An eDRAM-Based Approximate Register File for GPUs

Donghwan Jeong, Young H. Oh, and Jae W. Lee
Sungkyunkwan University (SKKU)

Yongjun Park
Hongik University

Editor's notes:
GPUs require large register files for fast context switching. This paper presents a high-density and energy-efficient approximate register file architecture based on embedded DRAM with lowered refresh rate to the low-order bits of register entries.

—Qiang Xu, The Chinese University of Hong Kong

Graphical processing units (GPUs) are widely adopted in all scales of modern computing platforms to execute compute-intensive, data-parallel applications efficiently. In the high-performance computing (HPC) domain, billion-transistor GPUs are already the norm to accelerate scientific workloads [1]. In the mobile computing domain, a processor system-on-chip (SoC) typically integrates a GPU to offload compute-intensive tasks such as media processing and 3-D rendering. With strong demands for energy-efficient performance the number of processing elements on a GPU will continue to scale up in the foreseeable future.

GPUs are throughput-optimized execution substrates, which execute a large number of threads concurrently via time sharing. To keep processor cores busy in face of long-latency memory operations, GPUs count on zero-latency context switching among warps, where a warp is the set of threads (typically 32 threads) that are scheduled together to execute the same instruction. Therefore, GPUs feature a large register file to hold the context information of all active warps scheduled to each core. For example, the aforementioned GK210 GPU uses 7.5 MB of on-chip SRAM just for register files, which is larger than any other on-chip memory structures, such as caches and shared memory.

This multimegabyte SRAM-based register file already occupies a significant portion of the chip area. In addition, it is one of the most power-consuming components per area in GPU due to high leakage [2]. This overhead makes the register file a serious scalability bottleneck for future GPUs as the number of threads continues to increase.

To address this problem eDRAM-based register files have been recently proposed, which have higher density and lower leakage [3], [4]. However, unlike SRAM, eDRAM cells must be periodically refreshed to retain values. Even worse, with technology scaling, the cell capacitance will continue to be scaled down to reduce retention time, which in turn will increase refresh power [5]. As the refresh power is the major source of power consumption, it is an important design goal to minimize it when architecting an eDRAM-based register file. To this end, several refresh strategies have been proposed, which exploit register access

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/MDAT.2015.2500185
Date of publication: 12 November 2015; date of current version: 18 January 2016.

patterns of an instruction stream [3] and the liveness of register variables [4], for example.

However, these refresh strategies aim to provide 100% reliability of stored data and do not exploit opportunities for further energy savings for those applications that do not require perfect correctness. For example, floating-point values are approximate by nature due to limited precision, for which the notion of perfect correctness is irrelevant. Also, many video applications can tolerate a small number of bit errors without degrading user experience. Thus, we can exploit precision-power tradeoffs to minimize refresh power by controlling refresh rates, while still providing good enough output quality.

This paper introduces an eDRAM-based approximate register file for GPUs. The proposed register file dynamically selects the refresh rate for each 32-element register entry based on the property of the value it holds, either precise or approximate. If the register entry holds precise values, it will be refreshed at the nominal rate; otherwise, at a lower rate. To ensure acceptable output quality, the higher order bits of an approximate value are still refreshed at the nominal rate as they have greater impact on the output quality than the lower order bits. The GPU pipeline is extended to automatically identify floating-point values, which are the primary target of approximation in this paper, by tracking the register operands of every floating-point instruction. This obviates the needs for user intervention or compiler support to mark approximate values. Evaluated with 22 programs taken from Polybench [6], Rodinia [7], and Parboil [8] benchmark suites, the proposed register file saves refresh power by 22% and 16% over a precise eDRAM-based register file at the 11-nm technology node with a root mean square error (RMSE) of 0.000045% and 0.005%, for double- and single-precision floating-point data, respectively, while increasing the transistor count of the eDRAM cell array by only 0.67%.

Background and motivation

Baseline GPU architecture

Figure 1 shows the baseline GPU architecture based on NVIDIAs Fermi architecture. The GPU has multiple streaming multiprocessors (SMs), each of which has 32 streaming processors (SPs). The warp scheduler selects two active warps that are ready to execute the next instruction. Since SPs internally run at 2× faster clock rate than the primary GPU clock, the warp scheduler can issue two instructions (one for each warp) every cycle. In other words, the GPU pipeline switches contexts (warps) every cycle. This frequent context switching mandates the GPU to retain the contexts of all active warps in an on-chip register file, instead of spilling into off-chip memory, to minimize context switching overhead. The total register file size amounts to 2 MB for Fermi and 7.5 MB for Kepler [1]. This is much larger than the last-level cache shared by all SMs, whose size is 768 KB for Fermi.

