2017 IEEE International Symposium on High Performance Computer Architecture

Pilot Register File: Energy Efficient Partitioned Register File for GPUs

Mohammad Abdel-Majeed*
Computer Engineering Department
The University of Jordan, Jordan
m.abdel-majeed@ju.edu.jo

Hyeran Jeon
Computer Engineering Department
San Jose State University, USA
hyeran.jeon@sjsu.edu

Alireza Shafaei, Massoud Pedram, Murali Annavaram
Computer Engineering Department
University of Southern California,USA
Shafaeib, pedram, annavara@usc.edu

***Abstract*—GPU adoption for general purpose computing has been accelerating. To support a large number of concurrently active threads, GPUs are provisioned with a very large register file (RF). The RF power consumption is a critical concern. One option to reduce the power consumption dramatically is to use near-threshold voltage(NTV) to operate the RF. However, operating MOSFET devices at NTV is fraught with stability and reliability concerns. The adoption of FinFET devices in chip industry is providing a promising path to operate the RF at NTV while satisfactorily tackling the stability and reliability concerns. However, the fundamental problem of NTV operation, namely slow access latency, remains. To tackle this challenge in this paper we propose to build a partitioned RF using FinFET technology. The partitioned RF design exploits our observation that applications exhibit strong preference to utilize a small subset of their registers. One way to exploit this behavior is to cache the RF content as has been proposed in recent works. However, caching leads to unnecessary area overheads since a fraction of the RF must be replicated. Furthermore, we show that caching is not efficient as we increase the number of issued instructions per cycle, which is the expected trend in GPU designs. The proposed partitioned RF splits the registers into two partitions: the highly accessed registers are stored in a small RF that switches between high and low power modes. We use the FinFET's back gate control to provide low overhead switching between the two power modes. The remaining registers are stored in a large RF partition that always operates at NTV. The assignment of the registers to the two partitions will be based on statistics collected by the a hybrid profiling technique that combines the compiler based profiling and the *pilot warp profiling* technique proposed in this paper. The partitioned FinFET RF is able to save 39% and 54% of the RF leakage and the dynamic energy, respectively, and suffers less than 2% performance overhead.**

***Keywords*-GPUs; Register File; Low power; FinFET;**

I. Introduction

One of the main draws of using graphics processing units (GPUs) to accelerate general purpose computing is their ability to power efficiently execute thousands of threads. While hand-tuned and custom-designed applications are able to truly derive the parallelism and power efficiency benefits of GPUs, many general purpose applications running on GPUs exhibit varying degrees of parallelism thereby compromising power and performance efficiency. As a result, improving performance and power efficiency has been a topic of great research interest, as can be evinced from some recent works [1–6].

GPUs are able to run massively parallel applications where thousands of threads will be running concurrently. To support such large number of concurrent threads and allow fast thread switching, GPUs rely on a very large register file (RF). Large RFs suffer from high leakage and dynamic power overheads. According to the GPUWattch [7] the RF consumes 13.4% and 17.2% of the GTX-480 and Quadro FX5600 chips power respectively. One way to reduce the RF power is to design the RF to operate at a near-threshold voltage(NTV). Research studies [8, 9] have shown that while NTV operation can dramatically reduce the power consumption it has two drawbacks. First, designing the RF to operate at NTV using planar CMOS technology, particularly at sub-16nm regime, is not feasible because of inadequate control of the short channel effects in transistors and high levels of variability (e.g. random dopant fluctuations and line edge roughness) in transistor I-V characteristics. These effects in turn result in high-leakage logic cells, low static noise margin (SNM) SRAM cells, as well as other reliability concerns [10, 11]. Second, the performance overhead due to slower access latency of a NTV register file is significant. For example, as we show in the results section, even if the NTV register file delay increases to just 3 cycles (from 1 cycle access latency at super-threshold operation) then the overall performance will degrade by 7.1%. Hence, the RF performance is critical to the GPU's overall performance.

The first concern mentioned above, regarding stability and reliability at NTV operation, has shown to be addressed effectively using the FinFET technology [11, 12]. FinFET devices offer superior scalability [13], lower gate leakage current [14], excellent control of short-channel effects [15], and relative immunization to the gate line-edge

* Most of this work was done while Mohammad Abdel-majeed was at USC

2378-203X/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCA.2017.47

roughness [16]. As a result, industry has adopted FinFET technology for their chip production since 2012 [17, 18]. Thus it is feasible to build a large but stable RF for GPUs using FinFET devices that operate at NTV. But the second concern, namely the performance impact of the slow device access time at NTV operation, cannot be addressed by simply changing the technology node.

