TEA-RC: Thread Context-Aware Register Cache for GPUs

IPOOM JEONG1, YUNHO OH2, (Member, IEEE), WON WOO RO1, (Senior Member, IEEE), AND MYUNG KUK YOON3

1School of Electrical and Electronic Engineering, Yonsei University, Seoul 03722, Republic of Korea
2School of Electrical Engineering, Korea University, Seoul 02841, Republic of Korea
3Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea

Corresponding author: Myung Kuk Yoon (myungkuk.yoon@ewha.ac.kr)

This work was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] (No.2021-0-02068, Artificial Intelligence Innovation Hub), in part by Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2022R1C1C1011021), and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (No. NRF-2021R1A2B5B01002932).

ABSTRACT Graphics processing units (GPUs) achieve high throughput by exploiting a high degree of thread-level parallelism (TLP). To support such high TLP, GPUs have a large-sized register file to store the context of all threads, consuming around 20% of total GPU energy. Several previous studies have attempted to minimize the energy consumption of the register file by implementing an emerging non-volatile memory (NVM), leveraging its higher density and lower leakage power over SRAMs. To amortize the cost of long access latency of NVM, prior work adopts a hierarchical register file consisting of an SRAM-based register cache and NVM-based registers where the register cache works as a write buffer. To get the register cache index, they use the partially selected bits of warp ID and register ID. This work observes that such an index calculation causes three types of contentions leading to the underutilization of the register cache: inter-warp, intra-warp, and false contentions. To minimize such contentions, this paper proposes a thread context-aware register cache (TEA-RC) in GPUs. In TEA-RC, the cache index is calculated considering the high correlation between the number of scheduled threads and the register usage of threads. The proposed design shows 28.5% higher performance and 9.1 percentage point lower energy consumption over the conventional register cache that concatenates three bits of warp ID and five bits of register ID to compute the cache index.

INDEX TERMS Graphics processing units, register file, register cache, volatile memory, non-volatile memory, hybrid register file, hierarchical register file.

I. INTRODUCTION

Graphics processing units (GPUs) are used to compute a wide range of general-purpose applications, and such a paradigm is known as general-purpose computing on GPUs (GPGPUs). Especially, machine learning and blockchain applications heavily rely on the high computing power of GPUs, which is derived from a massive number of computing cores [1]–[5]. To maximize their utilization, GPUs are capable of scheduling thousands of threads concurrently and have a large-sized register file to maintain the contexts of concurrently scheduled threads. The latest generation of NVIDIA GPU architecture, Ampere (A100), can schedule up to 221,184 (2,048 × 108) threads concurrently and has 27,648 KB (256 KB × 108) register file to hold the contexts of such threads [6]. Compared to the previous generation, Volta (V100), the Ampere architecture has 25.9% (20,480 KB vs. 27,647 KB) larger register file and can schedule up to 25.9% (163,840 vs. 221,184) more threads [6], [7]. A similar trend has been observed across several previous GPU architecture generations [8].

However, providing such high throughput leads to substantial energy consumption. Among the various components in GPUs, the register file is one of the most power-hungry...
components, consuming around 15–20% of total GPU energy [9]–[13]. Over the past decade, researchers have explored several architectural techniques to minimize the energy consumption of the GPU register file [9]–[11], [14]–[18]. One attractive solution to minimize the energy consumption of the register file is to adopt emerging non-volatile memories (NVMs) – such as spin-transfer torque magnetoresistive random access memory (STT-MRAM) and spin-orbit torque MRAM (SOT-MRAM) [11], [14], [16] – as a substitute for static random-access memory (SRAM) of an existing register file. Leveraging their low leakage power, implementing the register file using NVMs significantly reduces the leakage energy consumption.

Even with its lower leakage power, NVM cannot be solely used as the register file due to the longer access latency than SRAM, as shown in Table 1. To overcome the latency problem while taking advantage of low leakage power, researchers have proposed the hybrid or hierarchical register file composed of SRAM and NVM [11], [14], [19]. In the hybrid register file [14], the SRAM write buffers are inserted ahead of each STT-MRAM-based register bank. Computation results are first written to the SRAM write buffers, and then written back to the STT-MRAM-based registers during the execution of other instructions. Due to the high TLP of GPUs, the access latency of the STT-MRAM has been effectively hidden; however, all the write operations must be performed, which may yield an endurance problem for STT-MRAM. Jeon et al. proposed the hierarchical register file by placing the SRAM-based register cache in between functional units and the STT-MRAM-based registers [11]. Similar to the hybrid register file, the hierarchical register file performs the write-back operations. The key difference is that it performs the write-back operations only if there are no empty cache lines to support incoming write requests. The hierarchical register file is a more attractive solution since it could mitigate the endurance problem of STT-MRAM by minimizing the STT-MRAM write operations [11].

Table 1. Characteristics of SRAM and STT-MRAM [14].

| Parameter          | SRAM | STT-MRAM |
|--------------------|------|----------|
| Area (mm²)         | 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     |

The conventional hierarchical register file employs a direct-mapped register cache that is indexed by concatenating the partially selected bits of register ID and warp ID. According to our analysis, such an index calculation scheme causes three types of contentions that reduce the effective capacity of the register cache: *inter-warp*, *intra-warp*, and false contentions. These contentions arise since the number of scheduled warps and the number of registers used by each warp are different for each application. For example, a single cache line would be accessed by different register IDs within a warp, or even by different warps because both register ID and warp ID are used for indexing (see Section III).

If a large number of warps are scheduled in streaming multiprocessors (SMs), each of which requires a small number of registers, different warps may access the same cache line frequently. We name it as *inter-warp* contention. On the other hand, if a warp uses a large number of registers to compute instructions, different registers within a single warp would access the same cache line frequently, which is named as *intra-warp* contention. Lastly, statically selecting the partial bits of register ID and warp ID for indexing may cause false contention. As the register usage of a warp and the number of concurrently scheduled warps are different for each application, predefined bit ranges of register ID and warp ID would present a larger number than the number of registers that are actually used by warps and the number of scheduled warps, respectively. In this case, some of the selected bits may change but remain zeros; thus, some of the cache entries are not accessed during the execution of scheduled kernels. Therefore, only a limited number of cache entries are heavily accessed, while the others are not accessed at all. Due to the aforementioned three contentions, a large number of redundant write-back operations are observed in the hierarchical register file, resulting in overall performance degradation.