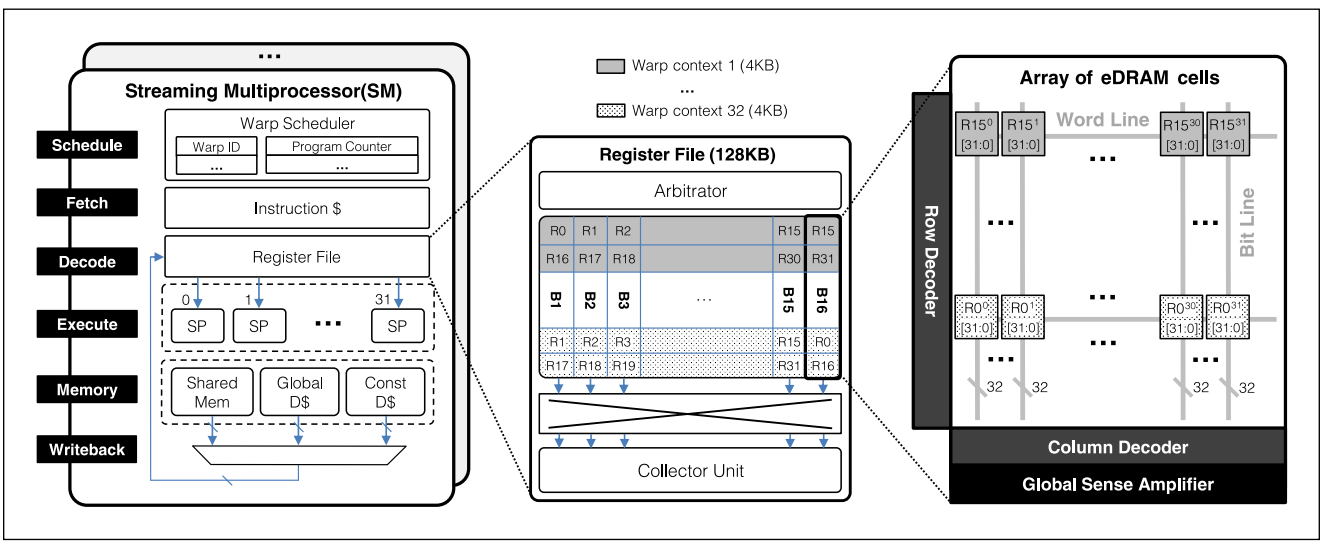

Figure 1. Baseline GPU architecture based on NVIDIA Fermi GPU.

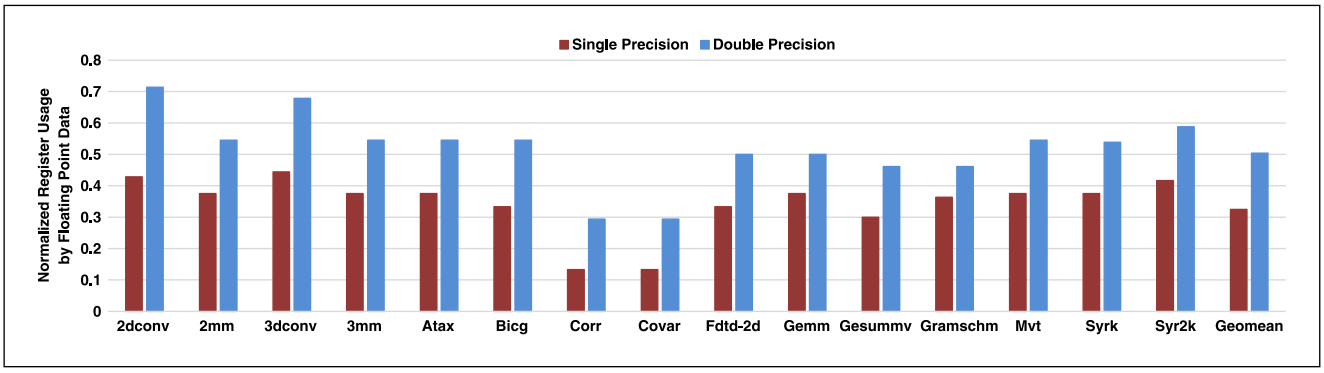

Figure 2. Portion of registers used to hold floating-point values for Polybench.

The register file size will continue to increase as both the number of SMs and the number of SPs per SM scale up over generations.

