Power-performance Co-optimization of Throughput Core Architecture using Resistive Memory

Nilanjan Goswami, Bingyi Cao and Tao Li

Intelligent Design of Efficient Architectures Laboratory (IDEAL)
Department of Electrical and Computer Engineering, University of Florida
{nil, caobingyi}@ufl.edu, taoli@ece.ufl.edu

Abstract

Massively parallel computing on throughput computers such as GPUs requires myriad memory accesses to register files, on-chip scratchpad, caches, and off-chip DRAM. Unlike CPUs, these processors have a large register file and on-chip scratchpad memory, which consume a significant portion of compute core power (35%-45%). In this paper, we introduce novel throughput architecture by integrating resistive memory (Spin Transfer Torque RAM) inside the compute core, which reduces leakage significantly, but introduces write power overhead and longer write latencies in GPU shared memory and register file accesses. We enhance the compute core by introducing register file organization with differential memory update mechanism to remove update redundancy during write operations. Furthermore, using merged register-write-mechanism and write-back buffer, we coalesce multithreaded GPU register write accesses to save write energy. In addition, we introduce hybrid shared memory design using SRAM and STT-MRAM that provides significant leakage/dynamic power savings without affecting performance. On average, across 23 GPGPU/graphics workloads, our schemes save 46% dynamic power due to register access (83% leakage power saving) with negligible performance degradation. On average, hybrid shared memory provides 10% reduction in dynamic power with maximum 1.6× performance improvement for the current workloads at no additional area overhead.

1. Introduction

In recent years, GPUs have experienced tremendous growth as general purpose throughput processors with products from Nvidia and AMD [1, 2]. As throughput-computing devices, GPUs have multiple shader cores composed of thread-schedulers, ALUs, load/store units, large register file, scratchpad memory, caches etc. Unlike graphics computing, general purpose GPU computing (GPGPU) exposes shader cores as massively parallel compute cores. Using a high bandwidth, low latency on-chip interconnect, compute cores communicate with the on-chip cache and off-chip memory. GPGPUs achieve teraFLOPs peak performance by executing thousands of threads in parallel, while consuming large amounts of energy [3]. This trend of throughput computing has pushed the limits of GPU design to optimize its microarchitecture and programming paradigm to achieve higher performance per watt [3-6], while still lowering average power (around 130W [3]). We have verified these claims for the Nvidia GTX470 GPU using current sensors [7] for Nearest Neighbor (NN, 140W peak) and LU Decomposition (LU, 170W peak) in Figure 1.

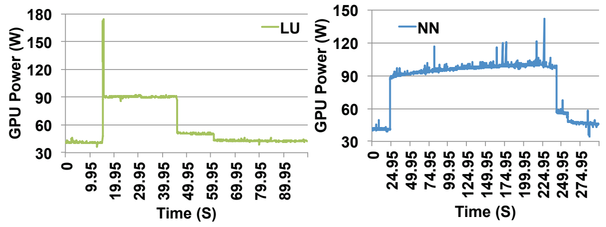

Figure 1. GPU power consumption in Watts

Compute cores have large on-chip memory and a register file built using CMOS-based SRAM memory. With growing transistor density in GPU dies, leakage power has begun to dominate the overall chip power for deep-sub-micron CMOS processes. To characterize the power behavior of GPUs, we have developed architecture level GPU power model in GPGPU-Sim [8] (see Section 5). The characterization reveals that on average across 23 GPGPU workloads, register file and shared memory dynamic power consumption is 44% and 7% of the total core power, respectively. Leakage power is 17% and 20% for register file and shared memory respectively. The leakage is even greater design impediment for sub-32nm technology nodes. Hence, to reduce overall compute core power consumption, it is imperative to re-architect various memory components of the compute core using emerging technologies.

The latest developments in magnetic junction transistor (MJT) based spin-transfer torque magnetic random access memory (STT-MRAM) demonstrate almost zero leakage power, better scalability, smaller foot-print, non-volatility, and radiation resistance [9, 10]. These improvements suggest that STT-MRAM may be a suitable replacement for on-chip SRAM based memory. Unlike SRAM, resistive memories rely on non-volatile, resistive information storage in a cell, and thus exhibit near-zero leakage in the data array [10]. Moreover, according to ITRS projections, STT-MRAM is expected to replace SRAM for on-chip memory designs [11]. Recently, Everspin Technologies [12] has officially announced commercialization of world's first STT-MRAM chip. In addition, IBM, Samsung and

Grandis Inc. are currently actively working on STT-RAM commercialization process. The STT-MRAM provides almost 4× more cell density and similar read latency compared to SRAM [13, 14]. STT-MRAM does not have write endurance problem (10^{15} writes [15]). Furthermore, STT-MRAM is capable of high performance operations and CMOS process compatible, which makes it suitable for wide range of applications [16]. However, STT-MRAM suffers from longer write latencies and higher write energies compared to SRAM [9]. To write a "0" or "1" into an STT-RAM cell, a strong current is necessary to reverse the magnetic direction of the storage node (Magnetic Tunnel Junctions, or MTJ). The latency and energy overhead associated with the write operations of these emerging memories has become a major obstacle in their widespread adoption [17]. In this work, we propose architectural enhancements in GPU compute cores to recuperate possible performance and energy losses involved due to STT-MRAM writes; hence, we achieve significant leakage power savings with negligible latency and energy due to writes.

As throughput-computing devices, GPUs execute multiple instances of identical or different sequence of instruction streams (GPGPU kernels) simultaneously to achieve higher overall performance, similar to graphics applications. Therefore, to preserve overall throughput of the graphics and GPGPU applications while lowering the overall power budget, we propose architectural enhancements in STT-MRAM based GPU on-chip memory design that hide the higher write latency of STT-MRAM based memory while decreasing overall write energy by using an arrayed register file design. The key idea is to reduce write accesses to the STT-MRAM based memory at the granularity of updated register array; explicitly unmodified register arrays are not written. Furthermore, using SRAM based small write-back buffers, we coalesce (wider update) multiple register writes from different threads in GPGPU thread-batches within a single register bank to reduce the register write energy. To address shared memory performance degradation due to slower STT-MRAM writes, we introduce a hybrid shared memory architecture based on SRAM and STT-MRAM. Because different applications exhibit temporal locality, spatial locality or a mixture of both in shared memory access, we propose SRAM configurable for either cache or scratchpad and dedicate STT-MRAM exclusively for scratchpad use. Heavy write intensive applications with inherent temporal locality access the STT-MRAM scratchpad using SRAM cache. For other applications, the shared STT-MRAM and SRAM scratchpad reduces write latency. We obtain this hybrid design without any additional overhead due to area saving in STT-MRAM design.

Our work makes the following contributions:

- A novel STT-MRAM design customized for GPUs to achieve lower write latency, lower energy, higher performance, and higher endurance at the cost of lower retention time (see Section 3).
- A STT-MRAM based GPU register file design that reduces leakage power and overall energy footprint of the compute cores. To recuperate the STT-MRAM write power overhead, we have designed a register file that is comprised of several arrays. Instead of updating the entire register during write operation, we update certain arrays within the register if any bit within the array is flipped.
- A SRAM/STT-MRAM based hybrid shared memory design to obtain leakage/dynamic power savings due to resistive memory, while keeping the performance intact.
- A method to reduce the overall energy consumption and leakage power by augmenting the shader core design with STT-MRAM based read-only on-chip caches.

The rest of this paper is organized as follows. Section 2 provides background and motivation of the work. Section 3 and 4 presents the proposed STT-MRAM based novel compute core design. Section 5 presents our experimental methodologies. In Section 6 we evaluate our design. Section 7 highlights previous research in this area. Section 8 concludes the paper.