Based on this observation, this paper proposes the thread context-aware register cache, called TEA-RC. In GPUs, there is a strong correlation between the number of registers used by each warp and the number of warps scheduled on SMs. The register file size is one major factor that determines the number of issuable warps in GPUs [20]. As the number of registers used by a warp increases, the number of warps scheduled on SMs decreases and vice versa. Thus, based on the register usages and scheduled warps, the number of partially selected bits of register ID and warp ID should be changed dynamically for optimal register cache management. However, it could be difficult to decide which bit(s) and how many bit(s) should be selected from the register ID and the warp ID for every application.

Instead of selecting partial bits of register ID and warp ID, our proposed architecture uses all bits to compute the index while automatically capturing the effective ranges of register ID and warp ID. To do this, TEA-RC first reverses the warp ID, so the least significant bit (LSB) is stored on the most significant bit (MSB) side, and the MSB is stored on the LSB side. Then, the index is computed via bitwise XOR operation between the reversed warp ID and the original register ID. Unlike other logical gates, the output of the XOR gate is evenly distributed into zero and one; thus, it leads to higher fairness [21], [22]. The MSB and LSB sides of the index are now largely affected by the reversed warp ID and register ID, respectively. In such a scheme, if the number of scheduled warps is decreased by the number of registers used by warps, then more bits of register IDs affect the calculation of the cache index. In contrast, if the number of registers used by
warps is decreased, more warps will be scheduled on GPUs. Then the output of the computed index is largely affected by the reversed warp ID.

In our evaluation, TEA-RC shows 103.3% higher performance than the STT-MRAM-based register file. Also, TEA-RC shows 28.5% better performance than the conventional register cache, selecting three bits of warp ID and five bits of register ID to compute the cache index. The proposed indexing scheme reduces the number of write-back operations by 28.3 percentage points over the conventional register cache. It shows 70.3 percentage points lower energy consumption than the baseline GPUs with the SRAM-based registers.

The remainder of this paper is organized as follows. In Section II, we briefly introduce GPGPU thread hierarchy and the hierarchical register file on top of the baseline GPU architecture. In Section III, we present the motivational data followed by an in-depth analysis. Section IV introduces detailed hardware modifications to support TEA-RC. In Section V, we show the evaluation results of proposed architecture. Section VI represents the related work. Finally, we conclude this paper in Section VII.

II. BACKGROUND
For consistency, this paper uses the NVIDIA terminology to explain software and hardware components [8], [23]. This section first introduces the thread hierarchy of GPGPU applications and the baseline GPU architecture with the hierarchical register file.

A. GPGPU THREAD HIERARCHY
FIGURE 1 shows the thread hierarchy of GPGPU applications. The applications typically consist of several kernels, and each is a fundamental computing block scheduled on GPUs. Then, each kernel again consists of thousands of thread blocks (TBs), and these TBs are scheduled on SMs in GPUs based on the TB scheduling policy. The number of TBs is determined by the programmers based on the application characteristics and the hardware limitations [24]. Furthermore, TBs usually consist of hundreds of threads and may have up to 1,024 threads [23]. Similar to the number of TBs per kernel, the number of threads in TBs is determined by the programmers based on the application characteristics [24]. Consequently, the total number of threads in each kernel can be calculated by multiplying the number of TBs by the number of threads per TB.

B. BASELINE GPU ARCHITECTURE
FIGURE 2 shows the GPU architecture with the hierarchical register file. GPUs have dozens of SMs, and the TB scheduler assigns TBs to SMs based on the scheduling policy. The number of assigned TBs is determined by thread count limit, shared memory size, and register file size [20], [25], [26]. As shown in FIGURE 1, TBs usually consist of hundreds of threads. After scheduling TBs in SMs, threads are grouped into warps consisting of 32 threads. Let us assume that TBs have 128 threads, then a total of four warps (128 \div 32) are created from each TB. Threads within each warp must execute the same instruction in lockstep, and such a computing paradigm is known as the single instruction multiple threads (SIMT) execution model. Warp schedulers in SMs select an executable warp, then the threads in the selected warp execute the same instruction with different data. The computed data are stored in the register file. In GPUs, registers are thread-specific; thus, each thread can access only its own registers. Because each SM can execute thousands of threads concurrently, the size of the register file must be large enough to keep the contexts of concurrently scheduled threads.

GPUs have a similar memory hierarchy to CPUs. In GPUs, SMs have a private L1 cache, and there is one L2 cache shared by all SMs. Unlike CPUs, GPUs have scratchpad memory (also known as shared memory), texture memory, and constant memory. The scratchpad memory is programmable memory that is visible to programmers. The texture memory is used to store texture data of 3D models for the rendering, and the constant memory is read-only memory visible to all the threads [27]. In addition, there is a global memory accessible to all scheduled threads in GPUs.

C. HIERARCHICAL REGISTER FILE
The hierarchical register file consists of an SRAM-based register cache and STT-MRAM-based registers. FIGURE 3 shows the conventional cache indexing scheme and the write-back operations in the hierarchical register file. As shown in TABLE 1, STT-MRAM has lower leakage power consumption than SRAM (248.7 mW vs. 16.2 mW);
I. Jeong et al.: TEA-RC: Thread Context-Aware Register Cache for GPUs

FIGURE 3. Index calculation steps of conventional SRAM-based direct-mapped register cache and write-back operations.

however, the write latency and write energy of STT-MRAM are higher than that of SRAM. Thus, the SRAM-based register cache works as a write buffer storing computed data. In prior work [11], the direct-mapped cache design is used for the register cache, and only partially selected bits of register ID and warp ID are used to compute an index \( \text{①} \). One possible design choice presented by the prior work uses three bits of warp ID and five bits of register ID for calculating the index, leading to 256 entries for the cache. The register cache performs the write-back operations moving data to the STT-MRAM-based registers when the computed data attempts to write the value in the cache line having valid data \( \text{②} \). Such a design can alleviate the long write latency, high write energy, and write endurance problems of STT-MRAM [11], [14].

The detailed analysis of the register cache is presented in the following section.