Fermi features the operand collector architecture, which emulates a multiported register file with a dual-ported multibanked SRAM (or eDRAM) array for area and power efficiency. If there is no bank conflict, it can provide one register entry every cycle, which a vector of 32 32-b registers sharing the same name (e.g., R15), to be used by the 32 threads of a warp. Note that the same register file is used by both integer and floating-point values. Our baseline eDRAM-based register file allocates one register entry (32 words) to an eDRAM row, which is read by a single row access and forwarded to the GPU pipeline.

eDRAM-based register files

An eDRAM cell uses much fewer transistors than an SRAM cell, which typically comprises six or more transistors, to provide higher density and lower leakage. However, like commodity DRAM, eDRAM cells also require periodic refreshes to retain stored bit values. Since the capacitance of an eDRAM cell is much smaller than a commodity DRAM cell, eDRAM requires much higher refresh rate. This results in higher power consumption and blocking time due to refresh operations, which are the main drawback of eDRAM. Even worse, this problem will be exacerbated with technology scaling. Hence, minimizing the refresh overhead is the primary challenge in architecting a large eDRAM-based register file for GPUs [3], [4].

While existing refresh-reduction techniques for both eDRAM [3], [4] and commodity DRAM [9], [10] can save a significant fraction of refresh power, additional savings can be achieved for applications that do not require perfect correctness. As many GPU workloads use floating-point data, ample opportunities for such savings exist because floating-point values are intrinsically approximate. For example, Figure 2 shows the portion of register usage by floating-point values with 15 Polybench programs [6], demonstrating up to 45% (3DCONV) and 72% (2DCONV) of registers can hold floating-point values for single- and double-precision cases, respectively. This work is the first to apply an approximation technique for energy savings to eDRAM-based register files on GPU. Unlike existing proposals for approximate memory components, such as Flikker (for commodity DRAM) [10] and Truffle (for SRAM-based register files and caches on CPU) [11], the proposed register file manages data criticality at a finer (subword) granularity to minimize error rate, while not requiring either user intervention or compiler support for identifying approximate data to incur no changes to the software stack.

eDRAM-based approximate register file

Register file organization

Figure 3 shows the bank structure of the proposed approximate register file. The eDRAM cell array is divided into two segments: critical and noncritical segments. The critical segment, which stores the upper half words of registers (i.e., bit numbers 31 through 16), is refreshed at the nominal rate to ensure reliable operation. The noncritical segment, which stores the lower half words of registers (i.e., bit numbers 15 through 0), is refreshed at a lower rate to save refresh power. To

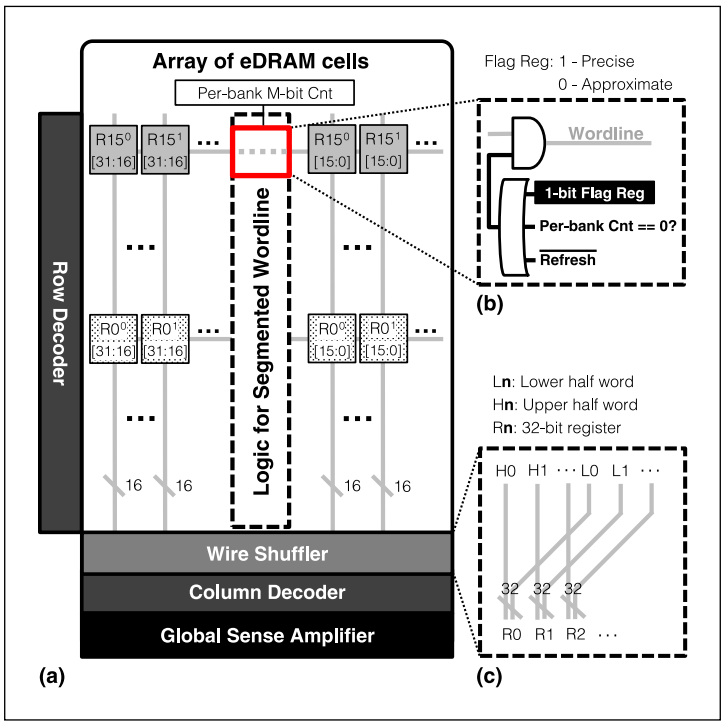

Figure 3. Proposed register file structure. (a) Single bank. (b) Segmenation logic. (c) Wire shuffler.

realize this, an eDRAM bank is augmented with the following hardware components.