2. Background and motivation

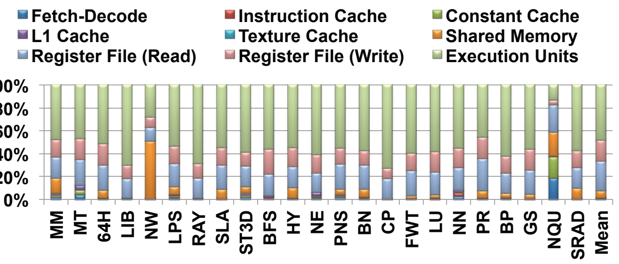

Figure 2. Dynamic power profile of compute cores

In single-instruction-multiple-data (SIMD) throughput processors, data computations in compute cores consume 35% to 45% (50W-85W across workloads) of total GPU power. This includes power consumption in integer/floating point units (ALU), special functional units (SFU), thread-scheduler, various on-chip caches, register file, and shared memory accesses [3]. In Figure 2, we show power consumption breakdown of the compute core components across several GPGPU and graphics workloads simulated using a customized McPAT [18] based GPU power model and GPGPU-Sim v3.0.1b [8] (see Section 4). Around 70%-90% of total compute core power is consumed by execution units (25%-45%), register file (20%-40%), fetch-decode unit (1%-10%), and shared memory (2%-40%) accesses. Since increasing compute core count almost linearly (assuming all cores are occupied) increases overall GPU power consumption, power optimization of intra-core components (such as register file, shared memory, on-chip caches etc.) is imperative to limit the energy consumption of the whole system. In this work, we primarily focus on energy consumption of the compute cores due to on-chip memory accesses' dynamic and leakage power. To reduce the leakage power for deep-submicron technology nodes, we propose resistive memory (STT-MRAM) based GPU compute core design. However, STT-MRAM based memory shows dynamic power overhead due to larger write energy requirement. In addition, longer write operation to these memory cells poses performance threat for GPGPU applications.

Intuitively, temporal locality of the register file access can be exploited using a small SRAM based cache to reduce write accesses to the STT-MRAM based large register file. To justify the feasibility of such design, we characterize the temporal locality of the GPU register write operations. Figure 3 reveals reusability of dynamically generated register updates during kernel execution. Across 23 GPGPU/graphics workloads, only 15 workloads have 20% or more 1-time register update. Using a small register file cache (RFC) [4] and dedicated memory update of one-time-register-writes, it is possible to reduce main register file write overhead. For rest of the workloads and unforeseen emerging throughput workloads, using RFC might be insufficient. Moreover, for deep-sub-micron technologies, leakage cannot be reduced by such methods. Hence, resistive memory (low or almost no leakage) on-chip storage solutions are imperative to address deep-sub-micron GPU core designs. We choose STT-MRAM based memory design due to significantly lower leakage power of these memory cells for sub-32nm technologies [9].

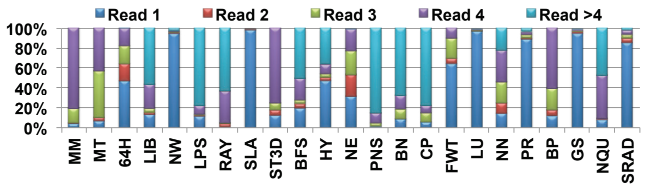

Figure 3. Reusability of registers write-back data

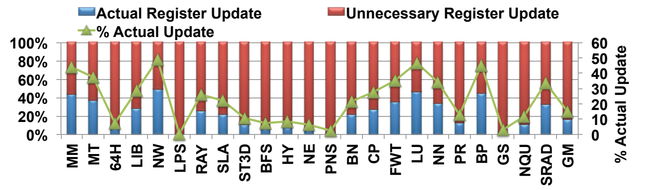

Figure 4. Percentage register file write (in bits)

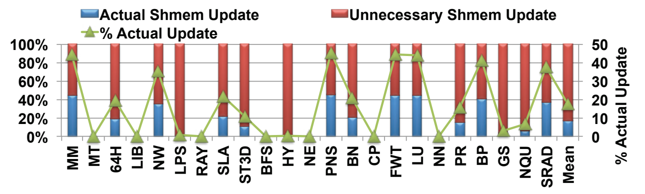

Figure 5. Shared memory write (in bits) profile

Therefore, we propose STT-MRAM based on-chip GPU memory design with architecturally optimized write access techniques. Due to larger energy consumption overhead, the write accesses need to be regulated to reduce the compute core power. Moreover, longer write latency also regulates the performance of the compute core. Intuitively, reduced memory cell updates are directly proportional to the write energy consumption. Hence, to understand the write update behavior of on-chip memories, we have characterized actual bit flip operations of register write for 23 GPGPU/graphics workloads. In Figures 4 (register file) and 5 (shared memory), we show that actual bit flips during write operation are significantly smaller compared to total write count multiplied by write access width. Surprisingly, for our workload set, only NW shows maximum 45% of bit flips of total writes. On average, only 20% of write operations flip the register cells. Similar trend is observable for shared memory in Figure 5. Write dominated workloads (PR, FWT, SRAD, GS, BN) show relatively small fraction of bit flips during shared memory writes (15%, 40%, 35%, 2%, 20%). On average, less than 10% bits are flipped during shared memory write. This insinuates architectural memory write enhancements to restrict unnecessary memory updates and reduce write energy.

3. Resistive memory and cell design

The STT-MRAM memory cell comprises of a MTJ and an access transistor (Figure 6). The MTJ device consists of two magnetic layers separated by a thin dielectric material. The dielectric acts as a tunnel for moving current between the magnetic layers. The access transistor is a CMOS, and the MTJ magnetic material is grown over the source and drain regions of the transistors. The MTJ device stores a "0" or "1" based on the direction of the free layer in contrast to the pinned layer. If the direction of both magnetic layers is same, the MTJ exhibits low resistance and represents "0" state. Contrasting magnetic directions represents "1". Therefore, to perform a write operation on MTJ, either a positive or negative voltage should be applied between the top and the bottom electrodes for writing a "0" or "1" respectively. A transistor and a MTJ cell are coupled through word-lines and bit-lines to form memory arrays. Each cell is read by driving the appropriate word-line (WL) that connects the relevant MTJ to the bit-line (BL) and source line (SL). When a small bias voltage (0.1V) is applied across the WL and BL, it senses the current passing through the MTJ using a current sensing amplifier connected to the BL.

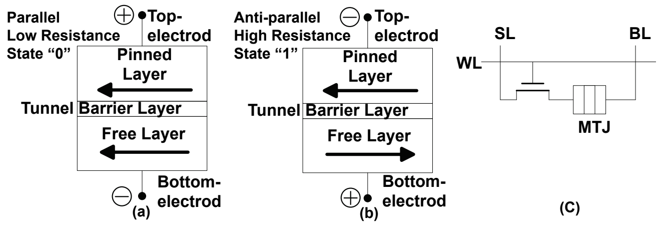

Figure 6. Block diagram of STT-RAM cell

Unlike SRAM, MTJ based resistive memory has high read speed, unlimited read and write endurance, and good compatibility with CMOS processes. Unlike 6 CMOS-based SRAM cell, the leaky CMOS based access transistor of STT-MRAM cell has no gate and sub-threshold leakage due to grounded SL, BL and WL lines. The MTJ device of STT-MRAM acts as a resistor only. Therefore, STT-MRAM consumes almost no leakage power compared to SRAM. However, STT-MRAM cell shows faster read access and slower write access capabilities compared to SRAM cell. The read speed is determined by three factors. Firstly, how fast the capacitive WL can be charged to turn on the access transistor. Secondly, how fast the BL can be raised to the required read voltage to sample the read-out current. Finally, how fast the sense amplifier reads. On the contrary, the write operation requires activating the access transistor, and applying comparatively higher voltage that can generate enough current to modify the spin of the free layer. MTJ performs such operation in three different modes: under thermal activation mode through the application of a long, low-amplitude current pulse (>10ns) or under a dynamic reversal regime with intermediate current pulses (3-10ns) or in a precessional switching regime with a short (<3ns), high-amplitude current pulse [19]. In a 1T-1MTJ cell with a fixed size MTJ, a trade-off exists between volatility and write-time. In our design, we sacrifice the non-volatility of STT-SRAM based register file and shared memory. Unlike [20], we have redesigned the MTJ (See Table 1) cell that has smaller free layer by using relaxed non-volatility mechanism [9]; smaller technology node (22nm) compared to [20] makes free layer even smaller. It provides much smaller cell size and 33% reduction in write energy. This results in reduced retention time, which is still long (0.1ms i.e. at 1.25GHz clock 125000 cycles) enough for the on-chip memory in GPU where data is updated much faster. However, special purpose long term registers such as program counters, stack pointers etc. are implemented using SRAM to address data remembrance issue.

