**Abstract**

Graphics Processing Units (GPUs) maintain a large register file to increase thread block occupancy, hence to improve the thread level parallelism (TLP). However, register files in the GPU dissipate a significant portion of the total leakage power. Leakage power of the register file can be reduced by putting the registers into low power (SLEEP or OFF) state. However, one challenge in doing so is the lack of precise register access information for each instruction at run-time.

This paper proposes **GREnEER** (GPU Register file ENergy Reducer): a tool for minimizing leakage energy of the register file of GPUs. GREEnEER employs a compile-time analysis to estimate the run-time register access information. The result of the analysis is used to determine the power state of the registers (ON, SLEEP, or OFF) after each instruction. We propose a new power optimized assembly instruction format that allows GREEnEER to encode the power state of the registers with its instruction. Further, GREEnEER transforms a given input assembly language to a power optimized assembly. The optimized assembly, along with a run-time optimization, results in significant power reduction.

We implemented **GREEnEER** in GPGPU-Sim simulator and used GPUWatch framework to measure the leakage power of register file. We evaluated the effectiveness of GREEnEER on 21 kernels from CUDASDK, GPGPU-SIM, Parboil, and Rodinia benchmarks suites. We observe an average reduction of register leakage energy by 69.2% and maximum reduction of 88.41% with a negligible performance overhead (0.05% slowdown on average).

**Keywords** Register File, Power, Energy, and Performance

1 Introduction

Graphics Processing Unit (GPU) achieves high throughput by utilizing thread level parallelism (TLP). Typically, GPUs maintain a large register file in each streaming multiprocessor (SM) to improve the TLP. GPUs allow a large number of resident threads [1] in each SM, and the resident threads can store their thread context in the register file, which facilitates faster context switching of the threads. The threads that are launched in each SM are grouped into sets of 32 threads (called warps), and they execute the instructions in a SIMD manner. To keep improving the TLP of the GPUs, GPU architects increase the maximum number of resident threads and register file sizes in every generation. For instance, NVIDIA Fermi GF100 has 128KB register file and allows up to 1536 resident threads, while NVIDIA Kepler GK110 has 256KB register file and allows maximum 2048 resident threads [1].

Earlier studies [2, 3] show that register files in GPUs consume a significant amount of power. With the technology advances, the leakage power component has increased and has become an important consideration for the manufacturing process [4, 5]. Moreover, registers in a GPU continue to dissipate leakage power throughout the entire execution of its warp even when they are not accessed by the warp.

1.1 Motivation

To understand the severity of leakage power dissipation by register file, consider Figure 1 which shows the access patterns of some registers of warp 0 during the execution of MUM application [6]. We use the access patterns of the registers of a single warp as a representative since all the warps of a kernel typically show similar behavior during execution [7]. We make the following observations:

- Register 10 is accessed very infrequently—it is accessed for only 3 cycles out of 15000 cycles shown in the figure. In fact, it is accessed for only 7 cycles during the complete execution (life time) of the warp (29614 cycles).
- Register 1 is the most frequently accessed register during the warp execution. However, it is accessed for only 330 cycles (∼1.11%) during the life time of the warp.

This shows that registers are accessed for a very short duration during the warp life time. However, they continue to dissipate leakage power for the entire life time of the warp. Figure 2 shows that the behaviour is not specific to MUM, but is seen across a wide range of applications. The figure shows the percentage of simulation cycles spent in register accesses (averaged over all the registers in all the warps) for several applications. We observe that registers on an average spend < 2% of the simulation cycles during the warp execution while leaking power during the entire execution.

One solution [8] to reduce the leakage power of the registers is by putting the registers into drowsy or SLEEP state immediately after the registers of an instruction are accessed.

---

1Drowsy [8, 9] and SLEEP [10] states refer to the same low power data preserving states. In this paper, we use the term SLEEP.
However, this can have run-time overhead whenever there are frequent wake up signals to the sleeping register. Consider Figure 1 again:

- Putting register 10 to SLEEP state immediately after its accesses saves significant power because there are gaps of several thousands of cycles between consecutive accesses.
- In contrast, register 1 is accessed very frequently. If it is put to SLEEP after every access, it will have a high overhead of wake up signals.
- The access pattern of register 7 changes during the warp execution. It is accessed frequently for some duration (for example, between cycles 10500–11250), and not accessed frequently for other duration (between cycles 3000–7500). To optimize energy as well as run-time, the register needs to be kept ON whenever it is frequently accessed, and put to SLEEP otherwise.
- The last access to register 8 is at cycle 1602. The register can be turned OFF after its last access to save more power.

To summarize, the knowledge of registers’ access patterns is required to improve energy efficiency without impacting the run-time adversely. Our proposed tool, GReENER, statically estimates the run-time usage pattern of the registers to reduce GPU register file leakage power.

### 1.2 Contributions

GReENER uses a compile-time analysis to determine the power state of the registers (OFF, SLEEP, or ON) for each instruction by estimating the register usage information. Further, it transforms an assembly input language by encoding the power state information at each instruction to make it energy efficient. The static analysis makes safe approximations while computing power state of the registers, therefore, the choice of the state can be suboptimal at run-time. Hence, to improve the accuracy and energy efficiency, it provides a run-time optimization that dynamically corrects the power state of registers of each instruction.

We make the following contributions in this paper:

1. We introduce a new instruction format that supports the power states for the instruction registers (Section 3.2).
2. We propose a compile-time analysis that determines the power state of the registers at each program point and transforms an input assembly language into a power optimized assembly language (Section 3.1 and 3.2).
3. We give a run-time optimization to reduce the penalty of suboptimal (but safe) choices made by static analysis (Section 3.3).
4. We implemented the proposed compile-time and run-time optimizations using GPGPU-Sim simulator [11]. We integrated GPUWatch [2] with CACT-P [12] version to enable power saving mechanism (Section 4).
5. We evaluated our implementation on 21 kernels from CUDA [13], GPGPU-SIM [6], Parboil [14], and Rodinia [15] benchmarks suites. We are able to reduce register leakage energy by an average of 69.2% and maximum of 88.41% (Section 5).