Figure 1 shows the impact of voltage scaling on the overall speed of an inverter built using 7nm FinFET technology (simulation model and details presented in section V). In this simulation we used devices with actual gate length of 7nm and 1.5nm underlap on each side, resulting in an effective channel length of 10nm [19]. As shown, the delay of the inverter will increase as we lower the voltage . In this work, we consider a supply voltage of 0.3V as NTV while 0.45V as super-threshold voltage (STV). The delay at NTV is significantly smaller than the delay of operating the inverter in the sub-threshold regime ($V_{dd} < V_{th}$) and yet it is still significant when compared to the delay at super-threshold voltage (STV). Such slowdown may have a negative impact on the overall performance of any computing device. For instance, recent FinFET based NTV designs have shown that access delay of a 16-bit adder triples from .051ns at STV to .153ns at NTV [20]. Thus it is imperative to deploy microarchitectural solutions to tackle the slow access time of RFs built using FinFETs operating at NTV.

Our analysis of the RF access pattern in GPUs, presented in Section II, show that warps disproportionately access a small subset of their total assigned registers. In other words, the working set size of the active registers is much smaller than the total number of registers assigned to that application. We propose to exploit this observation to tackle the negative impact of the long access latency of a FinFET RF operating at NTV. We propose to divide the main register file (MRF) into two partitions: highly utilized registers, which are placed in a fast register file (FRF) partition operating at STV where they can be accessed quickly, while the remaining registers are placed in a large slow register file (SRF) partition operating always at NTV, where an application may suffer from latency penalty for accessing them. The two partitions are designed with different FinFET operating parameters so as to achieve optimal design goals (switching speed and access energy) for each partition.

An alternate approach to a partitioned RF design proposed in this work is to cache a subset of the registers. The register file cache (RFC) approach was explored in [3, 21]. In the RFC approach, a set of recently accessed registers were cached in an SRAM structure named the register file cache (RFC). RFC reduces the number of accesses to the MRF and as a result reduces the RF access energy. A detailed comparison of our proposed design with RFC is presented in the results section. But to highlight the main issue, GPU trends indicate that the number of warps issued concurrently is increasing with each generation. For example, in the

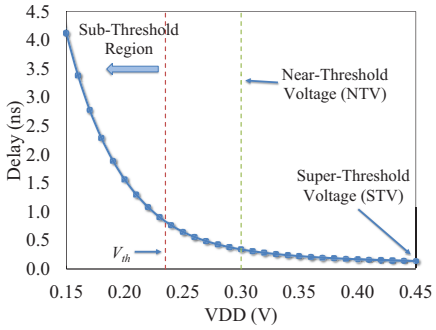


Figure 1: Delay of 40-stage FO4 inverter chain vs. Vdd for 7nm FinFET technology with V_{th}=0.23 Volts

kepler architecture [22], used in this paper, the SM has four schedulers and each scheduler can issue up to 2 instructions. In order for the schedulers to be able to issue 8 instructions each cycle they should be able to select from a larger pool of active warps. Accordingly, the RFC size should also grow with the issue width so as to cache the recently accessed registers from a larger active warp pool. Otherwise, the RFC hit rate suffers. In addition, RFC should have multiple read and write ports to service the tag check and RF access requests generated by up to 8 instructions each cycle. Hence, scaling the RFC (i.e. using larger cache and/or more read/write ports) compromises the energy efficiency of the RFC. Our results show that even with the ability to cache 6 registers for up to 16 active warps, the RFC hit rate does not go beyond 40%. Hence, still 60% of the accesses need to be serviced by the MRF, leaving a large part of energy consumption concern intact.