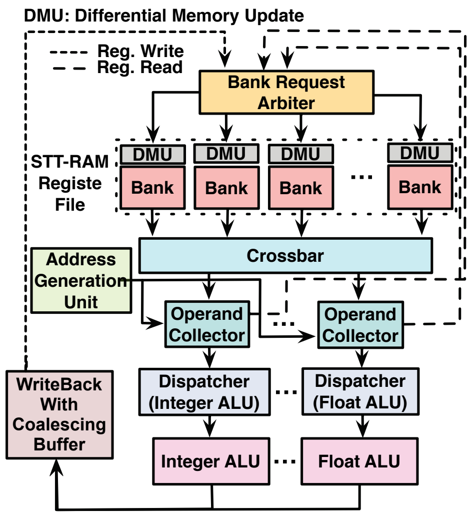

Figure 7. Differential Memory Update based GPU register unit architecture with STT-RAM

Table 1. Redesigned STT-MRAM and baseline SRAM cell parameters

Parameters	STT	SRAM
Cell Size	$9F^2$	$50F^2$
Switching Current	54μA	-
Switching Time	5ns	-
Write Energy	0.58pJ/bit	0.32pJ/bit
MTJ Resistance (R_{low}/R_{high})	1500/3000 Ω	-
Retention Time	0.1ms	-
Write Latency	1.4ns	0.77ns

4. Resistive memory based compute core

In this section, we introduce the STT-MRAM and SRAM based GPU compute core on-chip memory architecture that includes write-optimized (latency and energy) register file design for GPU, hybrid shared memory architecture, STT-MRAM based on-chip read only cache design.

4.1 Arrayed register file organization with differential memory update

In Figure 7, we have shown the register unit architecture for our throughput processor based on operand collector architecture [21]. Unlike traditional SRAM based register file, we design the entire register file memory array using STT-MRAM based memory cells. The design provides lower dynamic and leakage energy consumption compared to SRAM based main register file. In addition, the design also enhances read performance and reduces die footprint. However, STT-MRAM memory exhibits significantly longer write latency and consumes higher write energy. According to register write characterization in Section 2, a large percentage of register write does not result in memory cell modification. Intuitively, at register word granularity (width of the register write bus per register file bank), we can compute the actual memory cell modifications (0➔1 or 1➔0) by comparing the content of the register word and the register write back data generated by the compute core pipeline. By ignoring the unchanged memory cell, we obtain a register write mask at register word granularity. In each register bank interface, we propose to design a differential memory update unit (DMU), which will be responsible for calculating the mask at the register word granularity. In addition, by checking the set bits of the mask, the DMU will also provide per-bit word-line enable signal. Since the bank arbitration unit in operand collector always prioritizes the write operation over the read in each bank, for any incoming write request the DMU will read the current content of the register to generate the mask. Finally the mask will restrict large amount of current drawn by the unchanged bit-lines connected to the STT-MRAM cells. In the context of GPU, it will reduce a large percentage of register file access power without affecting the performance. Since the thread warps are scheduled in an interleaved fashion in GPU compute core pipeline, the latency of additional read operation during the write does not affect consecutive register read latencies and overall performance. Due to the fact that STT-MRAM read energy is significantly smaller compared to SRAM read, the overall energy consumption of register write is reduced due to masked memory update. We choose read-before-write memory update as opposed to early write termination (EWT) [22] for three reasons. Firstly, register reuse interval is long enough to prevent performance degradation due to additional read. Secondly, considering additional energy overhead (0.0457nJ/cell) of EWT circuit and unchanged cell (0.148pJ/cell) the read energy (0.013nJ/cell) is significantly lower (71%) [22]. Thirdly, in terms of area

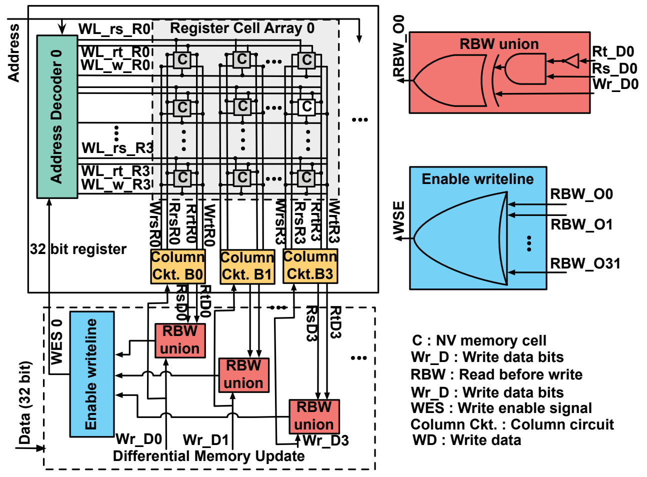

Figure 8. Microarchitecture of DMU (K=4) with multi-array register bank organization

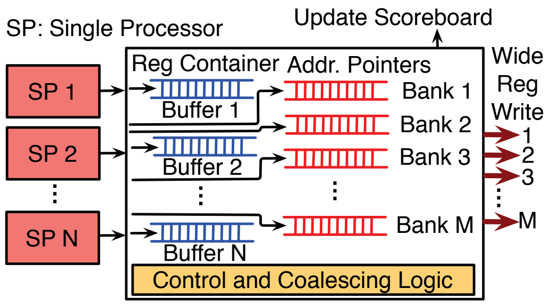

Figure 9. Coalescing register access using write-back buffer

and design complexity, the implementation overhead of EWT leaves read-before-write as a better choice.

Figure 8 shows the microarchitecture of the DMU. At the beginning, it treats each register write with a warp as read operation. So it enables the read-line of the register word using the address of the write request register. It compares the existing and incoming value of register word in read-before-write union (RBW) to generate a mask that acts as an enable signal. Finally, it uses write enable signal (WES) to determine whether to execute the write operation for the cell or not. However, per-bit write enable signal needs an additional address decoder; as a result it incurs large area overhead. To avoid area overhead and still perform masked register write, we propose novel arrayed register bank architecture. We propose to split an N bit register word into M arrays of K bits/array; where N = M×K. These arrays will be resident in a single bank, but will have separate decoder and write enable signals. Therefore, each K bits of the array share a single word-line enable signal. Philosophically, this reduces the granularity write operations. Register write characterization reveals that GPGPU kernels frequently update contiguous memory cells within a register. This opens up the opportunity of register update with granularity of multiple contiguous bits. Hence, instead of per-bit modification, we introduce per array modification in the bank. Single bit modification within the array will trigger whole K bit array write. We explore several values for determining the array width (K=1,2,4,8,16), in terms of area overhead and power saving in Section 6. Moreover, in the context of GPUs, arrayed register file memory layout opens up finer granular clock gating opportunities to further reduce dynamic energy consumption.