In the paper, Section 2 briefs the background required for GReENER, while the tool itself is described in Section 3. Section 4 and 5 give the experimental evaluation. Section 6 describes related work, and Section 7 concludes the paper.

### 2 Background

GPUs consist of a set of streaming multiprocessors (SMs). Each SM contains a large number of execution units such as ALUs, SPs, SFUs, and Load/Store units. GPUs achieve high throughput because they can hide long memory execution latencies with massive thread level parallelism. Each SM has a large register file, which allows the resident threads to maintain their contexts, and hence can have faster context switching. To reduce the access latency, the register file is divided into multiple banks. The registers from different banks can be accessed in parallel. A bank conflict occurs whenever multiple registers need to be accessed from the same bank, and these need to be accessed in serial. In GPGPU-Sim simulator [11] the requests for instruction registers are stored in a collector unit. When all the operands of the instruction are ready, it can proceed to the execution stage.

NVIDIA provides a programming language CUDA [16] to parallelize applications on GPU. The portion of the code which is to be parallelized is specified using a special function called kernel. A kernel is invoked with the number of thread blocks and the number of threads in each thread block.
as parameters. A program written in CUDA can be compiled using \texttt{nvcc} compiler. The compiler translates the program into an intermediate representation (PTX), which is finally translated to an executable code. NVIDIA provides tools such as \texttt{cuobjdump} to disassemble the executable into SASS assembly language. GPGPU-Sim converts SASS code to PTXPlus code for simulation.

\texttt{GPUWatch} [2] framework uses the simulation statistics of GPGPU-Sim to measure the power of each component in the GPUs. The framework is built on McPAT [10], which internally uses \texttt{CACTI} [17]. McPAT models the register files as memory arrays to measure the register power. \texttt{CACTI} divides memory arrays into set of banks, which are finally divided into subarrays (collection of memory cells).

\texttt{GREENER} optimizes the PTXPlus code to make it energy efficient by reducing the leakage power of the register files. Our experiments use \texttt{GPUWatch} framework to measure the leakage power.

3 \texttt{GREENER}: Reducing Register File Energy

To understand the working of \texttt{GREENER}, let $W$ denotes the minimum number of program instructions that are required to offset the wake-up penalty incurred when a register state is switched from OFF or SLEEP state to ON state. Consider a program that accesses some register $R$ in a statement $S$ during execution. The future accesses of $R$ in this execution govern its power state. The following scenarios exist:

1. The next access (either read or write) to $R$ is by an instruction $S'$ and there are no more than $W$ instructions between $S$ and $S'$. In this case, since the two accesses to $R$ are very close, it should be kept ON to avoid any wake-up penalty associated with SLEEP or OFF state.

2. The next access to $R$ is a read access by an instruction $S'$ and there are more than $W$ instructions between $S$ and $S'$. In this case, since the value stored in $R$ is used by $S'$, we can not switch $R$ to OFF state as it will cause the loss of its value. However, we can put $R$ in SLEEP state.

3. The next access to $R$ is a write access by an instruction $S'$ and there are more than $W$ instructions between $S$ and $S'$. In this case, since the value stored in $R$ is being overwritten by $S'$, we can put $R$ in OFF state.

4. There is no further access to $R$ in the program. In this case also, register $R$ can be safely turned OFF.

We now describe the compiler analysis used by \texttt{GREENER} to capture these scenarios.

3.1 Compiler Analysis

To compute power state of registers at each instruction, we perform compiler analysis at the basic block level, where each basic block contains single instruction. Determining the power state of each register requires knowing the life time of registers as well as the distance between the consecutive accesses to the registers. We use the following notations.

- $\text{isLive}, R$: True if the register $R$ is live at program point $\pi$.
- $\text{SUCC}, R$: The set of successors of the block $B$.
- $\text{INCB}, R$: The maximum value of $\text{DistINS}, R$ over the successors $S$ of $B$. A register $R$ can potentially be put into SLEEP or OFF state at a program point $\pi$.
- $\text{Power}, R$: The power state of the register $R$ at program point $\pi$. The liveness information of each register, $\text{isLive}, R$, can be computed using traditional liveness analysis [18] and is ignored for brevity. The data flow equations to compute the $\text{DistINB}, R$ and $\text{DistOUTB}, R$ are as follows:

\[
\text{DistINB}, R = \begin{cases} 
1, & \text{if } B \text{ accesses } R \\
\text{INCDistOUTB}, R, & \text{otherwise}
\end{cases}
\]

\[
\text{INCD}_x = \begin{cases} 
\infty, & \text{if } x \text{ is a write access} \\
x + 1, & \text{otherwise}
\end{cases}
\]

\[
\text{DistOUTB}, R = \begin{cases} 
\infty, & \text{if } B \text{ is Exit block} \\
\max_{S \in \text{SUCC}} \text{DistINS}, R, & \text{otherwise}
\end{cases}
\]

Note that $\text{INCD}_x$ is a saturating increment operator. Since our analysis aims to reduce the power consumption, we compute $\text{DistOUTB}, R$ as the maximum value of $\text{DistINS}, R$ over the successors $S$ of $B$. A register $R$ can potentially be put into SLEEP or OFF state at a program point $\pi$ if it is not accessed within the distance window $W$ on some path:

\[
\text{SleepOff}, R = \text{Dist}, R = \infty
\]

The power state of each register at each program point can be computed according to Table 1.

3.2 Encoding Power States

The power state ($\text{Power}, R$) of a register can be one of the three states: OFF, SLEEP, or ON. Thus, it requires two bits to represent $\text{Power}, R$ of one register. Since the power state can change after every instruction at run-time, we need to encode the $\text{Power}, R$ of the operand registers of an instruction in the instruction itself.