The contributions of the paper are as follows:

- We show a detailed analysis of the GPU RF access behavior across a wide range of workloads. Our analysis shows that all the warps within the same cooperative thread array (CTA) and across different CTAs within the same kernel have the same RF access behavior.
- We proposed the partitioned RF design to reduce the dynamic and leakage energy of the GPU's RF. The partitioned RF puts the highly accessed registers in the FRF, accessed within one cycle, and the remaining registers in the SRF partition that operates at NTV.
- We introduced a GPU-specific lightweight profiling mechanism, called *pilot warp profiling*, that exploits register access similarity across CTAs to identify the highly accessed registers. We integrated the *pilot warp profiling* technique with a *compiler-driven profiling* to introduce a *hybrid profiling* mechanism that is able to accurately identify the highly accessed registers across a wide range of workloads
- We proposed to use FinFET's binary back gate control to further reduce the FRF access energy. FinFET's binary back gate control enables fast and power efficient switching between two different power modes.

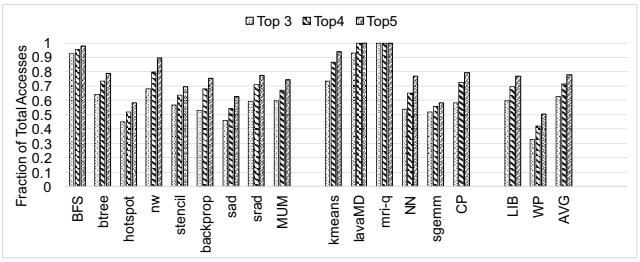


Figure 2: Percentage of accesses to the Top N highly accessed registers

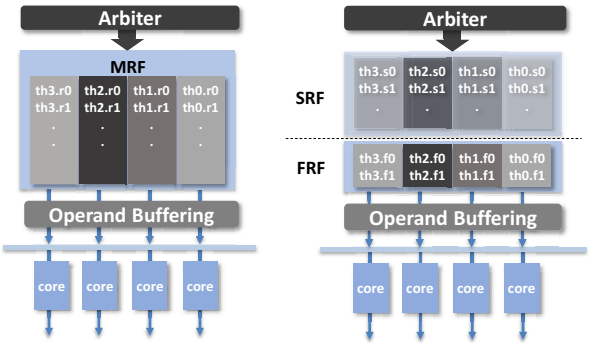
II. Register File Access Behavior in GPUs

Each kernel in a GPU application is compiled into thousands of threads. Threads are combined into hundreds of thread blocks, called cooperative thread arrays (CTAs), which are sub-divided into warps of 32 threads. The GPU runtime assigns multiple CTAs to each SM. Only when all the warps in a CTA complete a new CTA is assigned by the CTA scheduler. Table I shows the CTA information for a wide range of workloads. The second column shows the number of registers assigned per thread. The third column shows the number of threads per CTA. In Kepler architecture [22] up to 16 CTAs run concurrently inside each SM. Hence, each SM concurrently runs hundreds of threads, and a GPU chip may run thousands of concurrent threads.

Since all the threads are generated from a single kernel, all the warps will execute the same code but with different input operands. As a result, each thread that belongs to the same kernel will be allocated the same number of registers. While it is difficult to predict at compile time the access count of each allocated register, our runtime analysis of a wide range of GPU workloads shows that not all the allocated registers are equally accessed. Some registers are accessed more than other registers during the application runtime. For example, register R_0 in the backprop workload is accessed 6× more times than register R_6.

In order to quantify access variations across different workloads we calculated the percentage of accesses to the highly accessed registers in each workload. Figure 2 shows the number of accesses to the top 3, 4 and 5 highly accessed registers as a fraction of the total access count of all the registers. An access is defined as either a read or write operation. As shown, the top 3 registers in each kernel account for 62% of the total registers accesses on average across all the workloads. The top 4 and 5 registers account for 72% and 77% of the total accesses, respectively.

The highly accessed registers for each workload are different across the kernels of the same workload and across different workloads. For example, registers R_0, R_8, and R_9 are the highly accessed registers in the first kernel of backprop, while R_4, R_5, and R_6 are the highly accessed registers in the second kernel. Moreover, the highly accessed registers in CP workload are R_1, R_9, and R_{10}. Hence there is



(a) Baseline register file (b) Proposed register file

Figure 3: Baseline and proposed RFs

no correlation between the highly accessed registers between different kernels from the same workload and between kernels form different workloads.