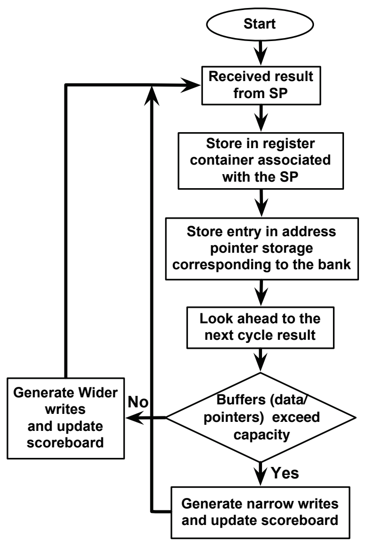
Start
Received result from SP
Store in register container associated with the SP
Store entry in address pointer storage corresponding to the bank
Look ahead to the next cycle result
Buffers (data/ pointers) exceed capacity
No
Yes
Generate Wider writes and update scoreboard
Generate narrow writes and update scoreboard

Figure 10. Register write-back coalescing flow

4.2 Coalesced register update

The register unit receives write requests from the write-back stage of the compute core pipeline. During register-write operation, every bank can write at most one register word, which is determined by the width of the per-bank register file write bus. Generally, each register word update requires entire register bank word-line to be activated. However, in the proposed arrayed architecture, total array activation count depends upon number of updated arrays which is less than or equal to the total array count that constitutes a whole row. Unwanted activation results in unnecessary power consumption. Intuitively, using wider access width can significantly reduce such power loss. With a wider write-port, greater number of bits per row can be written thus saving row activation energy.

In throughput architecture, thousands of threads work in smaller batches (warp). Within each warp, threads work in lock step fashion and generate several register write operations in quick succession. All the register writes within a single batch go to a single register bank in the traditional GPU. For each thread, the write requests are serviced at register word granularity. Hence, to service the entire warp, several cycles are wasted. However, using wider register write port, GPU warps can finish the pending register writes at faster rate. Naturally, it also opens up the performance improvement opportunity as well.

At any given time, several active warps are resident in a GPU. Often, individual warps access several registers from a single bank. Therefore, using per-bank write request buffering, we can coalesce multiple register writes across different warps that are writing to the same row of the register bank. To this end, we propose per-bank buffered write-back architecture to enable intra and inter-warp register write coalescing within each bank. We explore an optimum register write access width to determine the size of register write port while considering area, power and performance tradeoffs. Figure 9 shows buffered write-back stage architecture based on [23]. Unlike latency based buffering in [23], we propose per-SIMD lane (8-32 lanes/compute core) buffer and per-bank write address buffers. Figure 10 shows the steps involved in the coalescing process. The coalescing logic searches the per-bank address buffer to merge multiple register writes going to adjacent cells in a row of a register bank. Each bank with pending requests generates wider write request in each cycle by releasing the data from the register content buffer. Simultaneously, the scoreboard that manages read-after-write hazard is also updated to indicate multiple register release. We propose to implement SRAM based buffers due to smaller sizes of these buffers and faster performance requirement.

4.3 Hybrid shared memory architecture

Shared memory or software managed on-chip scratchpad memory in GPUs provide additional performance improvement opportunity for GPGPU applications. By exploiting application behavior, GPGPU developers manage repetitive data access using shared memory. In essence, it behaves like software-managed cache for GPGPU applications. The data access behavior of GPGPU applications exhibits four different patterns: temporal locality across warps, spatial locality across warps, mix of temporal/spatial locality across warps and, no locality. Applications with shared data locality among warps can further be optimized using shared memory caching with no additional programming overhead. Contrastingly, applications with no or low locality shared data access can be optimized using increased shared memory size. Moreover, on average, most of the GPGPU applications have multiple kernels (execution phases) that are temporally separated. For those, the shared memory access pattern within a single application changes during the overall execution period. Intuitively, to resolve these issues we need larger cached shared memory. However, this will introduce power consumption overhead in the existing SRAM based shared memory design.

To this end, we propose to implement configurable hybrid shared memory architecture that has SRAM and STT-MRAM based memory cells. Using STT-MRAM in shared memory design, we can reduce dynamic power, leakage power and area. We propose additional SRAM based shared memory area that can be configured using software level APIs to behave as cache or RAM. For temporal or spatial locality shared memory access, additional SRAM based cache reduces STT-MRAM access. Using compiler assistance, we forward last SRAM cache eviction data to the next level of memory avoiding STT-MRAM. Moreover, to avoid redundant STT-MRAM cell write operations, we use read-before-write based differential memory update (DMU) architecture. In essence, this architecture not only reduces STT-MRAM shared memory write access latency using SRAM cache, but also lowers energy consumption by using differential memory update. In addition, we further power gate the SRAM section of the shared memory to reduce power for applications with fewer shared memory update. In Section 6, we provide details about the area overhead and power/performance tradeoff of the design while varying the size of these memories. Note that existing GPGPU workloads do not optimize the kernels for cached shared memory access. Therefore, maximum potential of the cached architecture is unexplored. In brief, hybrid shared memory architecture provides the following benefits for GPGPU architectures: it reduces overall leakage power using STT-MRAM, addresses performance and energy overhead due to STT-MRAM write process using SRAM cache, and application level shared memory tuning offers more flexibility to the application developer to optimize performance and power of GPGPU applications that have phase change behavior across the kernels.

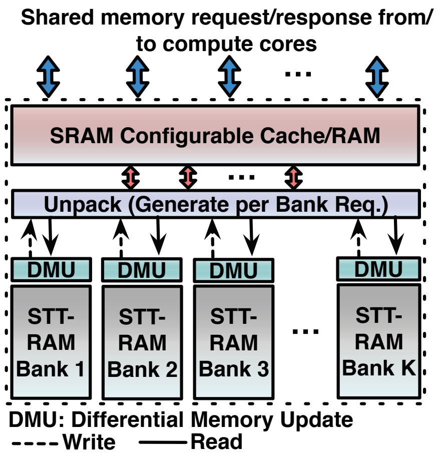
Shared memory request/response from/ to compute cores
...
SRAM Configurable Cache/RAM
...
Unpack (Generate per Bank Req.)
DMU
DMU
DMU
DMU
STT-RAM Bank 1
STT-RAM Bank 2
STT-RAM Bank 3
...
STT-RAM Bank K
DMU: Differential Memory Update
Write
Read

Figure 11. Hybrid shared memory architecture

4.3.1 Shared memory caching

Figure 11 shows the shared memory architecture with SRAM cache. In GPU, several threads of a warp access different banks of the shared memory. To keep shared memory performance intact, we should keep bank level access parallelism unchanged in our design. In the proposed architecture, every SRAM cache line compacts different threads of a warp that requests same tag in different banks. This reduces total cache line replacements per transaction. In the best-case scenario, this leads to 1 replacement per transaction. In the worst-case scenario, it leads to warp size count replacements per transaction. For 32 threads per warp, it will generate 32 transactions. Since warps execute in lock-step fashion, multiple replacements per transaction increase the warp level access latency of the transaction. In Section 6, we evaluate different design alternatives (direct mapped, set associative and fully associative) and properties (block count, block size) of hybrid design. The SRAM cache follows write-back policy. Intuitively, write-back policy avoids unnecessary STT-MRAM writes. Any write-miss during cache access writes the data to the SRAM cache but does not fetch the data from the STT-MRAM (no write-allocate policy). Since shared memory access is software managed, having a unified cache does not hamper the bank level parallelism much. During cache line replacement, data is received and extracted by the DMU of each bank. Finally, shared memory is updated only with the modified bits. However, read miss is served directly by the banks avoiding the DMU logic. Each cache line is equipped with an additional bit to indicate final update. After final update, cache line eviction sends the write-back to the next level of memory by avoiding STT-MRAM write energy and latency overhead.