PTXPlus instructions [11] can support up to 4 source and 4 destination registers. Encoding $\text{Power}, R$ of all the registers will require 16 bits. We observed that in our benchmarks,
most instructions use only up to 2 source registers and 1 destination register. Therefore, to reduce the number of bits required to encode Power_State in each instruction, we encode information only for 2 source registers and 1 destination register\(^2\). For instructions having more registers, Power_State of the remaining registers is assumed to be SLEEP to enable power saving. The modified instructions have the format:

\[
<\text{Opcode}> <\text{Options}> <\text{Operand}\_\text{List}> <\text{Power}\_\text{State}\_\text{List}>
\]

Note that Power_State encoded for a register \(R\) in an instruction in block \(B\) is given by PowerOUT\(_B\), \(R\).

**Example 3.1.** Figure 3(a) shows a snippet of power optimized PTXPlus code, which is generated for SP benchmark using a threshold value 7. Explicit branch addresses have been replaced by block labels for ease of understanding. The bold text in the instructions indicates the power states inserted by \texttt{GREENER}. The control flow graph (CFG) corresponding to the snippet is shown in Figure 3(b).

The instruction at Line-1 uses 2 source registers (r8, r0) and 2 destination registers (p2, o127). As discussed, our analysis inserts the power states only for 2 source registers and 1 destination register. In this case, the power states ON, SLEEP, ON correspond to the registers p2, r8, and r0 respectively. The power state of o127 register (the fourth register in the instruction) is set to SLEEP state after accessing the register.

For register r0 of the instruction, the next access to the register occurs at Line-3 (at distance 2, less than the threshold value 7). Hence, the compiler inserts the power state as ON. Register p2 is also kept in ON state for a similar reason. For register r8 of the same instruction, the next access occurs along two paths. One of the paths has a use at a distance of 8 (along B5 at Line-13, >7), and the other has a definition after B9 (not shown in the figure). \texttt{GREENER} keeps the register in SLEEP state since there is a path along which the next access happens after a distance >7.

\(^2\)NVIDIA does not disclose the machine code format of the instructions. However, the project “asfermi” \cite{asfermi} gives us a hope that there are enough unused bits in the instructions to encode six bits for power states.
if any instruction from the same warp has been decoded that accesses a register whose power state is being changed to SLEEP or OFF (Section 3.4). If so, then the register power is kept ON. This avoids the wake up latencies for instructions that access the same register within a short duration, thereby avoiding the performance penalty.

Example 3.3. Figure 4(b) shows a possible execution sequence of a program whose CFG is shown in Figure 4(a). The instruction $I_{B0}$ (the instruction in block B0) writes to register r0. After writing the register value in write back stage (WB), the register needs to be put into SLEEP state. Assume that the program takes the path along B10 and decodes the instruction $I_{B11}$ before the write back stage of $I_{B0}$. Our run-time optimization detects the future access to r0 by $I_{B11}$, and keeps the register in ON state instead of putting it into SLEEP state to avoid additional wake up latencies. On the other hand, if the program takes the path along B0, then the instruction present in the B9 would appear much later in the pipeline (after WB stage of $I_{B0}$). The register r0 will be set to SLEEP state.

4. The registers in SLEEP or OFF state are woken up by sending a wake up signal to the register file (Label (4)).

5. The read operands phase (Label (5)) is modified (a) to set the power state of source registers after they have been read and (b) to release the source registers of the instruction which were reserved by the scoreboard unit.

6. The write back stage (Label (6)) includes the logic to set the power state of the destination registers after the registers are written.

7. The run-time optimization is implemented by adding a lookup table (Label (7)) to keep track of the registers accessed by an instruction. For an instruction having program counter $PC$ and warp id $Wid$, the lookup table is indexed by $Wid$. When an instruction is decoded, the decode unit inserts the instruction’s operand registers into the lookup table. When a warp ($Wid$) needs to set the power state of a register ($R$) of an instruction ($PC$) to SLEEP or OFF, it searches the lookup table for another instruction (a different PC) with the same $Wid$ and accessing $R$. If a match is found, then the power state of $R$ is kept ON, otherwise, it is changed. After an instruction completes its writeback stage, the corresponding entry is removed from the lookup table.

Each entry for a warp in the lookup table stores instruction’s $PC$, and its register numbers. The number of entries required for each warp is determined by the pipeline depth, which can be large. However, in practice, the number of entries required per each warp is less, and experimentally we found that the average number of entries per warp is less than 2. If an SM allows maximum $W$ resident warps, stores $w$ entries per each warp, supports $r$ operand registers for each instruction, and allows maximum $R$ registers per each thread, then the size of look up table (in bits) is $W \ast w \ast \text{sizeof} PC + \log_2 R \ast r$.

### 3.4 Hardware Support

Figure 5 shows the modified pipeline of GPU Architecture that supports our proposed ideas, with the modified components shaded and labeled. The changes are described below:

1. To support the new instruction format (Section 3.2), we modify the decode unit to extract the power states of the registers from the instruction (Label (1) in Figure 5).

2. The scoreboard unit (Label (2)) is modified to track RAR (Read After Read) and WAR (Write after Read) dependencies in addition to RAW (Read After Write) and WAW (Write after Write) dependencies. This is done by adding instruction’s source registers in the scoreboard table. It is because an instruction can change the power state of a register to SLEEP or OFF after reading the registers. Hence, the subsequent instructions that read/write the same register need to wait until the power state is modified.

3. A warp is considered ready for issuing its current instruction only when all its operand registers are in ON state. A comparison circuit (Label (3)) is added to check if the register operands are in ON state.

4. The run-time optimization is implemented by adding a lookup table (Label (7)) to keep track of the registers accessed by an instruction. For an instruction having program counter $PC$ and warp id $Wid$, the lookup table is indexed by $Wid$. When an instruction is decoded, the decode unit inserts the instruction’s operand registers into the lookup table. When a warp ($Wid$) needs to set the power state of a register ($R$) of an instruction ($PC$) to SLEEP or OFF, it searches the lookup table for another instruction (a different PC) with the same $Wid$ and accessing $R$. If a match is found, then the power state of $R$ is kept ON, otherwise, it is changed. After an instruction completes its writeback stage, the corresponding entry is removed from the lookup table.