Benchmark	Registers/ Thread	Threads/ CTA	Pilot CTA%
Category 1			
BFS	7	256	0.12%
btree	15	508	0.7%
hotspot	27	256	3.6%
nw	21	16	0.48%
stencil	15	1024	0.2%
backprop	13	256	2.6%
sad	29	61	0.13%
srad	12	256	0.6%
MUM	15	256	37%
Category 2			
kmeans	9	256	7.5%
lavaMD	6	128	0.2%
mri-q	12	512	14.3%
NN	10	169	8.2%
sgemm	27	128	16.2%
CP	12	128	47%
Category 3			
LIB	18	64	60%
WP	8	64	75%
GEOMEAN			3%

Table I: Benchmarks runtime information.

III. Partitioned Register File

Based on the above observations, in this paper we propose to exploit the access bias towards the top 3 to 5 registers through partitioning the RF. Figure 3a shows the baseline RF where all the accesses will be forwarded to the MRF. On the other hand, figure 3b shows our proposed design where the RF is partitioned into two partitions: One partition of the RF operates at STV and has a small subset of the registers and we refer to it as the fast register file (FRF). The second partition has the remaining registers and we refer to it as the slow register file (SRF). The SRF is designed to operate at NTV to save leakage and dynamic energy.

Even though FRF and SRF have significantly different access latencies, the two RFs still appear as a one RF to the

compiler and they will also be transparent to the micro-architecture blocks surrounding the RF like the operand collectors. Our proposed microarchitecture automatically decides which subset of registers are moved to the FRF and which registers are swapped back to the SRF. The RF access mechanism accesses either the FRF or SRF, but never accesses both RFs. Notice that this design is different than the hierarchical RF [3, 21] where the first level of the hierarchy will be accessed first and if the register does not exist then the RF in the second hierarchy will be accessed.

In our proposed design the partitioned RF has the same number of banks as the baseline design. But each bank is sub-divided into FRF and SRF as shown in Figure 3b. Thus only one request is active for any bank. For example, if bank 0 in the SRF is assigned a read or write request, then the arbiter cannot issue a request to bank 0 in the FRF and vice versa. This approach guarantees the transparency of the new RF design to the operand collector logic.

Using the SRF will degrade the performance due to its longer access latency. To reduce the performance impact most of the accesses must be forwarded to the FRF. Simple allocation policies that statically allocates the first few architected registers in each thread, $R_0, R_1 ... R_n$ (assuming FRF has n registers), to the FRF, while allocating the remaining registers to SRF is not beneficial. For example, if we use this assignment and assume that the first four architected registers R_0, R_1, R_2 and R_3 will be allocated in the FRF then only 25% of the accesses in the *sgemm* workload will be forwarded to the FRF. However, if we can identify the highly accessed registers in sgemm and then allocate the top four registers to the FRF then 55% of the accesses will be forwarded to the FRF. However, as discussed before, there is no correlation between the highly accessed registers between the kernels of the same workload and across different workloads. Hence, the challenge is to identify the highly accessed registers for each kernel inside each workload and find a low overhead mechanism to move these registers to the FRF.

A. Highly Accessed Registers Profiling

In this paper we propose to use profiling as an approach to identify the highly accessed register set. We first describe three profiling approaches that we explored.

1) Compiler Based Profiling: One simple approach to identify highly accessed registers is to use compiler analysis to count the occurrences of each architected register in the kernel binary. Since binary analysis is a static approach it does not account for dynamic code execution characteristics such as loop counts and branches. We instrumented the ptx compiler with RF access profiler that reports the list of registers accessed by the kernel and the number of times each register appeared in code. We compared the number of accesses to the four registers identified by the compiler as appearing most frequently in the binary with the top four