4.4 Resistive read-only caches

Excluding register file and shared memory, GPU compute cores also contain several on-chip read-only memory for texture mapping and storing constant data. In brief, they are part of the fixed function logic in the GPU. In GPGPU applications, these memories are used as read-only caches to further expedite application performance. Fixed function logic brings data into these caches from host processor or other logic blocks of GPU pipeline. Due to repetitive read access of the cached data, STT-MRAM based memory significantly improves the read performance and reduces dynamic read power and leakage energy. Hence, energy and latency overhead of infrequent data load from the off-chip memory location is compensated by frequent read operations of these caches. Also, STT-MRAM reduces area overhead of the caches.

5. Experimental setup

Table 2 shows the architecture level GPU power simulator configuration that consists of GPGPU-Sim v3.0.1b [8] with GPU power model. The throughput architecture power model in power simulator is organized in a three level hierarchy of the programmable logic blocks available in GPU. At architectural level, the GPU is decomposed into major components such as compute cores, L2 cache, interconnect and memory controllers. Compute cores are further divided into SIMD cores, fetch/decode units, instructions issue units, on-chip caches, large register file, shared memory, and thread-scheduler. At circuit level, the architectural blocks are mapped to circuit structures like arrays and complex logic. Caches are modeled as memory arrays using CACTI [24] at the circuit level. Interconnect is composed of signal links and routers. Router is composed of flit buffers, arbiters and crossbars, which are modeled analytically. The memory controller models the design by Denali [25] and is composed of front-end-processing engine and physical interface. While front end is modeled using CAM and RAM structures, the latter two are modeled empirically. At the device level, the model uses data from ITRS roadmap [11] to calculate physical parameters of devices, such as capacitance, resistance etc. The power model uses relevant components from McPAT [18] and fits them into the compute core pipeline. Unlike McPAT, the GPU power simulator models multiple SIMD lane based throughput processor pipeline that has contrasting architecture compared to multicore processors. We validate our power model against the power values generated by actual power measurements using current sensors [7] attached to a GTX 470 GPU.

Table 2. GPGPU-Sim configuration

Parameters	Values
Compute Core Count	30 (10 cluster, 3 core/cluster)
Clock (Core/Icnt/MC)	1.25/0.65/0.8 GHz
Thread Batch Size	32
SIMD Pipe Width	32
Shared Memory	32KB
Shared Memory Banks	16
SHM Latency (Cycles)	20/20 (SRAM) 16/40 (STT)
Cache (L1/Cons/Tex)	16/8/4 (KB) per core
Threads / Core	1024
Memory Controller	8
Registers / Core	16K
Register File Banks	32
Register Latency (R/W)	2/2 (SRAM) 2/4 (STT) cycle
Interconnect Topology	Mesh
Channel BW	32B
Technology Node	22nm

We have heavily instrumented GPGPU-Sim v3.0.1b [8] to obtain register file, shared memory, on-chip cache update statistics. Moreover, we have implemented the latency model for all the on-chip memory models. Since STT-MRAM cell has different latencies for read and write operations, we have customized the simulator to have different latencies for register read, register write-back, shared memory load and shared memory store operations. In addition, we have implemented differential memory update mechanism within on-chip memory models.

We build a NGSPICE circuit model to obtain STT-MRAM cell electrical characteristics. We use 22nm CMOS process technology with a supply voltage of 0.9V. The physical MOSFET model is based on 22nm BSIM [29]. The circuit model has two parameters: write pulse duration (5ns [30]) and threshold current of MTJ devices. We assume the size of MTJs is 50nm×75nm [31] with a critical current density of 9×105A/cm^2 [31]. The derived threshold current is approximately 50uA, which is provided by transistors fabricated using existing CMOS technologies [32]. The MTJ is simulated using a constant resistor in the STT-MRAM cell model, while multiple time-varying resistors are used during switching simulation. To model the latency and power of register file and cache, we have implemented STT-MRAM cell model in CACTI 6.5 [24] using NGSPICE simulation results as input parameters.

Table 3. Simulated workloads [26-28] (AI: ALU Inst. per Memory Inst., SHM: Shared memory)

Workload (Abbr.)	AI	SHM?
Matrix Multiplication (MM)	42.4	Y
Matrix Transpose (MT)	195.6	N
64 Bin Histogram (64H)	22.8	Y
LIBOR (LIB)	353.6	N
Needleman Wunsch (NW)	80.1	Y
Laplace Solver (LPS)	118.9	Y
Raytrace (RAY)	335.0	N
Parallel Prefix Sum (SLA)	1.9	Y
Stencil 3D (ST3D)	822.32	Y
Breadth First Search (BFS)	99.7	N
Hybrid Sort (HY)	32.1	Y
Nearest Neighbor (NE)	196.8	N
Petri Net Simulation (PNS)	411.5	Y
Binomial Options (BN)	39.4	Y
Columbic Potential (CP)	471.8	N
Fast Walsh Transform (FWT)	289.1	Y
LU Decomposition (LU)	3.49	Y
Neural Network (NN)	71.5	N
Parallel Reduction (PR)	1.0	Y
Back Propagation (BP)	167.6	Y
Gaussian Elimination (GS)	5.7	Y
N-Queen Solver (NQU)	209.3	Y
Speckle Reducing Anisotropic Diffusion (SRAD)	14.0	Y

Using 23 real world GPGPU/graphics workloads, we have evaluated the effectiveness of the proposed compute core architectural enhancements to reduce energy consumption. Table 3 lists the workloads which are programmed using Nvidia CUDA APIs and possess good mix of intense arithmetic computation (increased register file access) and large local data sharing behavior (increased shared memory access) [33, 34].

6. Results and analysis

The following subsections evaluate the STT-MRAM based on-chip memory design benefits.

6.1 Register architecture evaluation

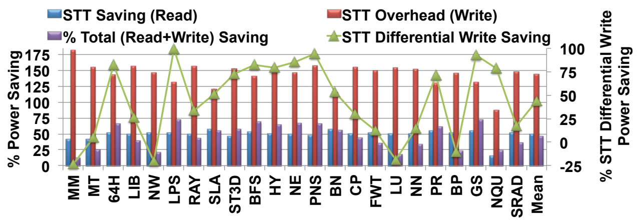

Figure 12. Power saving in DMU based (K=32) register file with STT-MRAM based memory cell

Figure 12 shows the power savings of the proposed STT-MRAM based register file architecture. The register read power of the STT based memory is reduced by 49% on average. Wire transfer power overhead of the read operations is largely mitigated by the copious amount of transistor switching power. Benchmarks such as NQU, MT, ST3D, and MM show lower power savings due to fewer read operations compared to the rest.

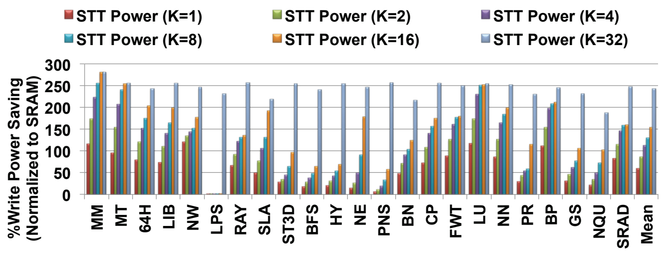

Figure 13. Comparison of power saving in the register file with variable array sizes (K=1,2,4,8,16)