Each entry for a warp in the lookup table stores instruction’s $PC$, and its register numbers. The number of entries required for each warp is determined by the pipeline depth, which can be large. However, in practice, the number of entries required per each warp is less, and experimentally we found that the average number of entries per warp is less than 2. If an SM allows maximum $W$ resident warps, stores $w$ entries per each warp, supports $r$ operand registers for each instruction, and allows maximum $R$ registers per each thread, then the size of look up table (in bits) is $W \ast w \ast \text{sizeof} PC + \log_2 R \ast r$.

### 4 Evaluation Methodology

We implemented the proposed hardware changes and compiler optimizations in GPGPU-Sim V3.x [11]. The modified instruction format is implemented by extending the PTX-Parser provided by GPGPU-Sim. The GPGPU-Sim configuration used for the experiments is shown in Table 2. We used GPUWatch [2] to measure the power consumption of register file.

### Table 2. GPGPU-Sim Configuration

| Resource                  | Configuration |
|---------------------------|---------------|
| Architecture              | NVIDIA Tesla K20x |
| Number of SMs             | 14            |
| Shader Core Clock         | 732 MHz       |
| Technology Node           | 32nm          |
| Register File Size per SM | 256KB         |
| Number of Register Banks  | 32            |
| Max Number of TBs per SM  | 16            |
| Max Number of Threads per SM | 2048         |
| Warp Scheduling           | LRR           |
| Number of Schedulers per SM | 4             |
We compiled all the applications using CUDA-4.0. Baseline Warp registers into OFF, SLEEP, and ON states, we set the Reg for the approach that optimizes the baseline approach by boil [14], and Rodinia [15]. Table 3 shows the list of applications and kernel that is simulated for each application.

| No. | Benchmark | Application | Notation | Kernel |
|-----|-----------|-------------|----------|--------|
| 1   | RODINIA   | backprop   | BP       | bpnn_adjustweights_cuda |
| 2   | RODINIA   | bfs        | BFS1     | Kernel |
| 3   | RODINIA   | bfs        | BFS2     | Kernel2|
| 4   | CUDA-SDK  | Blacksholes| BS       | BlackScholesGPU |
| 5   | RODINIA   | javaMD     | LMD      | kernel_gpu_cuda |
| 6   | GPGPU-SIM | LJB        | LJB      | Pathcalc_Portfolio_KernelGPU |
| 7   | GPGPU-SIM | LPS        | LPS      | GPU_laplac3d |
| 8   | CUDA-SDK  | MonteCarlo | MC1      | inverseCNDKernel |
| 9   | CUDA-SDK  | MonteCarlo | MC2      | MonteCarloOneBlockPerOption |
| 10  | PARBOIL   | mri-q      | MR1      | ComputePhiMag_GPU |
| 11  | PARBOIL   | mri-q      | MR2      | ComputeQ_GPU |
| 12  | GPGPU-SIM | MUM        | MUM      | mummergpuKernel |
| 13  | GPGPU-SIM | NN         | NN1      | executeFirstLayer |
| 14  | GPGPU-SIM | NN         | NN2      | executeSecondLayer |
| 15  | GPGPU-SIM | NN         | NN3      | executeThirdLayer |
| 16  | GPGPU-SIM | NN         | NN4      | executeFourthLayer |
| 17  | RODINIA   | pathfinder | PF       | dynproc_kernel |
| 18  | CUDA-SDK  | scalarProd | SP       | scalarProdGPU |
| 19  | PARBOIL   | sgemm      | SGEMM    | mysgemmNT |
| 20  | PARBOIL   | spmv       | SPMV     | spmv_jds |
| 21  | CUDA-SDK  | vectorAdd  | VA       | VecAdd |

Note that GPuWattch internally uses CACTI [17] to measure the power dissipation that does not support leakage power saving mechanism. Therefore, we modified GPuWattch to use CACTI-P [12] that provides power gating technique, which can minimize the leakage power by setting the SRAM cells into low power (SLEEP or OFF) state. It uses minimum data retention voltage so that SRAM cells can enter into SLEEP state without losing their data. We chose SRAM_vccmin to be the default value (provided by CACTI-P depending on the technology node, 32nm for this case). To put SRAM cells in OFF state (no data retention), we configured SRAM_vccmin to 0 V. In GPUs, the registers are allocated to each warp in a private manner. To independently turn the power state of warp registers into OFF, SLEEP, and ON states, we set the granularity of the subarray size to 1 warp register (a set of 32 registers). We chose the threshold distance to be 7 based on experiments described elsewhere [20]. We used the latency to wake up a register from SLEEP to ON state to be 1 cycle, and the latency to wake up a register from OFF to ON state be twice (i.e., 2 cycles), except for Section 5.6 where we consider other values for the wake up latencies.

We evaluated GreENER on several applications from the benchmark suites CUDA-SDK [13], GPGPU-SIM [6], Parboil [14], and Rodinia [15]. Table 3 shows the list of applications and kernel that is simulated for each application. We compiled all the applications using CUDA-4.0. We measured the effectiveness of our approach using the following metrics: (1) Power, (2) Energy, (3) Simulation Cycles.

5 Experimental Results

We use Baseline to denote the default approach that does not use any leakage power saving mechanisms. We denote Sleep-Reg for the approach that optimizes the baseline approach by (1) turning OFF the unallocated registers and (2) turning the allocated registers into SLEEP state immediately after the registers are accessed as described in [8].