III. MOTIVATION

The conventional register cache uses a fixed number of bits of warp ID and register ID to compute the index of the register cache. This paper reveals that such a fixed direct-mapped register cache is not the optimal solution for various GPGPU applications. FIGURE 4 shows the instructions per cycle (IPC) results by varying the sampling bits of warp ID and register ID to compute the index of the register cache. IPC is calculated by dividing the total number of executed instructions by the total execution cycles. As a result, as IPC increases, the total execution cycles of applications decrease. In other words, the relative execution cycles are equivalent to the inverse of normalized IPC since the executed instructions of all configurations for each application are always the same. In the figure, there are seven different configurations: Baseline, STT-RF, RC+RF (3/5), RC+RF (4/4), RC+RF (5/3), RC+RF (6/2), and RC+RF (Full).

The Baseline configuration is the conventional GPU which only consists of an SRAM-based register file, and STT-RF consists of only an STT-MRAM-based register file. RC+RF presents the hierarchical register file consisting of the SRAM-based register cache with STT-MRAM registers. Among all RC+RF configurations, RC+RF (3/5), RC+RF (4/4), RC+RF (5/3), and RC+RF (6/2) have the direct-mapped register cache. The first and second numbers inside each bracket present the number of sampled bits of warp ID and register ID, respectively. In the case of RC+RF (3/5), three bits of warp ID and five bits of register ID are used to compute the index. The partially sampled bits are always captured from the LSB. Lastly, RC+RF (Full) consists of a fully-associative SRAM-based register cache with the STT-MRAM registers. The detailed simulation configurations are presented in Section V.

As shown in the figure, the hierarchical register file with the direct-mapped register cache (RC+RF (3/5), RC+RF (4/4), RC+RF (5/3), and RC+RF (6/2)) dramatically improves the performance of applications compared to STT-RF since the SRAM-based register cache effectively mitigates the long write latency of STT-MRAM. Overall, RC+RF (5/3) shows the best performance among the direct-mapped register cache configurations, unlike the prior work proposing RC+RF (3/5) [11]. The reason for the different optimal points is that we evaluate the register cache performance on more recently released architecture, as we have listed in Section V. However, there is no one best direct-mapped cache configuration for all applications. In the case of BO, RC+RF (4/4) shows the best performance. MG shows the best performance in RC+RF (5/3), and VA shows the best performance in RC+RF (6/2). Note that in some applications, especially in NQU, RD, TP, VA, and MG, the hierarchical register file shows better performance than Baseline. This is because the cache entry is designed to store all the data of warps; thus, there are dramatic reductions in the register bank conflicts.

Based on our observation, the performance correlates strongly with the number of register write operations. FIGURE 5 shows the normalized write operations of seven

FIGURE 4. Normalized IPC for seven different configurations: Baseline, STT-RF, RC+RF (3/5), RC+RF (4/4), RC+RF (5/3), RC+RF (6/2), and RC+RF (Full).
different configurations. In the case of Baseline and STT-RF, there is no register cache; thus, all the write operations are always performed into the registers directly, named as direct-write (gray color). In the case of RC+RF, the register writes are only performed by the write-back operations moving data from the register cache to the STT-MRAM-based registers. Thus, each result of RC+RF refers to the number of write-back operations among all write operations. When the register cache absorbs the write operations more effectively, the value of write-back operations decreases. In summary, for RC+RF, the configuration with the lower normalized write-back value shows better performance for the applications as presented in FIGURE 4.

In RC+RF, such write-back operations are triggered by three different types of contentions: inter-warp, intra-warp, and false contentions. FIGURE 6a and FIGURE 6b show examples of inter-warp and intra-warp contentions, respectively. In the example of inter-warp contention, three bits of warp ID and five bits of register ID are used to compute the index of a register cache entry. When Warp #1 and Warp #9 attempt to write the computed values into Register #2, the computed index for both warps becomes the 34th (00100010) entry of the register cache. Thus, the computed values are stored in the same register ID in the same order for all warps. Due to such property, sampling partial bits of warp ID for computing the index increases contentions between warps for specific cache entries, and we name it as inter-warp contentions. As we sample more bits from the register ID, the sampled bits of warp ID must be decreased. In such case, the inter-warp contentions are more frequently observed because more warps with the same register ID have to share the one cache line.

Another type is an intra-warp contention that is caused when a single warp with multiple register IDs attempts to access the same cache line. Let us assume that we sample five bits of warp ID and three bits of register ID to compute the register cache index as shown in FIGURE 6b. In this example, Warp #9 attempts to write the computed value into Register #2 and #10, then the computed register cache index for both registers becomes the 74th (01001010) entry of the register cache. Similar to the previous example, to write the computed value of the subsequent write operation, the previously written value must be moved from the register cache to the STT-MRAM-based registers. The number of registers per thread is determined at the compiler time based on the compiler option or the property of applications [29], [30]. Therefore, the number of registers per thread varies from application to application. The HI application needs 36 32-bit registers per thread, while the BFS application uses only 16 32-bit registers per thread. As we sample more bits from the warp ID, the sampled bits of register ID must be decreased.
In this case, the intra-warp contentions are more frequently observed because only limited bits of register ID are sampled, and the probability of having the same value from the sampled bits between two registers is higher. On top of that, the intra-warp contentions are increased when applications require more registers per thread.

In FIGURE 5, for the RC+RF configurations, the normalized writes are divided into two categories: inter-warp and intra-warp contentions. The top (orange) and bottom (blue) of each bar indicate the inter-warp and intra-warp contentions, respectively. In RC+RF (3/5), among all write-back operations, an average of 90.8% and 9.2% of write-back operations are performed due to the inter-warp and the intra-warp contentions, respectively. The inter-warp contentions decrease, and the intra-warp contentions increase when the configuration is changed from RC+RF (3/5) to RC+RF (5/3). In RC+RF (5/3), an average of 38.5% and 61.5% of write-back operations are performed due to the inter-warp and the intra-warp contentions, respectively. In RC+RF (6/2), all the write-back operations are performed by intra-warp contentions since the baseline GPU architecture can schedule only up to 64 warps which can be presented using only six bits. Therefore, there are no contentions in between warps.