On average, non-differential update based register file with STT-MRAM cells show 1.43x increase in power consumption due to write overhead. Heavy write dominated workloads such as MM (1.81x), MT (1.55x), 64H (1.55x), CP (1.55x), PNS (1.56x) and RAY (1.56x) suffer the most. Using per-bit (K=32) write tracking, the differential memory update saves write energy by 44% (SRAM baseline) on average. GPGPU workloads with fewer bit updates save write power by 82% (64H), 82% (BFS), 85% (NE), 93% (PNS) and 92% (GS). On the contrary, workloads such as MM, NW, LU, BP suffer power overhead due to large amount of bit modification during register write. Overall, the read and write operations for these workloads save power by 11%(MM), 21%(NW), 17%(LU) and 22%(BP). Across 23 workloads, combined read and write operations save 46% dynamic power on average. LPS (72%), BFS (68%), GS (72%), NE (66%), PNS (65%) and HY (63%) are the highest power saving workloads. Due to area and power overhead, we have re-architected the register file with multiple arrays per bank. Figure 13 shows gradual increase in power saving as we decrease the array width of arrayed register file organization. On average, across 23 workloads, array width of 2-bits, 4-bits and 8-bits provides 12%, 29% and 53% power savings respectively. The area saving for additional wiring and address decoder in these configurations are 50%, 25% and 12% compared to the 32array/bank configuration (baseline). Due to contiguous register bit update pattern in MM, MT, LU, NN, BP, LIB and 64H, configurations such as K=4/8/16 consume power close to K=32 configuration. Interestingly, LPS shows almost no write power saving for K=4/8/16 configurations; interleaved bit update pattern of this workload hides the benefit of proposed scheme. Based on the power saving and area overhead, K=8/4 configurations provide good tradeoff. On average, STT-MRAM based register file reduces the leakage power by 32% across 23 GPGPU workloads in our design and 46% area compared to SRAM. Irrespective of lower read and higher write latencies, performance degradation of our design is negligible (see Figure 14) across most workloads. Only BFS, NE and FWT show 4%, 3% and 3% performance degradation respectively due to longer write latency of consecutive dependent instruction scheduling from the same warp. Read dominated 64H workload benefits from the faster read performance of the STT-MRAM cell and achieves 4% performance gain.

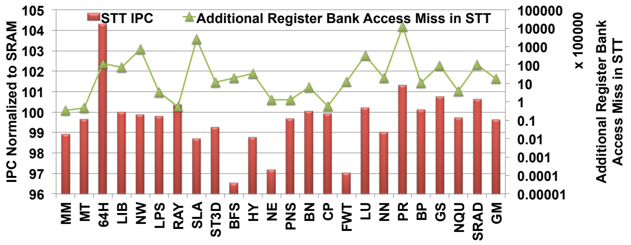

Figure 14. Performance impact of STT-MRAM based register file with DMU compared to pure SRAM (baseline) implementation

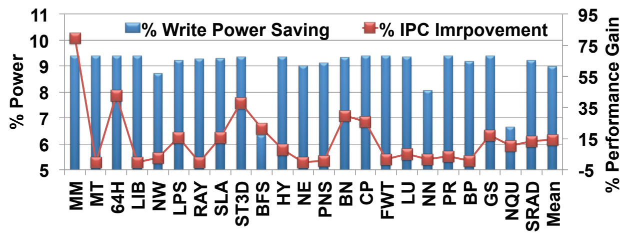

Figure 15. Power saving / performance impact of coalesced register update (baseline: pure SRAM)

Figure 15 shows the power and performance improvement of register access coalescing in STT-MRAM based register file. On average, coalesced writes save 9% of write power. Benchmarks such as LIB (9.4%), PR (9.4%), RAY (9.3%), and 64H (9.4%) provide maximum power benefit due to contiguous active threads in a warp that access contiguous registers in a row. Interleaved access pattern in BFS (6.3%), NQU (6.7%) and NN (8 %) restricts the power saving. In Figure 15, MM (79%), 64H (42%), ST3D (38%), BN (29%) and CP (25%) experience good performance improvement using the register write coalescing. Coalesced access with wider write port reduces total number of register writes generated by each warp. For example, N threads/warp generates N write requests per warp that reach same STT register bank. Regular coalescing of M threads/coalesce with all active threads per warp reduces total number of write request to N/M. This phenomenon significantly improves the overall register write latency of the workload and overall performance. However, in reality, thread divergence and memory miss divergence within warp reduces the active thread count that leads to non-contiguous active threads in warps and increases number of total inactive threads across all the warps. Therefore, MM, 64H, LIB, PR and RAY experience lower power saving and performance benefit. Note that, register write coalescing scheme will provide comparatively higher power benefit in SRAM based memory array due to additional CMOS transistors present in each memory cell compared to STT. For 32-entry buffer, area overhead of coalesced write-back architecture is only 0.1mm^2 per shader core.

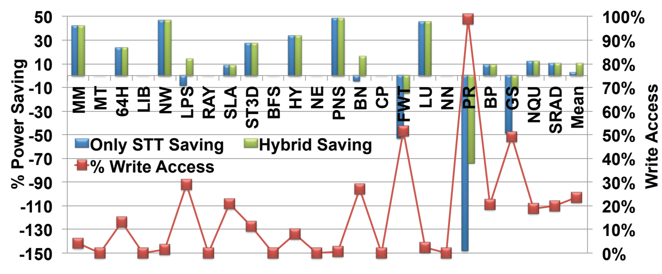

Figure 16. Power saving in hybrid shared memory

6.2 Hybrid shared memory evaluation

Figure 16 shows the power savings due to hybrid-shared memory. Across 23 GPGPU workloads, differential memory update based STT-MRAM saves only 3% power compared to SRAM based design. However, STT-MRAM only shared memory saves 41%, 46%, 33%, 48% and 45% memory access power for MM, NW, BFS, PNS and LU. On the contrary, due to large percentage of memory write, shared memory does not save additional power for several GPGPU workloads. Benchmarks such as LPS, BN, FWT, PR and GS have 29%, 27%, 51%, 98% and 49% write access of total shared memory access respectively. Therefore, un-optimized STT-MRAM shared memory suffers significant power overhead in these workloads. LPS, BN, FWT, PR and GS suffer 8%, 4%, 53%, 140% and 48% power overhead respectively. On the other hand,

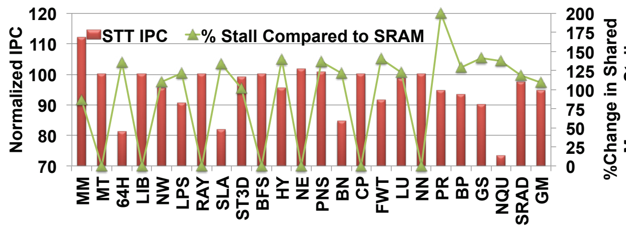

Figure 17. STT only shared memory performance (normalized to SRAM)

average performance degradation for STT only shared memory is 5% (see Figure 17). However, consecutive memory writes dominant workloads suffer significant performance loss. For example, 64H, BN, NQU, SLA, LPS and FWT experience 18%, 16%, 27%, 18%, 9% and 8% performance degradation respectively. The correlation between performance loss and power savings justifies the co-optimization approach of hybrid-shared memory. Cache is used to revive the performance and SRAM based scratchpad memory extension reduces write energy overhead. Hybrid shared memory improves the power overhead of LPS (13%) and BN (15%). Using the proposed scheme, FWT, PR and GS reduces power overhead by 3.8×, 2× and 4.4× respectively. Excessive write accesses with significant amount of bit modifications during write operations makes these workloads suffer power loss.

Figure 18 compares the performance improvement of hybrid-shared memory across several SRAM based cache

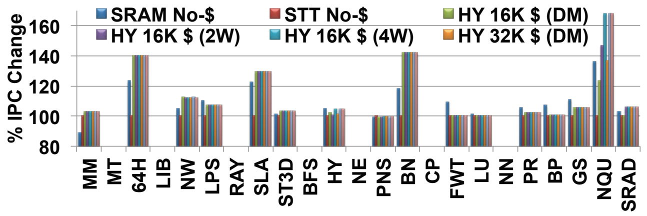

Figure 18. Performance improvement using hybrid shared memory

configurations. 16KB and 32KB caches with 2-way and 4-way set associativity recover the performance losses for BN, NQU, SLA, and 64H. Hybrid shared memory improves the performance of BN, NQU, SLA, and 64H by 40%, 42%, 30% and 40% respectively due to SRAM based cache (16KB 2-way set associative) that backs up STT-MRAM based scratchpad memory. The workloads add 14%, 15%, 12% and 12% additional performance improvement after recovering the IPC loss due to STT-MRAM only based shared memory design. FWT does not achieve performance improvement and LPS shows 2% performance degradation compared to SRAM based design. Note that LPS and FWT do not improve using the hybrid shared memory design. Special feature of these workloads is simultaneous power and performance degradation due to STT-MRAM write penalty. Close investigation reveals that these workloads reuse recently written data at a faster rate and overall write access density is also higher. Moreover, during kernel execution, temporal/spatial locality (requires cache) and intense write access with no locality (requires RAM) of shared memory changes dynamically over time. In future, dynamic switching between cache and RAM access modes of shared memory might resolve this issue.