5.1 Comparing Register Leakage Power

Figure 6 shows the effectiveness of GreENER and Sleep-Reg by measuring the reduction in leakage power with respect to Baseline. From the figure, we observe that GreENER shows an average (Geometric Mean denoted as G.Mean) reduction of leakage power by 69.21% when compared to the Baseline. It shows the GreENER is effective in turning the instruction registers into lower power state, such as SLEEP or OFF state depending on the behavior of the registers. The Baseline does not provide any mechanism to save the leakage power, as a result, the registers of a warp continue to consume leakage power throughout the warp execution. Figure 6 also shows that Sleep-Reg approach reduces the register leakage power by 60.13% when compared to Baseline, however, GreENER is more power efficient than Sleep-Reg. It is because Sleep-Reg approach reduces the leakage power by turning the instruction registers into SLEEP state immediately after the instruction operands are accessed, without considering the access pattern of the registers. If a register needs an immediate access, then keeping the register into SLEEP instead of ON state requires additional latency cycles to wake up the register, and during these additional cycles, the registers consume power. Further, GreENER saves more leakage power compared to Sleep-Reg by turning the registers into OFF state when there is no future use of the register, whereas Sleep-Reg turns the register into only SLEEP state irrespective of its further usage.

5.2 Performance Overhead Using Simulation Cycles

Figure 7 shows the performance overheads of GreENER and Sleep-Reg approaches in terms of the number of simulation cycles with respect to Baseline. On an average, the applications show a negligible performance overhead of 0.05%
with respect to Baseline. A slowdown is expected because Greener turns the registers into SLEEP or OFF states to enable power savings, and these registers are turned back to ON state (woken up) when they need to be accessed. This wake up process takes few additional latency cycles which leads to increase in the number of simulation cycles. Interestingly, some applications (MC2, MR1, NN1, NN2, NN3, SP, and VA) show improvement in their performance. This occurs due to the change in the issuing order of the instructions. The warps that require their registers to be woken up can not be issued in its current cycle, instead other resident warps that are ready can be issued. This change in the issue order leads to change in the memory access patterns, which in turns changes L1 and L2 cache misses etc. In case of MC2, NN1, NN2, and NN3 applications, we observe an improvement in the performance due to less number of L1 cache misses with Greener when compared to Baseline. Though SP and VA applications have same number L1 and L2 cache misses with Greener and Baseline approach, Greener shows less number of pipeline stall cycles when compared to Baseline. Similarly, MR1 shows same number of L1 and L2 cache misses, but it has less number of scoreboard stall cycles with Greener.

Figure 7 also shows that Sleep-Reg has an average performance degradation of 1.48% when compared to the Baseline approach. This degradation is more when compared to Greener because Sleep-Reg turns all the instruction registers into SLEEP state after the instruction operands are accessed, irrespective of their usage pattern. If a register in SLEEP state is accessed in near future, it needs to be turned on, this incurs additional wake up latencies with Sleep-Reg. Whereas, our approach minimizes these additional wake up latency cycles by retaining such registers in the ON state. However, some applications (MR2, ST) perform better with Sleep-Reg because it shows less number of scoreboard and idle cycles than that of Greener. Also, Sleep-Reg performs better with NN1 since it has less number of stall cycles when compared to Greener.

5.3 Comparing Register Leakage Energy

Figure 8 compares the total energy savings of Greener and Sleep-Reg w.r.t. Baseline. The results show that Greener achieves an average reduction of register leakage energy by 69.2% and 23.86% when compared to Baseline and Sleep-Reg respectively. From Figures 6 and 7, we see that Greener shows more leakage power saving, also has negligible performance overhead with respect to the Baseline, hence we achieve a significant reduction in leakage energy. Also, the applications that exhibit more power savings and improve their performance with Greener, further show more leakage energy savings. Similarly, the applications that show leakage power savings but has more performance overhead will reduce their leakage energy savings accordingly when compared to Baseline and Sleep-Reg approaches.

5.4 Effectiveness of Optimizations

We show the effectiveness of the proposed optimizations in Figure 9. From the figure, we analyze that the compiler optimization (discussed in Section 3.1, and denoted as Comp-OPT) saves more energy (average 69.04%) when compared to Sleep-Reg (59.54%). This shows that turning the registers into low power states (SLEEP or OFF state) with the knowledge of register access pattern is more effective than turning the registers into SLEEP state after accessing them.

The run-time optimization (discussed in Section 3.3) is evaluated by combining it with Sleep-Reg and Comp-OPT, and we denote them as Run-time-OPT and Greener respectively in the figure. From the results, we observe that, for most of the applications, Run-time-OPT and Greener approaches show minor improvements when compared to Sleep-Reg and Comp-OPT respectively. This is because the run-time optimization helps only in correcting power state of a register by turning to ON state when it detects the future access to the register at run-time. However, if the register is not found to be accessed in the near future at runtime, it does not modify and retains the power state as directed by the Sleep-Reg or Comp-OPT. For some applications (MC1, MC2, MR2, MUM, PF, and SGEMM), Run-time-OPT and Greener are less efficient when compared to Sleep-Reg and Comp-OPT respectively. It occurs when a register that is determined to be accessed in the near future does not get accessed due to reasons such as scheduling order, scoreboard stalls, or the unavailability of the corresponding execution unit. In those cases, keeping the register into low power states (SLEEP or OFF) can save more energy instead of keeping it in ON state. Note that the effectiveness of run-time optimization depends on the application behavior at the branch divergence points.

5.5 Leakage Power with Different Register File Sizes

Figure 10 shows the effect of register file size on leakage power for Baseline, Sleep-Reg, and Greener approaches. The register file sizes used are 128KB, 256KB, and 512KB. In the figure, Baseline-128KB denotes the Baseline approach that is evaluated with 128KB register file size. The other approaches are tagged in a similar way.
From the figure, we can see that for all the three approaches, leakage power increases with the increase in the register file size. This is because each memory cell in the register file consumes some amount of leakage power, and with the increase in the number of registers, leakage power increases. However, for Sleep-Reg and GREENER, this increase in the leakage power is less when compared to Baseline since the register files in Baseline consume leakage power irrespective of the register access. Whereas, registers can be put to lower power states with Sleep-Reg and GREENER. The results also indicate that GREENER is effective when compared to Sleep-Reg and Baseline even with the increase in register file size.