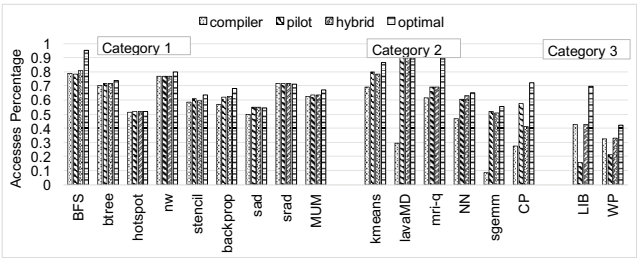


Figure 4: Efficiency of different profiling techniques

registers that are actually highly accessed. The results are shown in figure 4, first column labeled *compiler*. The Y-axis is the fraction of accesses to the four compiler identified registers over the total access count for all registers. The last column labeled *optimal* shows the fraction for the four actual highest accessed registers identified after each workload completes execution. We divided the workloads into three categories. For workloads in *Category 1* the highly accessed registers identified by the compiler account roughly for the same fraction (within 10%) of the total access counts as the registers identified by the *Pilot profiling* approach (which will be described next). For workloads in *Category 2* the fraction of total access counts to the highly accessed registers as identified by the compiler are lower by more than 10% compared to the *Pilot profiling*. For workloads in *Category 3* the fraction of total access counts to the highly accessed registers as identified by the compiler are higher by more than 10% compared to the *Pilot profiling*.

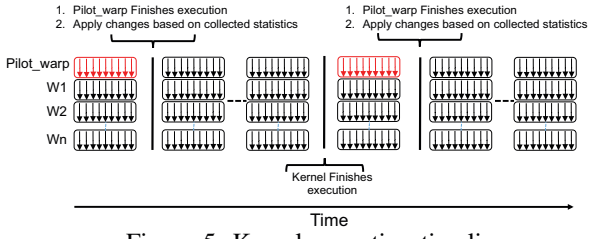


Figure 5: Kernel execution timeline

2) Pilot Warp Profiling: Workloads in *Category 2* require dynamic information, such as branch execution paths and loop counts, to accurately identify the highly accessed registers. Collecting such detailed register access count information at runtime is typically intrusive to program execution. However, GPU execution model provides a unique opportunity for a lightweight kernel profiling. In the GPU execution model each kernel is executed by thousands of warps where all the warps execute the same code. Hence, we propose to use one of the running warps as the *pilot warp* for each kernel. The *pilot warp* is one of the first running warps in each kernel. The *pilot warp* is used to collect statistics about the registers access count. Figure 5 shows how the *pilot warp* technique works. First, the *pilot warp* (marked in red) is selected from the first running CTA to collect the register access frequency statistics. Second,

when the *pilot warp* finishes execution the collected statistics will be sorted to identify the highly accessed registers. The identified registers are then placed in the FRF, as will be described shortly.

The fraction of the total accesses to the four registers identified by the pilot warp are shown in figure 4, second column labeled *pilot*. As shown in the figure, *pilot warp* is able to more accurately identify the highly accessed registers across a wider range of workloads in *Category 1* and *Category 2*.

Runtime Overhead: For the pilot warp profiling technique to be effective, the pilot warp should provide the information about the highly accessed registers early enough in the kernel execution time so the rest of the kernel CTAs can take advantage of allocating the highly accessed registers in the FRF. The last column in table I shows the percentage of time the pilot warp runs normalized to the kernel execution time. As shown, on average the pilot warp runs for less than 3% of the time. However, workloads like LIB and WP have very few warps in the kernel. Hence, the *pilot warp* itself takes a significant fraction of the overall execution time to complete. As such by the time the *pilot warp* completes its execution there is not enough time for other warps to benefit from the profiling information gathered by the pilot. As such, the *pilot warp profiling* may not be effective. Hence, in the next subsection we introduce the *hybrid profiling technique* to mitigate this effect.

Code Dynamics: While the *pilot warp* executes the same code as the other remaining warps in the kernel, the dynamic flow of the instructions may be different because of the branch divergence. To analyze the impact of divergence on the results we calculated the number of accesses to the registers in the RF across different warps from the same CTA and from different CTAs. Our results show that on average the number of accesses to various registers differ by no more than 5% irrespective of which warp is selected as a *pilot warp* in any CTA. Even more encouraging is the fact that even when the absolute register access count is not identical, the sorted list of registers based on access count is the same across the warps within the same CTAs and the warps across different CTAs in the same kernel.

3) Hybrid Profiling: Pilot warp profiling relies on at least one warp to finish its execution quickly and provide the access statistics to decide which registers are allocated to FRF and which registers are placed in SRF. As discussed above, in some workloads like LIB and WP the pilot warp takes too long to complete and as a result reducing its effectiveness. To tackle this challenge we propose the *hybrid profiling* technique. In the *hybrid profiling* we combine the *compiler based profiling* and the *pilot warp profiling*. In the *hybrid profiling* approach the compiler based profiling data is used initially while the pilot warp is running to decide which registers are allocated to the FRF. Once the pilot warp finishes execution then the dynamic profile data provided by the *pilot warp* is used in determining highly accessed registers. The third column in figure 4, labeled *hybrid*, shows the efficiency of the hybrid profiling. Clearly, for workloads where the pilot's execution time is small *hybrid profiling* essentially becomes *pilot warp profiling*. But for workloads where the pilot warp runs for a long time *hybrid profiling* technique provides a reasonable initial register classification that may be used to manage the FRF.