Lastly, the register cache contentions can be caused by the false contentions. If the number of sampled bits from register ID or warp ID is determined without considering the register usage of applications, the number of bits sampled from the register ID can possibly present a larger number than the number of registers used by a thread. In addition, the number of warps that can be expressed by the number of sampled bits from the warp ID can be greater than the number of warps actually scheduled on SMs. For example, in the RD application, threads use a total of 14 registers, and only four bits are needed to present all register IDs (from 0001 to 1110). If the RD application is executed in the RC+RF (3/5) configuration, then five bits of register ID are used to compute the index. In this case, the fourth bit (assuming the LSB is the 0th bit) of the sampled register ID always remains zero, and the cache entries with the fourth bit equal to one never be accessed during the execution of the kernel. FIGURE 7 shows the number of writes to the register cache entries in the RD application for the RC+RF (3/5) configuration. As shown in the figure, no write operations are performed from the 15th through the 32nd cache entries. Similar results are observed from the 47th entry, the 79th entry, and so on. Due to the unaccessed cache lines, write operations are heavily biased into the limited number of cache entries, which we name it as false contentions. The false contentions can also be observed in opposite configurations. In one of the NQU kernels, only 18 warps are scheduled because each thread needs many registers for the computation. In this case, the fifth bit of warp ID always remains zero; therefore, it is better not to use the fifth bit of warp ID to compute the cache index. In FIGURE 5, for RC+RF, if register write operations are all performed by either the inter-warp or intra-warp contentions, then the contentions are caused by the false contentions.

Such problems can be simply resolved if the cache is changed from the direct-mapped register cache into the fully-associative register cache. In the case of a fully-associative register cache, the computed data can be stored in any position of the register cache, unlike the direct-mapped register cache. In FIGURE 5, RC+RF (Full) shows the number of write-back operations in the fully-associative register cache. Overall, RC+RF (Full) shows the lowest write-back operations on average, and only 3.2% lower performance than Baseline. In RC+RF (Full), the true LRU replacement policy is used to select cache lines [31], [32]. Because of such a policy, all the cache lines are accessed, and the lastly accessed cache line is always removed from the register cache; thus, there are no false contentions, and inter-warp and intra-warp contentions are greatly reduced.

However, it is difficult to implement a fully-associative register cache due to the hardware complexity [33]–[36]. There are two major issues to implement the fully-associative register cache. Firstly, the access latency of the fully-associative cache is longer than the direct-mapped cache. Unlike the direct-mapped cache design, in the fully-associative cache, the data can be stored in any position. Thus, all the entries must be searched to find the desired data from the cache [36]. Also, additional hardware logic must be included to support the search mechanism. The second problem is the extraordinary complexity of true LRU replacement policy [37]. In the case of the true LRU replacement policy, the last accessed cache line is evicted from the cache when
I. Jeong et al.: TEA-RC: Thread Context-Aware Register Cache for GPUs

FIGURE 8. Correlation between the number of register bits and the number of warp bits.

There is no empty entry to store the incoming data. In order to detect the least recently used cache line, the cache must have the access history of all cache entries and include the logic to select the entry based on the access history. In this paper, instead of integrating a fully-associative register cache into the hierarchical register file, we propose a new cache index computation scheme considering the correlation between the number of scheduled threads and the number of registers required by threads.

IV. THREAD CONTEXT-AWARE REGISTER CACHE

Three factors determine the number of threads that can be scheduled on SMs: the size of the register file, the size of shared memory, and the maximum thread count limit [20], [25], [26]. TBs are scheduled on SMs until reaching one of the limitations. Therefore, there is a strong correlation between the number of scheduled threads and the number of registers used by threads. When threads need many registers to compute instructions, the number of threads scheduled on SMs is decreased and vice versa.

FIGURE 8 shows the correlation between the number of registers and the number of warps scheduled on SMs. If threads cannot be scheduled on SMs due to the shared memory size limitation, the number of threads scheduled on SMs is simply decreased regardless of the register file size and thread count limit. The bar indicates the number of scheduled warps based on the number of registers used by threads. When a kernel needs eight registers per thread, then a total of 64 warps can be scheduled on SMs (the second bar in the figure). The number of scheduled warps starts to decrease when threads need more than 32 registers. Threads need many registers to compute instructions, the number of threads scheduled on SMs is decreased and vice versa.

The yellow line indicates the number of bits needed to present all the scheduled warps, and the black line indicates the number of registers used by threads. The blue line represents the sum of warp ID and register ID when both warp and register bits are fully used, which is not possible. Since the baseline GPUs can schedule up to 64 warps and threads can use up to 256 registers, the blue line is always set to 14 (six bits for warps and eight bits for registers). The red line indicates the sum of actually used bits for warps and registers. When 64 warps are scheduled on SMs and threads use eight registers (the second bar on the left), six bits \( \log_2(64) \) and three bits \( \log_2(8) \) are needed to present all warp IDs and register IDs, respectively. Then the total number of bits that must be considered during computing the cache index is nine bits instead of 14 bits, which means that five bits of register ID always remain zero. The number of bits actually used in registers and warps only increases up to 12 bits, which means that up to two bits in the register ID and/or warp ID always remain zero.

In addition, the number of bits actually used for warp ID and register ID is even smaller than 12 bits when the threads use less than 36 registers. We measure the number of bits actually used in the benchmark applications, which are shown in FIGURE 9. The figure only shows the actually used bits for the first kernel from every application for simplicity. The top (orange) and bottom (blue) of each bar refer to the number bits of warp ID and register ID, respectively. The figure shows that most benchmark applications use less than 36 registers per thread; therefore, the number of bits needed to compute the cache line index is less than 12 bits. Based on these observations, we propose the TEA-RC that automatically captures the effective ranges of register ID and warp ID.

FIGURE 10 shows the proposed cache index computing scheme. In the conventional indexing scheme, the partially selected bits of warp ID and register ID are used to compute the index, as shown in FIGURE 3. In the proposed TEA-RC, all the bits of the warp ID and register ID are used to compute the index instead of selecting a portion of bits. The proposed architecture first reverses the position of bits in the warp ID ⨁, so the original MSB locates in the LSB position, and the original LSB locates in the MSB position. Then, the

FIGURE 9. Number of used bits for warp ID and register ID.
I. Jeong et al.: TEA-RC: Thread Context-Aware Register Cache for GPUs

FIGURE 10. Index computing steps of TEA-RC.

reversed warp ID and the original register ID are now used for inputs of the XOR gates ②. As shown in FIGURE 10, more bits are needed for the register ID compared to the bits of warp ID. Therefore, in order to perform XOR operations, we set align based on the MSB of the reversed warp ID and the MSB of the register ID.