Interestingly, GREENER with 512KB register file configuration consumes less amount of leakage power than that of Baseline with 256KB configuration. Also, the leakage power of GREENER for 512KB configuration is comparable to that of Baseline-128KB. This shows that GREENER with twice the register file size compared to Baseline, not only shows less amount of leakage power consumption, but also can improve the amount of thread level parallelism.

5.6 Effect of Wake up Latency

Figure 11 compares performance overhead of GREENER and Sleep-Reg with the Baseline for different values of wake up latencies. In the figure, GREENER-WL-X (X ∈ {2, 3, 4}) denotes the GREENER approach, which considers the wake up latency to change a register state from SLEEP to ON to be X cycles. Whereas, when a register state needs to be changed from OFF to ON, it considers the latency to be 2X cycles. We use the similar notation for Sleep-Reg as well.

In the figure GREENER and Sleep-Reg show performance degradation with the increase in the wake up latency. GREENER-WL-2, GREENER-WL-3, and GREENER-WL-4 on an average degrades by 1.85%, 2.04%, and 2.58% respectively. The increase in the overhead is expected because applications spend additional simulation cycles for changing register’s state from OFF or SLEEP state to ON state. Hence, with the increase in the wake up latency, these additional simulation cycles will increase. Interestingly, some applications (MC2, NN2, SP, and VA) show performance improvement with the increase in the wake up latency. This is because, as discussed in Section 5.2, with the addition of wake up latency, the warps in the SM can get issued in different order, which can change the number of L1-cache misses, L2-cache misses, and stall cycles etc. In case of MC2, we find that the number L1 cache misses decrease with increase in the wake up latencies. Similarly, for NN2 we observe more number of L1 misses with GREENER when used with wake latency 2 cycles than that of 3 cycles, hence GREENER performs better with wake up latency 3 cycles. Also, for NN2, GREENER performs better than Baseline for all wake up latencies due to a decrease in the L1 misses when compared to Baseline. In case of SP, GREENER-WL-1 has more stall cycles when compared to GREENER-WL-2. Further, for most of the applications GREENER performs better than Sleep-Reg with various wake up latencies.
5.7 Leakage Energy Savings with Routing

So far we discussed the energy efficiency of registers in a register file, however, GPUs also consume energy for routing of data and address through the register file. While modeling the register file, McPAT uses H-Tree distribution network to route data and address [10]. The H-Tree network spends a constant amount of leakage power, and various organizations can be exploited to reduce this power and to meet routing requirement [17]. Our work focuses only on reducing the leakage power of memory cells of the register file by analyzing the register access patterns, and reducing the routing power is not in this scope. However, we show the effectiveness of GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} by including the constant routing energy as shown in Figure 13. From the results, we observe that GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} reduces the leakage energy up to 19.37%, which is more than that of Sleep-\textsubscript{OPT} (13.65%). However, the energy savings when including the routing energy are reduced when compared to that of results in Figure 8 because GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} does not provide any mechanism to optimize the routing power, hence the routing power remains unaffected.

5.8 Leakage Energy with Various Technologies

Figure 14 shows the effectiveness of GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} for various technology parameters (45nm, 32nm, and 22nm). The results show a significant reduction in leakage energy with GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} for all the applications even for various technology nodes. Further, it reduces the energy when compared to Sleep-Reg. With transition of technology from 45nm to 32nm, we observe an increase in the leakage energy for the Baseline approach, but GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} shows an increase in the leakage energy savings even with the transition. To model 22nm technology node, McPAT uses double gated technology to reduce the amount of leakage power, even with the advances in technology, GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} shows a reduction in leakage power when compared to Baseline. To summarize, architectural techniques help in reducing the leakage power of a register file, in addition, the knowledge of register access patterns and compiler optimizations further help in reducing the leakage power and energy.

We also compare the energy savings by varying the wake up latencies as shown in Figure 12. The results indicate that even with varying the wake up latency, the applications show significant reduction in the leakage energy when compared to Baseline. Also, the applications show more energy savings with GR\textsubscript{E\textsubscript{E}N\textsubscript{E}R} when compared to Sleep-Reg for all wake up latencies.
With different scheduling policies, the warps in the SM have relatively consumes less leakage energy when change in the scheduler.  
Baseline-GTO with change in the scheduler.  
Energy savings of Baseline-two-level compared to Baseline-GTO.  
However, the average energy savings of Greener are not affected significantly even with change in the scheduler.

5.9 Comparison with Different Schedulers

Figure 15 and 16 show the effectiveness of Greener when it is evaluated with GTO and two-level scheduling policies respectively. The figures compare Greener and Sleep-Reg with Baseline by measuring the reduction in leakage energy for the corresponding scheduling policies. The results show that Greener-GTO and Greener-two-level achieve an average reduction leakage energy by 69.3% and 69.65% with respect to Baseline-GTO and Baseline-two-level respectively. With different scheduling policies, the warps in the SM have different interleaving patterns, which affect the distance between the two consecutive accesses to a register. Even with the change in these access patterns, Greener shows reduction in leakage energy when compared to Baseline and Sleep-Reg. We also find that Baseline-GTO performs better than Baseline-two-level in terms of simulation cycles, hence Baseline-GTO relatively consumes less leakage energy when compared to Baseline-two-level. However, the average energy savings of Greener are not affected significantly even with change in the scheduler.

5.10 Leakage Energy for Various Threshold Distances

Table 4 shows the effect of threshold distance on register leakage energy for Greener approach. The results are collected by varying the threshold distance from 1 to 21. The table shows that keeping the threshold distance to 1 does not benefit with respect to energy consumption because with the shorter threshold distance, the registers of an instruction are turned to SLEEP or OFF state very soon, which leads to increase in the simulation cycles when there are frequent wakeup calls to the registers. Whereas, having a high threshold distance (i.e., distance 21) also does not help in minimizing energy because with longer distance, a register can be turned into SLEEP or OFF only when it is accessed after longer distance, this will result in losing the opportunity to save leakage energy. Hence, having the threshold distance at intermediate level can achieve more energy savings.