A 32KB STT-MRAM with 16KB cache and 64KB STT-MRAM with 16KB cache saves leakage power by 82% and 106% respectively. A 32KB and 64KB STT-MRAM shared memory saves the area by 0.011mm^2 and 0.021mm^2 respectively. 16KB and 32KB 2-way set associative SRAM cache adds additional area of 0.015mm^2 and 0.029mm^2. Therefore, 32KB STT-MRAM with 16KB SRAM cache or 64KB SRAM with 32KB cache have negligible area overhead.

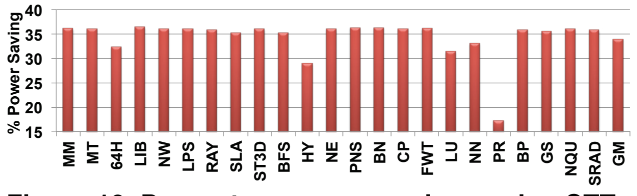

Figure 19. Percentage power savings using STT-RAM based read-only memory in GPU

6.3 Evaluation of STT based on-chip caches

GPU on-chip caches significantly improve read-only data access performance. Using STT-MRAM based memory architecture saves significant leakage power in addition to improving the performance. Figure 19 shows that read-only caches save power by 34% on average compared to SRAM. Excluding HY, PR and LU, most of the GPGPU workloads show around 35% power savings due to constant read access rate. However, PR, LU and HY have limited read-only data accesses, which restricts their large power improvements. On average, leakage power savings due to resistive memory is 31%.

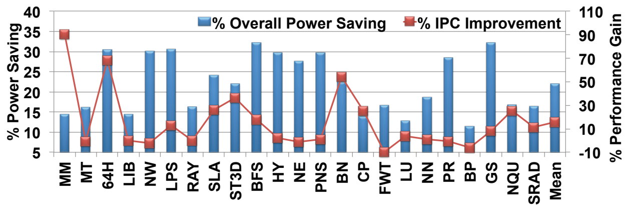

Figure 20. Overall power and performance improvement using STT-MRAM architecture

6.4 Overall power and performance impact

Figure 20 shows overall power saving and performance improvement of STT-MRAM based computer core architecture using arrayed register file organization, hybrid shared memory, and register write coalescing. On average, across 23 GPGPU workloads the architecture saves 22% power and improves performance by 16%. 64H (31% power/68% IPC), BN (22% power/54% IPC), NQU (16% power, 23% IPC) and ST3D (21% power/36% IPC) receive most benefit using the architecture. All of these workloads possess a large number of redundant register update, temporally localized shared memory accesses, and significant percentage of contiguously active threads in a warp. On the contrary, BFS (32% power/18% IPC), HY (29% power/2% IPC), PNS (29% power/2% IPC), NW (30% power/ -2% IPC) and GS (32% power/8% IPC) have ample amount of redundant register file writes, localized shared memory accesses, and thread level inactivity in warp with frequent reuse of register writes. These reduce power consumption without improving the performance significantly. Only FWT (16% power/-9% IPC) and BP (11% power/-5% IPC) have moderate power with performance degradation. FWT suffers maximum performance degradation due to shared memory write latency overhead introduced by STT. Shared memory write latency also affects performance of BP. Using dynamic switching between RAM and cache behavior during kernel execution might resolve the performance degradation issue of FWT and BP.

7. Related work

Zhao *et al.* [35] have proposed non-volatile FPGA circuit based on the STT-MRAM. They further enhance the power efficiency and the startup time of the design in [36]. Guo *et al.* [10] proposed the idea of applying MTJ device to most of the on-chip storage to improve power efficiency and maintain performance within 5% decrease. [9] proposed the idea of relaxing the non-volatility of STT-MRAM cells to reduce the high dynamic energy and slow write latencies. In [22], Ping *et al.* proposed early write termination to avoid writing unnecessary bit flips by reading the current content of the register for caches. Since a heavy current pulse is applied at the end of the write operation to alter the bit, [22] proposed to wait until the end of the write pulse to determine the memory bit flip status. In early write termination, considerable amount of energy is wasted during the wait in the write process. Though it is faster, the scheme consumes unnecessary power during the wait. In GPUs, since the warps are further interleaved in time, register reuse by the same warp provides enough time to perform energy efficient but slower read-after-write operations instead of faster but power consuming early write termination. In [30], Sun et al. proposed buffered L2 cache to reduce the write access to the L2 cache. In addition, they have proposed SRAM-MRAM based hybrid 3D L2 cache architecture. In [37], Wu et al. proposed inter/intra cache level hierarchy design based on disparate memory technologies. Unlike [30], we propose differential memory update based shared memory organization using software level configurable SRAM based cache/RAM and resistive memory based RAM. For write intensive GPGPU shared memory access applications, we use SRAM cached resistive shared memory to reduce write latency and power. On the contrary, GPGPU applications with balanced read-write shared memory accesses still can benefit the dynamic/leakage power saving of resistive memory without compromising the performance. In essence, to achieve write power reduction and faster write performance, we exploit GPGPU/graphics workload behaviors to enhance the throughput core architecture by differential memory update in a novel arrayed resistive memory organization.

Franklin *et al.* [38] analyzed the reusability of updated register values and reuse interval for parallel processors. Previously, ample amount of work discussed viability of register file caches in terms of performance for CPU [39-44]. Unlike register file latency, we aim to design energy efficient throughput processor register files for deep-sub-micron technology nodes (very low leakage). Recently, Gebhart [4] proposed register file caches with hierarchical thread scheduling for energy efficient design. However, the work does not address the leakage issue of deep-sub-micron technology nodes for any compute core memories. We also incorporate architectural enhancements to reduce write power and latency, which are overhead of the STT-MRAM technology. Satyamoorthy [45] implemented SRAM write-buffer to address write latency for STT-MRAM based shared memory performance overhead and reported 17% energy with 50% area saving. Instead, we have enhanced shared memory architecture using differential bit updater (write energy saving) and small cache per shared memory bank (recuperate write latency overhead).

In the context of phase change memory, Lee et al. [32] proposed PCM cell read mechanism to improve the endurance of the off-chip PCM based memory by avoiding write operations. At L2 cache level, they check the dirty bit at word granularity to check for the update status of the memory word. In [46] Zhou et al. proposed PCM memory with cell level write redundancy removal scheme to improve endurance. Instead of word level or cell level granularity, our techniques employ STT-MRAM based register file and shared memory update at 4-16 bits granularity. This reduces the update status check (1 check/cell) overhead compared to [46]. Moreover, using new arrayed (2/4/8/16 bits) GPU register file organizations, we reduce per-bit enable-signal-decoder area overhead with only 16/8/4/2 decoder signals.