Such a computing scheme is very effective in computing the cache index. As shown in FIGURE 8, the number of scheduled threads and the number of registers used by threads have a strong correlation. If a few registers are needed for threads scheduled on SMs, more threads can be scheduled on SMs; in contrast, if the threads require many registers, the TB scheduler assigns a less number of issuable threads on SMs. If only a few registers are used, the bits from the warp ID have a more significant influence on the output of the index since the reversed warp ID is used for computing the cache index, and the bits from the register ID have less influence on the output of the index and vice versa. Our proposed scheme, in the worst case, can cause contentions by overlapping four bits in the warp ID and register ID; however, as shown in FIGURE 8 and FIGURE 9, on average, up to two bits are overlapped, which yields the cache contentions. With our proposed method, there are no false contentions since all bits are always considered for computing the cache index, and there are no intra-warp contentions since bits from register ID are not overlapped when computing the index. Thus, the overlapped bits only produce the inter-warp contentions. Based on our evaluation results, our proposed index computing technique shows similar performance to the fully-associative register cache.

V. METHODOLOGY & EVALUATION

This section presents the methodology to verify our proposed TEA-RC design. Then, we show the performance improvement of our proposed cache design compared to prior work and present the detailed energy analysis of the proposed register cache.

TABLE 2. Simulation configuration.

| GPU Parameters                | Value          |
|-------------------------------|----------------|
| Number of SMs                 | 80             |
| Warp size                     | 32 threads     |
| Maximum number of threads / SM| 2,048 (64 warps) |
| Number of warp schedulers / SM| 4 schedulers   |
| Warp scheduling policy        | GTO            |
| Number of SRAM register cache entries | 256 entries |
| SRAM register cache entry size | 1,024 bits    |
| Number of SMMRAM register banks | 8 banks       |
| SMMRAM register bank width    | 32 bits        |
| Max. number of SMMRAM registers | 65,536 registers |
| L1 cache size / SM            | 32 KB          |
| L1 cache access latency       | 20 cycles      |
| Shared memory size/SM         | 96 KB          |
| Shared memory access latency  | 20 cycles      |
| L2 cache size                 | 4.3 MB         |
| L2 cache access latency       | 160 cycles     |
| Interconnection               | Crossbar (32B bit) |

TABLE 3. Benchmark applications.

| Application Name | Abb. | Num. of Instructions |
|------------------|------|----------------------|
| N-queens solver  | NQU  | 21M instructions     |
| Binomial options | BO   | 1B instructions      |
| Fast-walsh       | FWT  | 1B instructions      |
| transform [39]   |      |                      |
| Histogram        | HI   | 1B instructions      |
| Merge sort [39]  | MS   | 1B instructions      |
| Reduction [39]   | RD   | 239M instructions    |
| Sorting network  | SN   | 1B instructions      |
| Transpose [39]   | TP   | 1B instructions      |
| Vector addition  | VA   | 10M instructions     |
| MRI-gridding [40]| MG   | 1B instructions      |
| Breadth-first search [41]| BFS | 554M instructions |
| LUD [41]         | LUD  | 1B instructions      |
| Stream cluster   | SC   | 1B instructions      |
| 2D convolution   | 2D   | 218M instructions    |
| Scan [43]        | SCN  | 1B instructions      |
| Stencil 2D [43]  | S2D  | 1B instructions      |

A. METHODOLOGY

We use GPGPU-Sim 4.0 [44], a cycle-driven simulator, to verify the performance of the proposed architecture. The baseline GPU architecture has a total of 80 SMs, and each is able to execute up to 2,048 threads (64 warps) concurrently. Each SM has four warp schedulers that are based on the greedy-then-oldest (GTO) scheduling policy. Similar to prior work, the register cache has a total of 256 entries, each of which can store 1,024 bits. The detailed simulation configurations are listed in TABLE 2. A total of 16 applications are used to verify the performance and energy consumption of our proposed architecture. The applications, listed in TABLE 3, are chosen from six benchmark suites: ISPASS2009 [38], NVIDIA SDK [39], PolyBench [42], Rodinia [41], SHOC [43], and Parboil [40]. In benchmark suites, there are several applications showing similar execution behavior; in this case, we choose one representative application among them. For example, BFS applications can
be found in ISPASS2009 [38], Rodinia [41], and SHOC [43], and exhibit similar improvements in performance and energy efficiency across all applications. Thus, we choose the one from the Rodinia [41]. Lastly, all our applications are executed from the beginning up to 1 billion instructions using the simulator.

In addition, NVSim [45], a circuit-level simulator, is used for measuring the energy consumption of SRAM-based register cache and STT-MRAM-based registers. We use parameters listed in Table 1 to estimate the leakage energy consumption and per-access dynamic energy consumption of SRAM-based register cache and STT-MRAM-based registers, respectively. The overall dynamic energy consumption of register caches and registers is estimated using per-access dynamic energy results calculated by NVSim and access counts measured by GPGPU-Sim. The leakage energy consumption is estimated using the execution cycles driven by GPGPU-Sim and the computed leakage energy results by NVSim.

### B. PERFORMANCE ANALYSIS

FIGURE 11 shows the normalized IPC results of six different configurations: Baseline, STT-RF, RC+RF (3/5), RC+RF (BEST), RC+RF (Full), and TEA-RC. RC+RF (3/5), which is the configuration proposed by prior work [11], and RC+RF (Best), TEA-RC shows an average of 28.5% and 3.2% better performance, respectively. Note that our proposed architecture is 1.8% slower than Baseline and 1.4% faster than RC+RF (Full). In NQU and MG, TEA-RC shows much higher performance than Baseline. This is because the register cache reduces the bank conflicts that are observed frequently in the Baseline configuration. In NQU, BO, FWT, MS, BFS, and S2D, TEA-RC shows higher performance than RC+RF (Full). Due to the scheduling order, in the fully-associative register cache, some warps could occupy more cache entries than the other warps; thus, the inter-warps contentions are more frequently observed in some cases. However, in TEA-RC, the maximum cache entries that warps can access are evenly distributed. Each warp can only remove one cache line among cache entries that the warp can access. Therefore, TEA-RC shows better performance than RC+RF (Full) by reducing inter-warps contentions in some applications.