Consider the application SGEMM, it achieves lowest energy at a threshold distance of 7. Further, with an increasing in the distance from 1 to 7, the leakage energy tends to decrease, and with an increasing in the distance beyond 7 tends to increase the energy. The similar behavior can be observed with BS, NN2, MC1, PF, and SPMV applications as well. However, some applications like BS, LB, MC2, and MR2 achieve their minimum energy at a threshold distance other than 7. For some applications like BFS, the leakage energy does not change beyond particular threshold distance because a register can be turned into SLEEP or OFF only up to a certain threshold distance, beyond that the register must be kept in the ON state even with increasing the threshold distance.

Finally, we chose the threshold distance of 7 for the experiments in the paper, which achieves lowest energy for maximum number of applications. However, this value can be reconfigured depending on the application behavior.

6 Related Work

Leakage and dynamic power are the two major sources of power dissipation in CMOS technology. A comprehensive list of architectural techniques to reduce leakage and dynamic power of CPUs are discussed in [5]. Mittal et al. [21] discuss...
As expected, the leakage power savings in this (drowsy) approach are less when compared to Gated-Vdd approach. Therefore, if a register is accessed more frequently for some duration, and less frequently for other durations, then allocating the register to either of the partitions can make it less energy efficient. 

Table 4. Comparing Register Leakage Energy by Varying Threshold Distance

| Benchmark/Threshold | 1           | 3           | 5           | 7           | 9           | 11          | 13          | 15          | 17          | 19          | 21          |
|---------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| BP                  | 161.35      | 157.53      | 160.15      | 159.24      | 160.37      | 159.31      | 159.77      | 158.76      | 158.76      | 158.17      | 158.17      |
| BFS1                | 24.35       | 24.41       | 24.37       | 24.37       | 24.37       | 24.37       | 24.37       | 24.37       | 24.37       | 24.37       | 24.37       |
| BFS2                | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        | 9.48        |
| BS                  | 1,294.03    | 1,248.48    | 1,297.95    | 1,217.05    | 1,234.91    | 1,272.51    | 1,274.84    | 1,226.73    | 1,226.73    | 1,318.11    | 1,318.11    |
| LMD                 | 1.29        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        | 1.19        |
| LIB                 | 1,450.33    | 1,417.11    | 1,405.82    | 1,398.28    | 1,397.44    | 1,392.54    | 1,384.47    | 1,387.17    | 1,388.15    | 1,392.62    | 1,396.44    |
| LPS                 | 127.78      | 126.42      | 123.90      | 123.74      | 123.06      | 124.57      | 124.57      | 124.57      | 124.58      | 124.59      | 122.89      |
| MC1                 | 13.97       | 13.11       | 13.92       | 13.11       | 13.16       | 13.36       | 13.36       | 13.36       | 13.36       | 13.36       | 13.36       |
| MC2                 | 2,519.92    | 2,509.54    | 2,513.09    | 2,514.36    | 2,510.54    | 2,507.01    | 2,509.69    | 2,507.90    | 2,507.90    | 2,509.70    | 2,509.70    |
| MR1                 | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        | 0.45        |
| MR2                 | 1,610.22    | 1,604.20    | 1,588.87    | 1,690.66    | 1,609.18    | 1,609.18    | 1,573.74    | 1,573.74    | 1,666.44    | 1,638.38    | 1,638.38    |
| MUM                 | 207.28      | 205.54      | 205.18      | 208.21      | 206.68      | 205.71      | 205.31      | 204.16      | 204.16      | 206.54      | 205.67      |
| NN1                 | 35.86       | 33.32       | 33.90       | 33.67       | 33.55       | 34.53       | 34.53       | 33.52       | 33.52       | 34.61       | 34.61       |
| NN2                 | 91.32       | 86.19       | 86.86       | 80.37       | 82.30       | 84.90       | 84.83       | 82.24       | 83.68       | 83.68       | 83.68       |
| NN3                 | 1,181.92    | 1,095.75    | 1,096.35    | 1,097.49    | 1,048.82    | 1,003.42    | 1,003.22    | 1,003.88    | 1,001.88    | 1,001.88    | 1,001.88    |
| NN4                 | 11.09       | 10.47       | 10.47       | 10.50       | 10.06       | 9.72        | 9.59        | 9.87        | 9.60        | 9.60        | 9.60        |
| PF                  | 175.20      | 173.19      | 173.42      | 173.12      | 173.29      | 173.33      | 173.84      | 174.83      | 174.04      | 174.04      | 174.04      |
| SP                  | 184.20      | 164.98      | 174.55      | 165.88      | 183.55      | 183.42      | 172.22      | 172.22      | 172.22      | 172.22      | 172.22      |
| SGMEM               | 4,129.08    | 4,110.52    | 4,108.37    | 4,101.65    | 4,123.14    | 4,109.10    | 4,102.83    | 4,109.63    | 4,155.39    | 4,145.52    | 4,145.52    |
| SPVM                | 21.86       | 21.67       | 21.57       | 21.61       | 21.67       | 21.90       | 21.89       | 22.27       | 22.31       | 22.30       | 22.30       |
| VA                  | 4.09        | 4.07        | 4.05        | 4.05        | 4.05        | 4.05        | 4.05        | 4.05        | 4.05        | 4.05        | 4.05        |
number of stall cycles. Warped compression [37] exploits the register value similarity to reduce effective register file size to minimize the dynamic as well as leakage power. Gebhart et al. [38] propose two complementary techniques to reduce GPU energy. The hierarchical register file proposed by them reduces register file energy by replacing the single register file with a multi-level hierarchical register file. Further, they design a multi-level scheduler that partitions warps to active and pending warps and propose mechanisms to schedule these warps to achieve energy efficiency. These techniques mainly focus on reducing the dynamic power of GPUs and are orthogonal to our approach.