8. Conclusion

Larger on-chip memory components (register file, shared memory, various caches) in throughput processors become design impediment for deep-sub-micron technology nodes due to excessive leakage power and overall energy footprint. Emerging resistive memory technologies such as STT-MRAM provide feasible on-chip memory design alternative to address power problems. However, such memory technologies have longer write latency and higher energy consumption. To this end, we propose on-chip memory organization for throughput architectures using resistive memory that implements differential memory update based STT-MRAM register file and hybrid shared memory. The proposed architecture reduces STT-MRAM write power for several on-chip memories with 83% saving in leakage power. Resistive memory based memory organization saves 46% of the register file dynamic power with less than 1% performance overhead. We further optimize the register file access power by coalescing several consecutive register writes with wider write ports and small SRAM based write-buffers. This provides additional 8% power savings. By incorporating SRAM and STT-MRAM based hybrid shared memory organization with interchangeable cache and RAM behavior of the memory, we can compensate the performance loss due to STT-MRAM write latency and improve the power efficiency. On average, hybrid shared memory provides 10% power savings and 1.6× performance improvement without any area overhead.

9. Acknowledgements

This work is supported in part by NSF grants 1117261, 0937869, 0916384, 0845721(CAREER), 0834288, 0811611, 0720476, and by Microsoft Research Trustworthy Computing, Safe and Scalable Multi-core Computing Awards. Authors acknowledge UF HPC center for providing computational resources.

10. References

[1] NVIDIA Corporation, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", Nvidia White Paper, 2009.
[2] N. Brookwood, "AMD Fusion™ Family of APUs: Enabling a Superior, Immersive PC Experience", AMD White Paper, 2010.
[3] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model", ISCA, 2010.

[4] M. Gebhart, D. Johnson, D. Tarjan, S. Keckler, W. Dally, E. Lindholm, and K. Skadron, “Energy-efficient Mechanisms for Managing Thread Context in Throughput Processors”, ISCA, 2011.
[5] A. Maashri, G. Sun, X. Dong, V. Narayanan and Y. Xie, “3D GPU Architecture Using Cache Stacking: Performance, Cost, Power and Thermal Analysis”, ICCD, 2009.
[6] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, “Statistical Power Modeling of GPU Kernels using Performance Counters”, ICGC, 2010.
[7] U.R.D. International, Inc. DC Current Sensor Model: HCS-20-10-AP.
[8] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, “Analyzing CUDA Workloads Using a Detailed GPU Simulator”, ISPASS, 2009.
[9] C. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. Stan, “Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches”, HPCA, 2011.
[10] X. Guo, E. Ipek, and T. Soyata, “Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM based Computing”, *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 371–382, 2010.
[11] “International Technology Roadmap for Semiconductors,” ITRS, 2009.
[12] E. Technologies, “Everspin Debuts First Spin-Torque MRAM for High Performance Storage Systems ”, 2012. http://www.everspin.com/PDF/ST-MRAM_Press_Release.pdf.
[13] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, “Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement”, DAC, 2008.
[14] S. Lai and T. Lowrey, “OUM - A 180 nm Nonvolatile Memory Cell Element Technology for Stand Alone and Embedded Applications”, IEDM, 2001.
[15] F. Tabrizi, “The Future of Scalable STT-RAM as a Universal Embedded Memory”, *EETimes Design*, 2007. http://www.eetimes.com/design/embedded/4026000/The-future-of-scalable-STT-RAM-as-a-universal-embedded-memory.
[16] C. Smullen, A. Nigam, S. Gurumurthi, and M.R. Stan, “The STeTSiMS STT-RAM Simulation and Modeling System”, ICCAD, 2011.
[17] A. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. Das, “Architecting On-Chip Interconnects for Stacked 3D STT-RAM Caches in CMPs”, ISCA, 2011.
[18] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures”, MICRO, 2009.
[19] R. Buhrman, “Spin Torque MRAM - Challenges and Prospects”, DRC, 2009.
[20] Y. Chen, X. Wang, H. Li, H. Xi, Y. Xie, and W. Zhu, “Design Margin Exploration of Spin-Transfer Torque RAM (STT-RAM) in Scaled Technologies”, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 12, pp. 1724–1734, 2010.
[21] S. Liu, J. Lindholm, M. Siu, B. Coon, and S. Oberman, “*Operand Collector Architecture*”, U.S. Patent 7834881, 2010.
[22] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “Energy Reduction for STT-RAM Using Early Write Termination”, ICCAD, 2009.
[23] S. Ranganathan, “High Performance Architecture for a Write-back Stage”, U.S. Patent 7519794, 2009.
[24] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi, “CACTI 6.5”, *hpl.hp.com*. http://www.hpl.hp.com/research/cacti.
[25] Denali Software, Inc., “Using Configurable Memory Controller Design IP with Encounter RTL Complier”, Cadence CDNLive, 2007.
[26] NVIDIA Corporation, “GPU Computing SDK”, *developer.nvidia.com*, 2010. http://developer.nvidia.com/gpu-computing-sdk.
[27] Impact Research Group, “Parboil Benchmark Suite”, *impact.crhc.illinois.edu*, 2010. http://impact.crhc.illinois.edu/parboil.php.
[28] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron, “Rodinia: A Benchmark Suite for Heterogeneous Computing”, IISWC, 2009.
[29] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, “New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation”, CICC, 2000.
[30] G. Sun, X. Dong, Y, Xie, J. Li, and Y. Chen, “A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs”, HPCA, 2009.
[31] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, “A Novel Nonvolatile Memory with Spin Torque Transfer Magnetization Switching: Spin-Ram”, IEDM, 2005.
[32] S. Lee, H. Lee, S. Kim, S. Lee, and H. Shin, “A Novel Macro-Model for Spin-Transfer-Torque based Magnetic-Tunnel-Junction Elements”, *Solid-State Electronics*, vol. 54, no. 4, pp. 497–503, 2010.
[33] NVIDIA Corporation, “NVIDIA CUDA Programming Guide”.
[34] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable Parallel Programming with CUDA”, *ACM Queue*, vol. 6, no. 2, pp. 40–53, 2008.
[35] W. Zhao, E. Belhaire, Q. Mistral, E. Nicolle, T. Devolder, and C. Chappert, “Integration of Spin-RAM Technology in FPGA Circuits”, ICSICT, 2006.
[36] W. Zhao, E. Belhaire, C. Chappert, F. Jacquet, and P. Mazoyer, “New Non-volatile Logic based on Spin-MTJ”, *Physica Status Solidi (a)*, vol. 205, no. 6, pp. 1373–1377, 2008.
[37] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, “Hybrid Cache Architecture with Disparate Memory Technologies”, ISCA, 2009.
[38] M. Franklin and G.S. Sohi, “Register Traffic Analysis for Streamlining Inter-Operation Communication in Fine-Grain Parallel Processors”, *SIGMICRO Newsletter*, vol. 23, no. 1, pp. 236–245, 1992.
[39] R. Balasubramonian, S. Dwarkadas, and D. Albonesi, “Reducing the Complexity of the Register File in Dynamic Superscalar Processors”, MICRO, 2001.
[40] E. Borch, S. Manne, J. Emer, and E. Tune, “Loose Loops Sink Chips”, HPCA, 2002.
[41] T. Jones, M. O'Boyle, J. Abella, A. Gonzalez, and O. Ergin, “Energy-Efficient Register Caching with Compiler Assistance”, *ACM Trans. Archit. Code Optim.*, vol. 6, no. 4, pp. 1–23, 2009.
[42] J. Cruz, A. Gonzalez, M. Valero, and N. Topham, “Multiple-Banked Register File Architectures”, *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 316–325, 2000.
[43] P. Nuth and W. Dally, “The Named-State Register File: Implementation and Performance”, HPCA, 1995.
[44] H. Zeng and K. Ghose, “Register File Caching for Energy Efficiency”, ISLPED, 2006.
[45] P. Satyamoorthy, “*STT-RAM for Shared Memory in GPUs”*, M.S. Thesis, School of Engineering and Applied Science, University of Virginia, 2011.
[46] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “A Durable and Energy Efficient Main Memory using Phase Change Memory Technology”, ISCA, 2009.