As we mentioned in Section III, the number of write-back operations is strongly correlated to the performance of the hierarchical register file. FIGURE 12 shows the normalized register write operations for Baseline, STT-RF, RC+RF (3/5), RC+RF (Best), RC+RF (Full), and TEA-RC. For Baseline and STT-RF, there is no register cache; thus, all writes are performed directly to registers. For RC+RF and TEA-RC, the register writes are only performed by the write-back operations, moving data from the SRAM-based register cache to the STT-MRAM registers. Based on our evaluation, TEA-RC produces 28.3 percentage points and 2.9 percentage points...
less write-back operations compared to RC+RF (3/5) and RC+RF (Best), respectively. In addition, TEA-RC reduces the write-back operations by 4.1 percentage points compared to RC+RF (Full). Overall, our proposed TEA-RC generates the write-back operations similar to RC+RF (Full) and shows a similar performance to Baseline.

C. ENERGY ANALYSIS

FIGURE 13 shows the normalized register file energy consumption. The results are normalized to Baseline. Because of the low leakage energy consumption of STT-MRAM, STT-RF shows 51.7 percentage points lower energy consumption than Baseline. In the case of the hierarchical register file, the write-back operations are greatly reduced due to the register cache; therefore, a large amount of write energy consumption is reduced. In RC+RF (3/5), an average energy consumption is 61.1 percentage and 9.4 percentage points lower than Baseline and STT-RF, respectively. In RC+RF (Best), average energy consumption is 7.8 percentage points lower than RC+RF (3/5). With our proposed architecture, the write-back operations are further reduced. Overall, TEA-RC reduces an average energy consumption of 70.3 percentage, 18.6 percentage, and 9.1 percentage points compared to Baseline, STT-RF, and RC+RF (3/5), respectively. In TEA-RC, the leakage and dynamic energy consumption of SRAM-based register cache account for 19.5% and 11.5% of the total hierarchical register file, respectively. Also, the leakage and dynamic energy consumption of STT-MRAM-based registers account for 34.5% and 34.5% of the total hierarchical register file, respectively. Based on prior work, the register file of GPUs consumes approximately 15–20% of total GPU energy [9]–[13]. With TEA-RC, it is expected to reduce the total energy consumption of GPUs by approximately 10.6–14.1%. Note that the energy consumption of a fully-associative register cache cannot be accurately estimated using the circuit-level simulator. However, it is expected that the energy consumption of the fully-associative register cache is always higher than TEA-RC due to the additional hardware overhead for searching the desired cache entry and the LRU replacing policy as we have mentioned in Section III.

D. DATA CACHE ANALYSIS

We measure the L2 data cache miss rate to verify how the proposed architecture affects the memory subsystem. FIGURE 14 shows the L2 data cache miss rates of Baseline, RC+RF, and TEA-RC. The proposed architecture only changes the access pattern inside the register file, so there is no significant change in the L2 data cache miss rate. Similar results are observed in the L1 data cache. In the case of BO, all the memory requests are for the constant cache or the shared memory; thus, there are no L2 data cache accesses.
SMs. Also, the register file in each SM is divided into sub-cores capable of executing a portion of warps scheduled on multiple sub-cores, and each sub-core has one warp scheduler. In the recently released GPU architecture [7], each SM consists of four sub-cores; each has one warp scheduler capable of executing 16 warps concurrently and one-quarter size of the register cache and registers. Thus, each sub-core has a 64-entry register cache, and six bits are used for computing the index of the register cache. We measure the performance of the conventional register cache by varying the partially selected bits of warp ID and register ID (RC+RF). The first and second numbers inside the bracket present the number of select bits of warp ID and register ID to compute the index of the register cache, respectively.

We also measure the performance of TEA-RC by changing the number of register cache entries. FIGURE 16 shows the performance result of TEA-RC varying the number of the register cache entries. Whenever the number of cache entries doubles, the performance of TEA-RC dramatically increases. When the cache entry can be accessed in one clock cycle, increasing the number of cache entries from 256 to 512 improves the performance by 37.3%. As we have mentioned in the previous sub-section, the register cache can reduce the bank conflicts of the main registers because all data of threads within each warp are stored in one cache line. Therefore, increasing the size of the cache entry can significantly increase the performance of TEA-RC. The improved performance ratio decreases as the access latency of cache entry increases (reducing the size of cache entry). When the cache access latency is increased to 16 cycles, increasing the number of cache entries from 256 to 512 shows only 1.0% performance improvement. Overall, 256 entries are large enough to minimize energy consumption while maintaining the GPU performance. Increasing cache entries could be one possible solution to improve the performance of GPUs by minimizing register bank conflicts.

E. SENSITIVITY STUDY OF REGISTER CACHE
In the default design configuration, each cache entry is designed to store 1,024 bits storing all data of threads in one warp. The cache access latency is increased when the size of cache entry decreases. FIGURE 15 shows the performance result varying the access latency of the register cache. As increasing cache access latency from 1 cycle to 2 (512 bits per entry), 4 (256 bits per entry), 8 (128 bits per entry), and 16 (64 bits per entry) cycles, the performance of TEA-RC decreases 0.2%, 1.3%, 4.7%, and 22.3%, respectively.

We also measure the performance of TEA-RC in the sub-core GPU architecture. Similar to the NVIDIA Volta GPU architecture [7], we divide SMs into four sub-cores; each has one warp scheduler capable of executing 16 warps concurrently and one-quarter size of the register cache and registers. Thus, each sub-core has a 64-entry register cache, and six bits are used for computing the index of the register cache. We measure the performance of the conventional register cache by varying the partially selected bits of warp ID and register ID (RC+RF). The first and second numbers inside the bracket present the number of selected bits of warp ID and register ID to compute the index of the register cache, respectively.

FIGURE 17 shows the performance results in the sub-core GPU architecture. Based on the evaluation results, RC+RF (3/3) shows the best performance among RC+RF (1/5), RC+RF (2/4), RC+RF (3/3), RC+RF (4/2), and RC+RF (5/1). RC+RF (3/3) shows the 83.1% higher performance than STT-RF and is 13.0% slower than Baseline. Our proposed TEA-RC shows the 95.9% and 7.0% higher performance than STT-RF and RC+RF (3/3), respectively. Note that TEA-RC is only 5.6% slower than Baseline.

VI. RELATED WORK
To the best of our knowledge, this paper is the first paper that proposes the register cache indexing scheme, considering the knowledge of the correlation between the number of scheduled warps and the number of registers used by threads. In this section, descriptions of prior work highly related to this study are divided into four categories.