7 Conclusions and Future Work

This paper focuses on reducing the leakage power of the register file in GPUs. We discuss various opportunities to save leakage power of the registers by analyzing the access patterns of the registers. We propose a new assembly instruction format that supports the power states of instruction’s registers. Further, we provide a compiler analysis that determines the power state of each register at each program point, also transforms an input assembly to power optimized assembly code. To improve the effectiveness further, we introduce a run-time optimization that dynamically corrects the power states determined by the static analysis.

We implemented the proposed ideas in GPGPU-Sim simulator and evaluated them on several kernels from CUDASKD, GPGPU-SIM, Parboil, and Rodinia benchmark suites. We achieved an average reduction of leakage energy by 69.2% and maximum reduction of 88.41% with a negligible performance overhead when compared to baseline approach.

The register leakage power constitutes a part of the total leakage power. Similarly, other resources in the GPU such as shared memory, cache, and DRAM, dissipate leakage power during a kernel execution. In future, we plan to work on reducing the power consumption of the other GPU resources by analyzing the application behavior and the resource access patterns.

References

[1] “Kepler Architecture,” https://www.nvidia.com/content/PDF/kepler/ NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2014.
[2] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “GPUWatch: Enabling Energy Optimization in GPUS,” in International Symposium on Computer Architecture, 2013.
[3] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, “How a single chip causes massive power bills GPUsimProw: A GPGPU power simulator,” 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013.
[4] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, “Leakage Current: Moore’s Law Meets Static Power,” Computer, vol. 36, Dec. 2003.
[5] S. Kaxiras and M. Martonosi, Computer Architecture Techniques for Power-Efficiency. 1st ed., 2008.
[6] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, “Analyzing CUDA workloads using a detailed GPU simulator,” in IEEE International Symposium on Performance Analysis of Systems and Software, 2009.
[7] M. Abdel-Majeed, A. Shafaei, H. Jeon, M. Pedram, and M. Annanvar, “Pilot Register File: Energy Efficient Partitioned Register File for GPUs,” in 2017 IEEE International Symposium on High Performance Computer Architecture. HPCA 2017, Austin, TX, USA, February 4-8, 2017, pp. 589–600, 2017.
[8] M. Abdel-Majeed and M. Annanvar, “Warped Register File: A Power Efficient Register File for GPGPUs,” in High Performance Computer Architecture, 2013.
[9] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy Caches: Simple Techniques for Reducing Leakage Power,” SIGARCH Comput. Archit. News, vol. 30, May 2002.
[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing,” ACM Trans. Archit. Code Optim., vol. 10, Apr. 2013.
[11] “GPGPU-Sim Simulator,” http://www.gpupsim.org, 2014.
[12] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, “CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques,” in Proceedings of the International Conference on Computer-Aided Design, ICCAD ’11, 2011.
[13] “CUDA-SDK,” http://docs.nvidia.com/cuda/cuda-samples, 2014.
[14] “Parboil Benchmarks.” http://impact.crhc.illinois.edu/Parboil/parboil.aspx.
[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in IEEE International Symposium on Workload Characterization, 2009.
[16] “CUDA C Programming Guide.” http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, 2012.
[17] “CACTI.” http://www.hpl.hp.com/research/cacti.
[18] U. Kedker, A. Sanyal, and B. Karkare, Data Flow Analysis: Theory and Practice. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2009.
[19] “asfermi.” https://code.google.com/archive/p/asfermi/.
[20] anonymous, “Anonymized tech report.”
[21] S. Mittal, “A survey of techniques for designing and managing CPU register file,” Concurrency and Computation: Practice and Experience, 2017.
[22] S. Mittal and J. S. Vetter, “A Survey of Methods for Analyzing and Improving GPU Energy Efficiency,” ACM Comput. Surv., vol. 47, Aug. 2014.
[23] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, “Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-submicron Cache Memories,” in Proceedings of the 2000 International Symposium on Low Power Electronics and Design, ISLPED ’00, 2000.
[24] S.-H. Yang, B. Falsafi, M. D. Powell, K. Roy, and T. N. Vijaykumar, “An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches,” in Proceedings of the 7th International Symposium on High-Performance Computer Architecture, HPCA, 2001.
[25] S. Kaxiras, Z. Hu, and M. Martonosi, “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,” in Proceedings of the 28th Annual International Symposium on Computer Architecture, ISCA, 2001.
[26] M. Zhang and K. Asanović, “Fine-grain CAM-tag Cache Resizing Using Miss Tags,” in Proceedings of the 2002 International Symposium on Low Power Electronics and Design, ISLPED, 2002.
[27] J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir, “Exploiting Program Hotspots and Code Sequentiality for Instruction Cache Leakage Management,” in Proceedings of the 2003 International Symposium on Low Power Electronics and Design, ISLPED ’03, 2003.
[28] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, “Exploiting Temporal Locality in Drowsy Cache Policies,” in Proceedings of the 2Nd Conference on Computing Frontiers, CF ’05, 2005.

[29] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, “GPU Register File Virtualization,” in International Symposium on Microarchitecture, 2015.

[30] M. Abdel-Majeed, D. Wong, and M. Annavaram, “Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs,” in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, 2013.

[31] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for Reduced CPU Energy,” in Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation, OSDI ’94, 1994.

[32] F. Xie, M. Martonosi, and S. Malik, “Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits,” in Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, PLDI ’03, 2003.

[33] C.-H. Hsu and U. Kremer, “The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction,” in Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, PLDI ’03, (New York, NY, USA), pp. 38–48, ACM, 2003.

[34] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, “Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture,” in Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture, MICRO 35, 2002.

[35] A. Sethia and S. Mahlke, “Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution,” in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47, 2014.

[36] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, “Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling,” in Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques, PACT ’11, 2011.

[37] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, “Warped-compression: Enabling Power Efficient GPUs Through Register Compression,” in Proceedings of the 42Nd Annual International Symposium on Computer Architecture.

[38] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, “A Hierarchical Thread Scheduler and Register File for Energy-Efficient Throughput Processors,” ACM Trans. Comput. Syst., vol. 30, Apr. 2012.