B. Architectural Support

To enable the design of the partitioned RF we should determine access frequencies of registers during *pilot warp* execution. To collect that information a set of 63X2-byte counters are added to each SM. The selection of 63 counters is influenced by the fact that each thread can be allocated at most 63 registers in our simulated GPU design. However, based on the benchmark info shown in table I on average 16 registers were allocated for each workload. As a result, not all the counters are active during the register access profiling phase. Additionally a one byte *pilot-warp-id* is added to the SM. Once a pilot warp is selected its *id* is stored in the pilot-warp-id register. Also a profiling mask bit is added to determine if we are in the pilot warp profiling phase. The profile mask bit is initially set to one to indicate that a pilot warp is currently in the process of collecting the access counts. Once the pilot warp terminates it automatically resets the mask bit. The mask bit is set again only when a new kernel is launched on the SM.

When a warp instruction is scheduled for register access the profile mask bit is checked. If the mask bit is reset then the register access proceeds normally. But if the mask bit is set then the warp id of the register access warp is compared against the warp id stored in the pilot-warp-id register. If they match then the access count for the register that is about to be accessed is incremented. Since we have 63 access counters these are simply indexed by the register number.

Once the pilot warp completes execution the counters values are compared to identify the top n most accessed registers. Sorting operation can be implemented efficiently on GPU itself using built-in ISA support such as the SHFL instruction in Kepler [23]. The counters and the sorting logic are only used when the pilot warp is active which counts only for 3% of the kernel execution time. Hence, the power of the additional sorting is negligible because it is active once per SM over the entire duration of the kernel.

To allocate the highly accessed registers in the FRF and the remaining registers in the SRF, we propose a simple register swapping approach supported with a renaming table. For example if R_{n+2} (which is located in the SRF by default) is one of the highly accessed registers for that specific workload and R_0 (which is located in the FRF by default) is not one of the top n highly accessed registers, then R_{n+2} is swapped into FRF in place of R_0, while R_0 is placed at R_{n+2}. In order to enable the swapping process

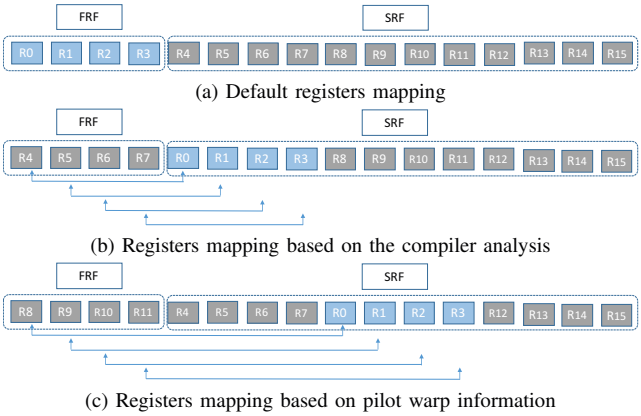
a swapping table is integrated in each SM. The swapping table can be implemented as a small CAM structure that is responsible for holding the register mapping for the top n most accessed registers. There are $2n$ entries in the table (i.e. n entries for the originally mapped register and n entries for the new top most accessed n registers). Each entry stores the original architected register number and the current mapping of that register in either SRF or FRF. Hence, if we are storing the top 4 accessed registers then the swapping table will have 8 entries and each entry has 13 bits (i.e. 6 bits for the original register *id*, 6 bits for the swapped register *id* and 1 valid bit) for a total size of 104 bits. Note that the swapping table only changes the mapping for 8 registers at most.

