architecting STT-RAM as registers in rad-hard environment," *Sustain. Comput., Informat. Syst.*, vol. 22, pp. 206–218, Jun. 2019.

[10] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "Towards warp-scheduler friendly STT-RAM/SRAM hybrid GPGPU register file design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 736–742.

[11] Y. Ni, Z. Gong, W. Chen, C. Yang, and K. Qiu, "State-transition-aware spilling heuristic for mlc stt-ram-based registers," *VLSI Des.*, vol. 2017, pp. 1–8, Nov. 2017.

[12] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 371–382, Jun. 2010.

[13] J. Zhang, M. Jung, and M. Kandemir, "FUSE: Fusing STT-MRAM into GPUs to alleviate off-chip memory access overheads," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 426–439.

[14] *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*, NVIDIA, Santa Clara, CA, USA, 2012.

[15] *NVIDIA GeForce GTX 980: Featuring Maxwell, The Most Advanced GPU Ever Made*, NVIDIA, Santa Clara, CA, USA, 2014.

[16] *NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built*, NVIDIA, Santa Clara, CA, USA, 2016.

[17] *NVIDIA Tesla V100 GPU Architecture: THE WORLD'S MOST ADVANCED DATA CENTER GPU*, NVIDIA, Santa Clara, CA, USA, 2017.

[18] *NVIDIA Turing GPU Architecture: Graphics Reinvented*, NVIDIA, Santa Clara, CA, USA, 2018.

[19] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, "Warped-compression: Enabling power efficient gpus through register compression," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 502–514, 2015.

[20] S. Mittal, J. S. Vetter, and D. Li, "A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1524–1537, Jun. 2015.

[21] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[22] S. Yazdanshenas, M. R. Pirbasti, M. Fazeli, and A. Patooghy, "Coding last level STT-RAM cache for high endurance and low power," *IEEE Comput. Archit. Lett.*, vol. 13, no. 2, pp. 73–76, Jul. 2014.

[23] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2009, pp. 163–174.

[24] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2012, pp. 377–388.

[25] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center Reliable High-Perform. Comput.*, vol. 127, pp. 1–12, Mar. 2012.

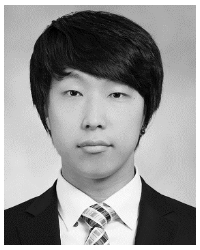
[26] *NVIDIA CUDA SDK Code Sample 4.0*, NVIDIA, Santa Clara, CA, USA, Mar. 2016.

[27] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Dec. 2010, pp. 1–11.

[28] L.-N. Pouchet. (2012). *Polybench: The Polyhedral Benchmark Suite*. [Online]. Available: http://www.cs.ucla.edu/pouchet/software/polybench

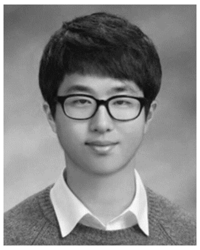
[29] S. Lee, K. Kim, G. Koo, H. Jeon, M. Annavaram, and W. W. Ro, "Improving energy efficiency of GPUs through data compression and compressed execution," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 834–847, May 2017.

[30] D. Wong, N. S. Kim, and M. Annavaram, "Approximating warps with intra-warp operand value similarity," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Mar. 2016, pp. 176–187.



WON JEON (Graduate Student Member, IEEE) received the B.S. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree with the Embedded Systems and Computer Architecture Laboratory, School of Electrical and Electronic Engineering. His current research interests are GPU memory systems, non-volatile memory, processing-in-memory architecture designs, and approximate computing for neural network applications.

JUN HYUN PARK received the B.S. degree in electrical and electronic engineering from Chung-Ang University, Seoul, South Korea, in 2018, and the M.S. degree in electrical and electronic engineering from Yonsei University, Seoul, in 2020. He is currently working as an IP Design Engineer with the System LSI Division, Samsung Electronics. His current research interests are interconnection technology in SoC and GPU architecture.



YOONSOO KIM received the B.S. and M.S. degrees in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2016 and 2018, respectively. He is currently working as a System Engineer with NAND Solution Division, SK Hynix. His current research interests are the functionality of solid-state drive and GPU architecture.

GUNJAE KOO (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Seoul National University, in 2001 and 2003, respectively, and the Ph.D. degree in electrical engineering from the University of Southern California, in 2018. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Korea University. His research interests are in the general area of computer system architecture and spans parallel processor architecture, storage and memory systems, accelerators, and secure processor architecture. Prior to joining Korea University, he was an Assistant Professor at Hongik University. His industry experience includes the position of Senior Research Engineer at LG Electronics and also a Research Intern with Intel. He is a member of the ACM.

WON WOO RO (Senior Member, IEEE) received the B.S. degree in electrical engineering from Yonsei University, Seoul, South Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, in 1999 and 2004, respectively. He worked as a Research Scientist with the Electrical Engineering and Computer Science Department, University of California, Irvine, CA, USA. He currently works as a Professor with the School of Electrical and Electronic Engineering, Yonsei University. Prior to joining Yonsei University, he worked as an Assistant Professor with the Department of Electrical and Computer Engineering, California State University, Northridge, CA, USA. His industry experience includes a college internship with Apple Computer, Inc., and a Contract Software Engineer with ARM, Inc. His current research interests include high-performance microprocessor design, GPU microarchitectures, neural network accelerators, and memory hierarchy design.