A. NVM REGISTER FILE
Many researchers architected the GPU register file using emerging NVM memory technologies. It is well-known that NVM memory technologies have several advantages over SRAM, such as lower static energy consumption and higher density [11], [14], [16], [19], [46], [47]. Li et al. proposed STT-MRAM-based register file in GPUs [14]. In this study, SRAM-based write buffers are inserted in front of each register bank, and all the register writes are performed in the buffer. While executing other instructions, the data stored in the write buffer is moved to the STT-MRAM registers.
Deng et al. proposed warp-scheduler friendly STT-MRAM and SRAM hybrid GPU register file [46]. This study enables silent data transfer from SRAM to STT-MRAM considering the warp scheduling order, which minimizes the register file bank conflicts. Liu et al. proposed warp rescheduling policy for STT-MRAM-based register file [47]. The proposed policy changes the scheduling order to minimize the waiting time of issued warps considering the register bank access. Such designs can lessen the performance loss caused by the long write latency of STT-MRAM; however, all the writes must be performed into the STT-MRAM, which cannot resolve the write endurance problem of STT-MRAM. Jeon et al. proposed a hierarchical register file, which consists of an SRAM-based register cache and STT-MRAM registers [11]. This design only performs the write operations in STT-MRAM registers if there are no entries to store incoming data in the SRAM-based register cache; therefore, the number of STT-MRAM writes is significantly reduced, and it also overcomes the STT-MRAM write endurance problem. Mittal et al. proposed SOT-MRAM-based register file in GPUs [16]. This paper revealed that unchanged data are repeatedly stored in the SOT-MRAM registers. Therefore, these duplicated write operations are removed. The NVM-based register files are attractive solutions to reduce the register energy consumption due to their lower leakage power than that of SRAM. However, the write operations must be minimized to prevent losing the performance due to their longer write latency and to resolve the write endurance problem. Our proposed TEA-RC can be one potential solution to minimize such writes. Also, the proposed technique can be implemented with future NVM memory technologies.

**B. REGISTER RENAMING**

Several previous studies proposed the register file virtualization technique to improve the performance of GPUs or minimize the register file energy consumption. Jeon et al. propose the register file virtualization in GPUs [18]. The baseline GPUs allocate and deallocate registers when the TBs are scheduled on SMs and after the TBs are executed, respectively. In the proposed technique, the dead registers that are no longer used by TBs are released right after the last use. Then, the register usage can be minimized, and it reduces the register file’s static and dynamic energy consumption. Kim and Ro revealed that the same instructions with the same input values are frequently executed in GPUs [48]. The same instructions with the same input values always produce the same output values. Based on this observation, warp instruction reuse (WIR) architecture is proposed, which eliminates the instructions that produce the same output values by renaming the physical registers. The proposed architecture reduces the energy consumption of execution units and the register files in SMs. These renaming techniques can be applied to our proposed architecture and can further reduce the register file energy consumption.

**C. REGISTER COMPRESSION**

In GPUs, threads in a warp execute the same instruction simultaneously on different data values. Lee et al. revealed that there is a strong value similarity between threads within a warp [10], [15]. By exploiting the fact, the register compression technique known as warped-compression is proposed, which compresses the data of threads after executing instructions. The compressed data is decompressed after reading the data from the register file. Due to the compression, the number of bits stored in the register file is decreased, and it helps to reduce the number of reads and writes in the register banks. Zhang et al. proposed a similar data compression technique for the emerging NVM register file [49]. Jeon et al. applied the similar compression technique in the hierarchical register file [11]. These compression techniques are orthogonal to our designs and can be included with our proposed architecture to further reduce register file energy consumption.

**D. COMPILER OPTIMIZATION**

Several prior work uses the compiler to analyze the register liveness to improve the performance or the energy efficiency of GPUs. Xie et al. proposed coordinated register allocation and thread-level parallelism (CRAT) to maximize the performance of GPU by analyzing the trade-off between the performance of a single thread and TLP [50], [51]. CRAT uses a compiler to analyze register liveness and determines the optimal TLP considering the performance of a single thread. Oh et al. proposed the FineReg architecture [20]. In FineReg, the compiler detects the liveness of registers to determine the optimal context switching timing of threads in GPUs. With the timing information, FineReg supports the fast context switching technique to improve TLP in GPUs. Esfeden et al. proposed breathing operand windows (BOW) to exploit bypassing in GPUs [9]. In BOW, the compiler detects the register data dependency information. Using the dependency information, the computed data is stored directly in operand collectors, which minimizes the register file access. Our TEA-RC can be implemented with these techniques to further improve the energy efficiency of GPUs.

**VII. CONCLUSION**

This paper proposes TEA-RC to minimize the write-back operations in the hierarchical register file. TEA-RC computes the index of the register cache by considering the strong correlation between the number of scheduled threads and the number of registers used by threads. The TEA-RC architecture minimizes the *inter-warp*, *intra-warp*, and *false* contentions, and it shows a similar performance with the hierarchical register file consisting of a fully-associative register cache and the baseline GPU architecture. Based on our intensive simulations, the proposed architecture shows 28.5% better performance and 9.1 percentage points lower energy consumption than the previously proposed register cache.
ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for their helpful feedback.

REFERENCES

[1] M. B. Taylor, “The evolution of Bitcoin hardware,” Computer, vol. 50, no. 9, pp. 58–66, 2017.
[2] S. McNally, J. Roche, and S. Caton, “Predicting the price of Bitcoin using machine learning,” in Proc. 26th Euromicro Int. Conf. Parallel, Distri.

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for their helpful feedback.

REFERENCES

[1] M. B. Taylor, “The evolution of Bitcoin hardware,” Computer, vol. 50, no. 9, pp. 58–66, 2017.
[2] S. McNally, J. Roche, and S. Caton, “Predicting the price of Bitcoin using machine learning,” in Proc. 26th Euromicro Int. Conf. Parallel, Distrib.

Netw.-Based Process. (PDP). Mar. 2018, pp. 339–343.
[3] X. Dong and Y. Yang, “Searching for a robust neural architecture in four GPU hours,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.
(CVPR), Jun. 2019, pp. 1761–1770.
[4] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, “Survey and benchmarking of machine learning accelerators,” in Proc.

IEEE High Perform. Extreme Comput. Conf. (HPoE), Sep. 2019, pp. 1–9.
[5] S. Morishima and H. Matsutani, “Accelerating blockchain search of full nodes using GPUs,” in Proc. 26th Euromicro Int. Conf. Parallel, Distrib.