1) Flag register: Each row has a one-bit flag register indicating whether the register entry (a vector of 32 words) associated with the row holds precise values (1) or approximate ones (0). Assuming a 128-kB register file per SM (with 16 8-kB banks), the flag registers will cost 128 B per SM.
2) *M*-bit counter: Each bank has an *M*-bit counter to control the refresh period for approximate register entries. It is incremented by one at every nominal refresh period (denoted by T_0). Then, the refresh period of the noncritical segment will be set to $2^M \times T_0$.
3) Segmentation logic: Each word line is segmented by a two-input AND gate and a three-input OR gate as shown in Figure 3b. The two subword lines for each row must be connected if either of the following conditions holds: a) regular read/write operation is being performed (shown by *Refresh* signal); b) the register entry is holding precise values; and c) the register entry is holding approximate values and the refresh period for the noncritical section has lapsed since the last refresh (i.e., *M*-bit counter becomes zero).
4) Wire shuffler: This block is placed right above the column decoder as shown in Figure 3c. When a register entry is written, the wire shuffler sorts incoming data bits by their criticality to cluster upper half words into the critical section and lower half words into the noncritical section. It restores the original bit ordering when the register entry is read out. This block is nothing but shuffled wires with relatively low wiring overhead.

Tracking approximate values

For the approximate register file to operate properly, approximate values must be identified and marked by the one-bit flag for the register entry that holds them. This flag bit can be set manually by the user, possibly with compiler/OS support, or automatically by hardware. This work adopts the latter by extending the GPU pipeline to identify floating-point values on the fly (which are approximate). Unlike CPUs, there is no separate floating-point register file for GPUs, and one cannot tell whether the stored values are floating point just by inspecting register IDs.

Instead, the GPU pipeline infers approximate data by tracking register entries that are read or written by floating-point instructions. The native Fermi Instruction Set defines 16 floating-point instructions, such as FADD (single-precision add), FMUL (single-precision multiply), FMNMX (single-precision minimum/maximum), DADD (double-precision add), etc. The source and destination registers for those instructions are inferred to be floating point (approximate) and their one-bit flags are updated when retiring them. The rest of the instructions updates the one-bit flag of the destination register entry to be precise but preserves the flags of the source registers.

Refresh strategies

The proposed register file saves refresh power by lowering the refresh rate for the less critical bits of an approximate value. It is our design decision to refresh the critical section at the nominal rate as the upper half word of a floating-point number contains a sign bit and exponent bits whose precision is critical to maintain acceptable output quality. However, it is feasible to extend the

proposed register file to support more than two segments (even with nonuniform widths) and/or refresh periods. Our evaluation shows that refreshes are still necessary to maintain a reasonable error bound. The worst case analysis with the noncritical section is discussed in a later section.

Evaluation

Evaluation Methodology

GPGPU-Sim [12] is used to model the proposed register file based on NVIDIAs GTX 480. Simulation parameters are tabulated in Table 1. We enable the PTX-plus simulation mode, which emulates precise register allocation using NVIDIA CUDA compiler, for simulation accuracy and approximate value tracking. We use all 15 programs from Polybench-GPU benchmark suite [6]. Seven programs that use at least 20% of total registers to hold floating-point values are also taken from Rodinia [7] and Parboil [8] benchmark suites. Note that, while the current implementation only targets floating-point data for approximation, there is no fundamental limitation on extending the ISA to accommodate user-identified, non-floating-point approximate data for greater energy savings.

1) Error injection methodology: We use the eDRAM bit error model by Cho et al. [5]. Using their model, we obtain the bit error rate for a given refresh period. We assume that bit errors are randomly distributed over the entire register file. At every nominal refresh period (T_0), we randomly select the same number of bits as the expected bit errors, and zero them. Note that a bit error occurs in a discharging direction only—from one to zero—but not in the opposite direction.
2) Power and retention time model: We adopt the power and retention time model of an eDRAM-based register file by Jing et al. [3] with 3T1D eDRAM cells as summarized in Table 1. Figure 4 shows the results of a sensitivity study with varying the value of M. With a constraint of maintaining the maximum RMSE to be less than 1% at 11-nm technology, the optimal value of M is 3, which sets the refresh period of the noncritical section to be $8T_0$. The nominal refresh period (T_0) is set identically to the cell retention time, which translates to 2048, 1024, and 512 cycles at the 22-, 16-, and 11-nm technology nodes, respectively. We assume no error occurs if cells are refreshed every T_0 cycles (or shorter).

Table 1 Simulation parameters.

Processor Parameter	Specification		
# of PE (per SM)	16		
# of SM cores	15		
Execution Model	In-order		
Max Warp per SM	48		
Size of register file	128 KB (per SM)		
# of register banks	16		
Shared memory	48 KB		
L1 Cache	16 KB		
L2 Cache	768 KB		
Clock frequency	1400 MHz (2× 700 MHz GPU clk)		
Warp Scheduler	Greedy then oldest		
eDRAM Parameter	**22nm**	**16nm**	**11nm**
Read Energy (fJ / access)	316	284	256
Write Energy (fJ / access)	149	134	121
Leakage Power (uW / bank)	41.5	45.7	50.2
Cell Retention (cycles)	2K	1K	512

Results

Output quality. To assess the impact of approximation on output quality, we use an application-specific output metric for each program as summarized in Table 2. For Polybench programs, which implement linear algebra kernels, we measure the RMSE of the program output against the original output with no approximation at the 11-nm technology node. The error rate generally decreases as we increase feature size or switch to double precision. For Rodinia programs, we select the output metric based on the descriptions of each program. For Parboil, we use a pass/fail output leveraging the Python script for output testing, included in the original distribution. We run the simulation at least three times and take the one with the highest error rate. The mean of RMSE over the 22 programs is very small—only 0.005% (and 0.000045% for double precision).

The worst case output quality degradation is also estimated by not refreshing the lower half word of approximate values and assuming a bit error rate of one (i.e., approximate bits are always flipped). The RMSE in this case is 29% while saving refresh energy only by additional 2%. Therefore, our refreshing scheme provides a nice balance in power-precision tradeoffs.

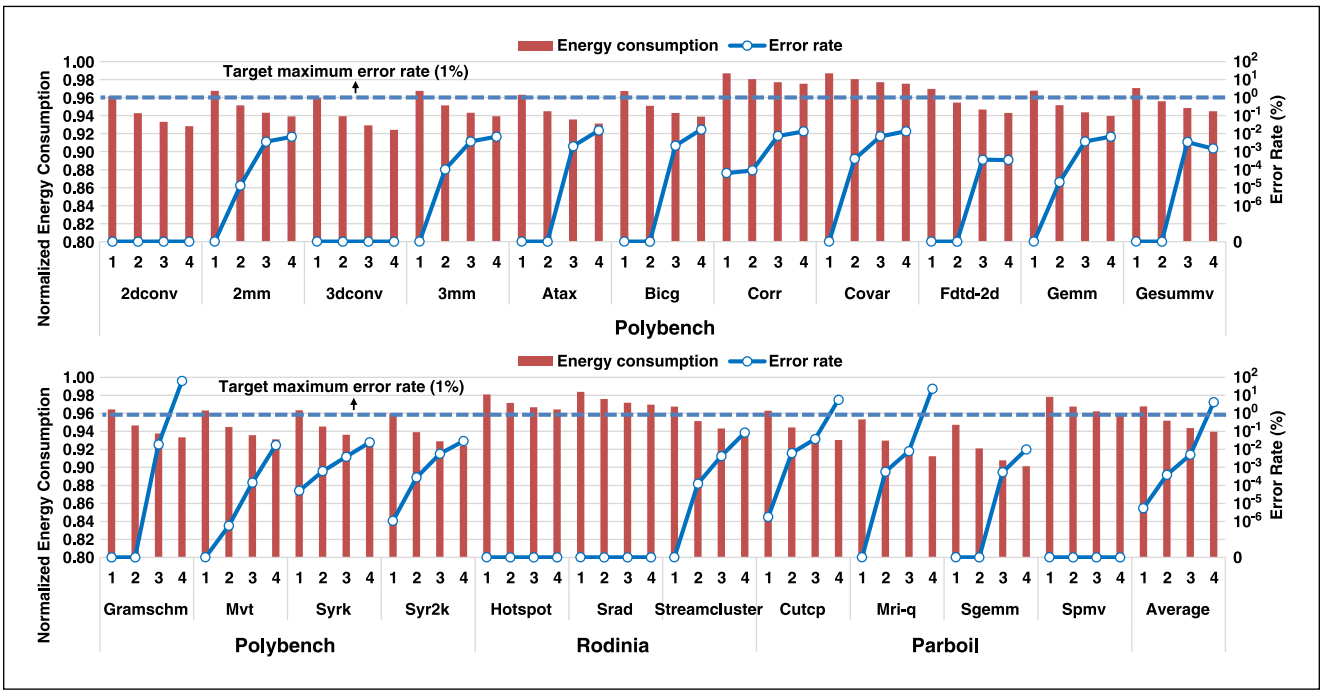

Figure 4. Sensitivity study with varying value of *M* (the width of the per-bank refresh counter) from 1 through 4.