Netw.-Based Process. (PDP), Mar. 2018, pp. 244–248.
[6] NVIDIA A100 Tensor Core GPU Architecture. Accessed: Jan. 28, 2022. [Online]. Available: https://images.nvidia.com/content/a100/pdf/a100-whitepaper.pdf
[7] NVIDIA Tesla V100 GPU Architecture. Accessed: Jan. 28, 2022. [Online]. Available: https://images.nvidia.com/content/tesla/v100/pdf/tesla-v100-whitepaper.pdf
[8] NVIDIA Tesla P100. Accessed: Jan. 28, 2022. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/pascalarchitecture-whitepaper.pdf
[9] H. A. Esfeden, A. Abdolrazhani, S. Rahman, D. Wong, and N. Abu-Ghazaleh, “BOW: Breathing operand windows to exploit bypassing in GPUs,” in Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2020, pp. 996–1008.
[10] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavararam, “Warped-compression: Enabling power efficient GPUs through register compres-

sion,” ACM SIGARCH Comput. Archit. News, vol. 43, no. 3R, pp. 502–514, 2015.
[11] W. Jeon, J. H. Park, Y. Kim, K. Goo, and W. W. Ro, “Hi-End: Hierarchi-

cal, endurance-aware STT-MRAM-based register file for energy-efficient GPUs,” IEEE Access, vol. 8, pp. 127768–127780, 2020.
[12] C.-C. Hisao, S.-L. Chu, and C.-C. Hsieh, “An adaptive thread scheduling mechanism with low-power register file for mobile GPUs,” IEEE Trans. Multimedia, vol. 16, no. 1, pp. 60–67, Jan. 2014.
[13] S.-L. Chu, C.-C. Hisao, and C.-C. Hsieh, “An energy-efficient unified register file for mobile GPUs,” in Proc. IFIP 9th Int. Conf. Embedded Ubiquitous Comput., Oct. 2011, pp. 166–173.
[14] 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., Jun. 2015, pp. 1–6.
[15] S. Lee, K. Kim, G. Koo, H. Jeon, M. Annavararam, 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.
[16] S. Mittal, R. Bishnoi, F. Oboril, H. Wang, M. Tahoori, A. Jog, and J. S. Vetter, “Architecting SOT-RAM based GPU register file,” in IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI), Jul. 2017, pp. 38–44.
[17] 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.
[18] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavararam, “GPU register file virtualization,” in Proc. 48th Int. Symp. Microarchitecture, Dec. 2015,

pp. 420–432.
[19] Y. Oh, I. Jeong, W. W. Ro, and M. K. Yoon, “CASH-RF: A compiler-assisted hierarchical register file for GPUs,” IEEE Embedded Syst. Lett., early access, Mar. 31, 2022. doi: 10.1109/LES.2022.3163749.
[20] Y. Oh, M. K. Yoon, W. J. Song, and W. W. Ro, “FineReg: Fine-grained register file management for augmenting GPU throughput,” in Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2018, pp. 364–376.
[21] D. Gope and M. H. Lipasti, “Bias-free branch predictor,” in Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture, Dec. 2014, pp. 521–532.
[22] H. K. Kim, H. S. Kim, C. M. Eun, H. H. Cho, and O. H. Jeong, “A high-performance branch predictor design considering memory capacity limitations,” in Proc. Int. Conf. Circuits. Syst. Simulation (ICCSS), Jul. 2017, pp. 49–53.
I. Jeong et al.: TEA-RC: Thread Context-Aware Register Cache for GPUs

[46] 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.

[47] X. Liu, M. Mao, X. Bi, H. Li, and Y. Chen, “Exploring applications of STT-RAM in GPU architectures,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 68, no. 1, pp. 238–249, Jan. 2021.

[48] K. Kim and W. W. Ro, “WIR: Warp instruction reuse to minimize repeated computations in GPUs,” in Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA), Feb. 2018, pp. 389–402.

[49] 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., Jun. 2016, pp. 1–6.

[50] X. Xie, Y. Liang, X. Li, Y. Wu, G. Sun, T. Wang, and D. Fan, “Enabling coordinated register allocation and thread-level parallelism optimization for GPUs,” in Proc. 48th Int. Symp. Microarchitecture, Dec. 2015, pp. 395–406.

[51] X. Xie, Y. Liang, X. Li, Y. Wu, G. Sun, T. Wang, and D. Fan, “CRAT: Enabling coordinated register allocation and thread-level parallelism optimization for GPUs,” IEEE Trans. Comput., vol. 67, no. 6, pp. 890–897, Jun. 2018.

IPOOM JEONG received the B.S. and Ph.D. degrees in electrical and electronics engineering from Yonsei University, Seoul, South Korea, in 2014 and 2020, respectively. He currently works as a Research Professor with the School of Electrical and Electronic Engineering, Yonsei University. His research experience includes a Hardware Engineer in memory business at Samsung Electronics. His current research interests include high-performance computer system design, energy-efficient CPU/GPU microarchitectures, and memory system design.

YUNHO OH (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, South Korea, in 2009, 2011, and 2018, respectively. He is currently working as an Assistant Professor with the School of Electrical Engineering, Korea University. Prior to joining Korea University, he worked as an Assistant Professor with the Department of Electrical and Computer Engineering, Sungkyunkwan University (SKKU). He also worked as a Postdoctoral Researcher with the Parallel Systems Architecture Laboratory (PARSA), EPFL, Switzerland, from 2019 to 2021. From 2011 to 2014, he was worked as a Software Engineer in mobile communications business at Samsung Electronics. His research interests include hardware and software architectures for energy-efficient datacenters, processor architectures (CPUs, GPUs, and neural network accelerators), in-storage processing, memory systems, AI acceleration systems, and high-performance computing.

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 at Irvine, 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.

MYUNG KUK YOON received the B.S. degree in computer engineering and computational mathematics from Washington State University (WSU), Pullman, Washington, USA, in 2011, and the Ph.D. degree in electrical and electronic engineering from Yonsei University, Seoul, South Korea, in 2018. He is currently working as an Assistant Professor with the Department of Computer Science and Engineering, Ewha Womans University. Prior to joining Ewha Womans University, he worked as a Software Developer at Samsung Inc. His research interests include CPU/GPU microarchitecture, machine learning accelerators, and parallel programming.