Energy savings. Figure 5a shows refresh energy consumption of the proposed register file normalized to the precise eDRAM-based register file. It reduces refresh energy by about 16% and 22% for single- and double-precision programs, respectively.

Table 2 Output error rate for single-precision case at 11-nm node.

Benchmark	Program	Error Metric	Result
Polybench	2dconv	Max / RMSE	0.00% / 0.00%
	2mm	Max / RMSE	0.058% / 0.0036%
	3dconv	Max / RMSE	0.00% / 0.00%
	3mm	Max / RMSE	0.048% / 0.0037%
	Atax	Max / RMSE	0.20% / 0.0020%
	Bicg	Max / RMSE	0.050% / 0.0022%
	Corr	Max / RMSE	0.13% / 0.0074%
	Covar	Max / RMSE	0.13% / 0.0070%
	Fdtd-2d	Max / RMSE	0.0013% / 0.00036%
	Gemm	Max / RMSE	0.071% / 0.0037%
	Gesummv	Max / RMSE	0.047% / 0.0034%
	Gramschm	Max / RMSE	0.10% / 0.019%
	Syrk	Max / RMSE	0.16% / 0.0039%
	Syr2k	Max / RMSE	0.18% / 0.0057%
Rodinia	Hotspot	Max / RMSE	0.00% / 0.00%
	Srad	PSNR	Infinite
	Streamcluster	# mismatches	0
Parboil	Cutcp	Pass or Fail	Pass
	Mri-q	Pass or Fail	Pass
	Sgemm	Pass or Fail	Pass
	Spmv	Pass or Fail	Pass

The refresh energy savings are closely correlated with the floating-point register usage shown in Figure 2. Note that the refresh energy savings are independent of feature size as the numbers are normalized to the baseline eDRAM register file, hence only the total number of refreshes counts but not the energy consumption per refresh.

We also evaluate the total energy savings by taking into account refresh, dynamic, and leakage energy. Though the normalized refresh energy savings are preserved across technology nodes, the total energy savings actually increase as we scale down, because the relative portion of the refresh energy increases due to shorter retention time. Figure 5b shows total energy consumption normalized to the precise register file at the 22-, 16-, and 11-nm technologies, respectively. For single-precision programs, the total energy savings are 3% and 4% on average at the 22-nm technology node, and 6% and 8% (with maximum savings of 12%) at the 11-nm node for single- and double-precision cases, respectively.

The proposed technique is complementary to and can augment existing, non-approximation-based techniques to save refresh energy for both commodity DRAM and emerging eDRAM. To demonstrate this, we apply our technique to an

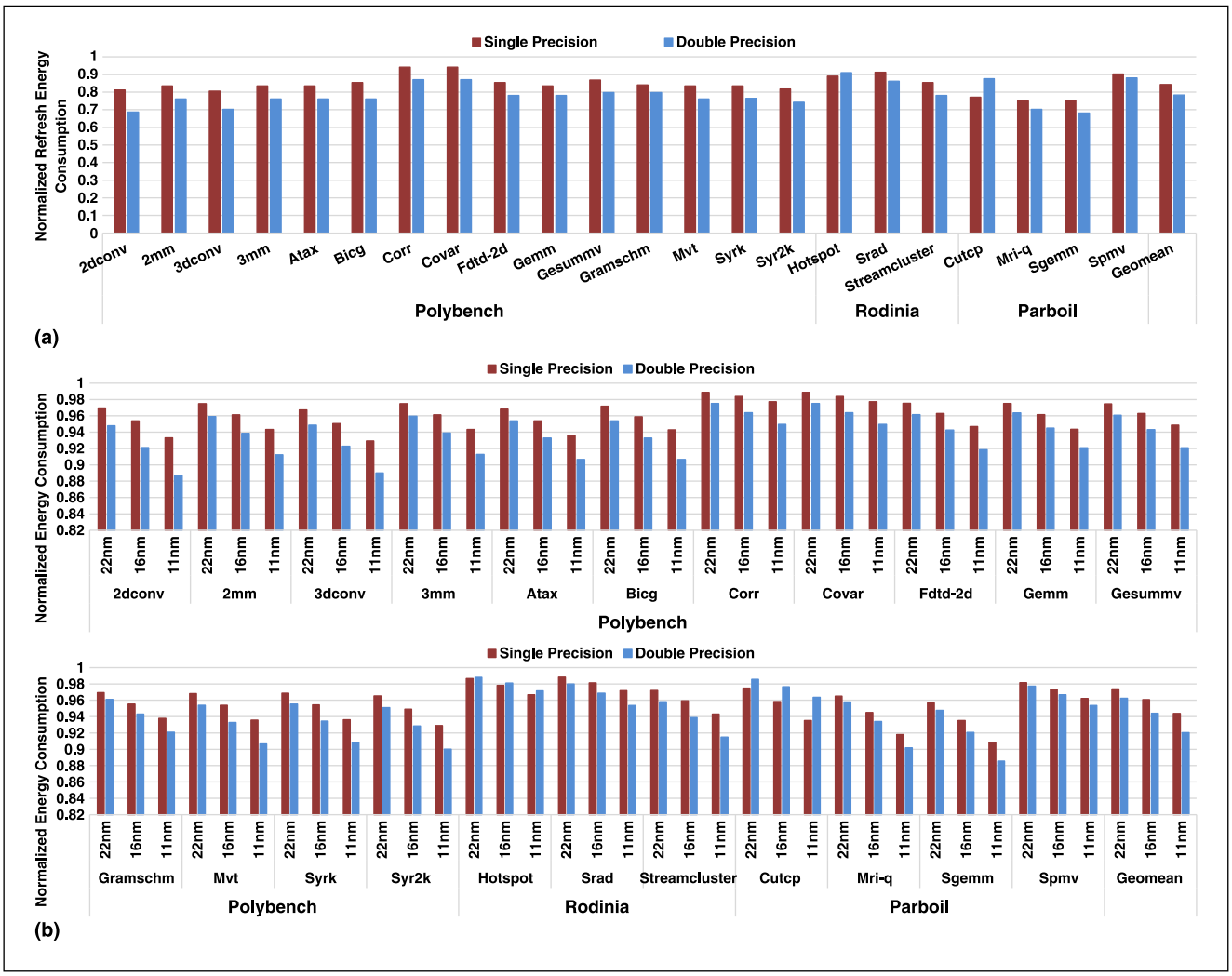

Figure 5. Energy consumption for all programs. (a) Refresh energy consumption normalized to precise eDRAM-based register file. (b) Total register file energy consumption normalized to precise eDRAM-based register file.

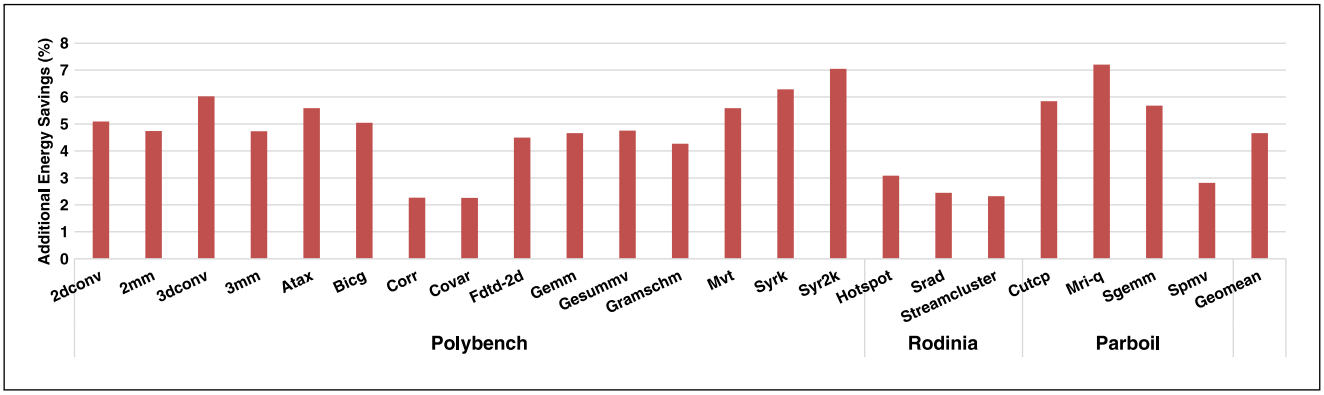

Figure 6. Additional energy savings achieved by our technique on optimized baseline register file capable of power gating.

optimized baseline with power-gating unallocated registers to achieve up to 7.2% of total energy savings in Figure 6.

Hardware cost. As a rough estimate of the hardware cost, we count the number of transistors added to the baseline. The proposed register file requires a flag register (6T), a two-input AND gate (6T), and a three-input OR gate (8T) for each row. It also uses a 3-b counter (36T) for each bank. Assuming 3T1D eDRAM cells, the approximate register file will increase the transistor count of an eDRAM bank by only 0.67%. This estimation is conservative as we do not count either the diodes in the cell array or transistors for peripheral blocks.

THIS PAPER MAKES a case for eDRAM-based approximate register files for GPUs. Many GPU workloads process floating-point values, which make a good target for approximation. The proposed approximate register file demonstrates significant energy savings without incurring any change to the software stack. By applying nonuniform refresh rates at a subword granularity, we ensure acceptable output quality. For 22 floating-point programs, the proposed register file saves refresh energy by up to 32% with geomean savings of 22% and 16% for double- and single-precision cases, respectively. In the future, we plan to investigate an ISA extension to expose control of flag bits to software, and quantify additional energy benefits coming from application-specific software optimizations, including compiler- and programmer-assisted approximation and hardware-guided feedback techniques. ■

Acknowledgment

We would like to thank N. Jing and X. Liang for their help with energy modeling of eDRAM-based register files. This work was supported by the Ministry of Science, ICT & Future Planning (MSIP) under the "IT R&D program of MSIP/KEIT" (KI001810041244, Smart TV 2.0 Software Platform) and the "Basic Science Research Program" (NRF-2014R1A1A1005894), and by 2015 Hongik University Research Fund.

■ References

[1] NVIDIA Corp., "GK110/GK220 architecture whitepaper." [Online]. Available: https://www.nvidia.com

[2] J. Lim et al., "Power modeling for GPU architectures using McPAT," *ACM Trans. Design Autom. Electron. Syst.*, vol. 19, no. 3, Jun. 2014, Art. no. 26.

[3] N. Jing et al., "An energy-efficient and scalable eDRAM-based register file architecture for GPGPU," in *Proc. 40th Annu. Int. Symp. Comput. Architect.*, 2013, pp. 344–355.

[4] N. Jing, H. Liu, Y. Lu, and X. Liang, "Compiler assisted dynamic register file in GPGPU," in *Proc. Int. Symp. Low Power Electron. Design*, 2013, pp. 3–8.

[5] K. Cho, Y. Lee, Y. H. Oh, G.-C. Hwang, and J. W. Lee, "eDRAM-based tiered-reliability memory with applications to low-power frame buffers," in *Proc. Int. Symp. Low Power Electron. Design*, 2014, pp. 333–338.

[6] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to GPU codes," in *Proc. Innovative Parallel Comput.*, 2012, pp. 1–10.

[7] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. Int. Symp. Workload Characterization*, 2009, pp. 44–54.

[8] University of Illinois at Urbana-Champaign, "Parboil Benchmark Suite." [Online]. Available: http://impact.crhc.illinois.edu

[9] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *Proc. 39th Annu. Int. Symp. Comput. Architect.*, 2012, pp. 1–12.

[10] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. 16th Int. Conf. Architect. Support Programm. Lang. Oper. Syst.*, 2011, pp. 213–224.

[11] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proc. 17th Int. Conf. Architect. Support Programm. Lang. Oper. Syst.*, 2012, pp. 301–312.

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

Donghwan Jeong is currently working toward an MS at the Department of Semiconductor and Display Engineering, Sungkyunkwan University (SKKU), Suwon, Korea. His research interests include computer architecture, parallel programming, and memory systems. Jeong has a BS in semiconductor systems engineering from SKKU (2014).

Young H. Oh is currently working toward an MS/PhD in the Department of Electrical and Computer Engineering, Sungkyunkwan University (SKKU), Suwon, Korea. His research areas include specialized accelerator architectures, parallel programming, and GPU optimization for deep learning. Oh has a BS in electronics engineering from SKKU (2013).

Jae W. Lee is currently an Assistant Professor in the Department of Semiconductor Systems Engineering, Sungkyunkwan University (SKKU), Suwon, Korea. His research areas include computer architecture, compilers, parallel programming, and computer security. Lee has a PhD in computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA (2009). He is a Member of the IEEE.

Yongjun Park is currently an Assistant Professor in the School of Electronic and Electrical Engineering, Hongik University, Seoul, Korea. His research interests include compilers and computer architectures for various computer systems. Park has a PhD in electrical engineering from the University of Michigan, Ann Arbor, MI, USA (2013). He is a Member of the IEEE.

■ Direct questions and comments about this article to Jae W. Lee, College of Information and Communication Engineering, Sungkyunkwan University (SKKU), Suwon 440-746, Korea; jaewlee@skku.edu.