We explored both the indexed and the CAM based designs for the swapping table but given its small size and access energy compared to the RF the differences between the two options are negligible. Hence, without loss of generality, we used the CAM design in the rest of the paper for explanation purposes. Even if the indexed design is used the results are unchanged. Furthermore, we did detailed RTL evaluation for the swapping table using 22nm CMOS, 16nm CMOS and 7nm FinFET and the results show that the delay of the swapping table is 105, 95 and 55ps, which is less than 10% of a typical GPU clock cycle (900Mhz). Given the small size of the table we assumed we can integrate the table access with the register access time. But if we conservatively assumed that the swapping table access adds one cycle to the register access pipeline then the overall performance overhead is still less than 1%.

In order to serve the requests of one warp the swapping table should have three read/write ports. However, in our evaluation we assumed that up to 8 instructions can be issued each cycle. Hence, to make sure that the swapping table is not the bottleneck we propose to *replicate* the swapping table (104 bits/table) to support concurrent accesses by all the warps rather than supporting large number of ports. Note that swapping table is only configured once after the end of a pilot warp run and hence replication is easier without worrying about consistency of updates.

Example: Figure 6 shows an example on the mapping of the registers between the FRF and the SRF. This example shows the details of the RF mapping when the *hybrid profiling* technique is used. As shown in figure 6a, before running the workload the registers will be physically divided between the FRF and the SRF. The first few architected registers, such as R_0, R_1, R_2 and R_3, are initially in the FRF and the remaining registers will be mapped to the SRF.

As discussed earlier, we use the *hybrid profiling* technique in our implementation. Hence, after the kernel starts execution and before the pilot warp finishes execution the highly accessed registers reported by the compiler analysis will be mapped to the FRF. For illustrative purposes in this example we assume that compiler based profiling identified R_4, R_5, R_6 and R_7 as the highly accessed registers. Hence,



(a) Default registers mapping

(b) Registers mapping based on the compiler analysis

(c) Registers mapping based on pilot warp information

Figure 6: Registers mapping between FRF and SRF when hybrid profiling is used

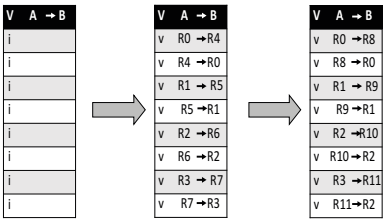


Figure 7: The swapping table content before and after the pilot warp finishes execution

registers R_4, R_5, R_6 and R_7 are mapped to the FRF and the remaining registers are mapped to the SRF as shown in figure 6b. In this case whenever a request to R_4, R_5, R_6 and R_7 is issued then the request is forwarded to the FRF, all other requests are forwarded to the SRF.

After the pilot warp finishes execution the highly accessed reported by the pilot warp profiling (say, R_8, R_9, R_{10} and R_{11} in this example) will replace the highly accessed registers reported by the compiler based profiling. However to simplify the design of the swapping table, before mapping the highly accessed registers the current mapping will be reset back to the original mapping shown in figure 6a and then the mapping of the highly accessed registers reported by the pilot warp will be applied as shown in figure 6c.

Figure 7 shows the content of the swapping table before the application starts execution (figure 7(left)), after the compiler based data is used (figure 7(middle)), and after the pilot warp finishes execution (figure 7(right)). Each entry shows the valid bit and the architectural register number and the location of the mapped register. For instance, as shown in the table since R_8 is the most accessed register then the first entry stores the information that R_0 is mapped to R_8 and the second entry stores the information that R_8 is mapped to R_0 and so on. To access register R_0 the swapping table is CAM searched for R_0 to find that the register R_0 is stored in the location associated with register R_8. On the other hand, if the register does not have an entry in the table this means that the old and the new locations are the same.

IV. Partitioned RF Design Using FinFET

A. SRAM Cell Design in Sub-10nm

As technology scales, operating the SRAM cells at lower voltage causes stability issues especially when we scale the voltage closer to the minimum data retention voltage, V_{DDMIN}. V_{DDMIN} is basically the minimum voltage required to make sure that the SRAM read and write operations are stable and do not cause any read or write failures. To eliminate stability concerns at low Vdd, SRAM cells can be sized up [24] or different SRAM structures like 8T [25], 9T [26] and 10T [27] may be used.

On the manufacturing front, at the sub-10nm regime MOSFETs suffer from process variations due to random dopant fluctuation (RDF) [11, 28]. Due to their high scalability and immunity to RDF, FinFET has attracted the attention of industry as a reliable solution for building future chips. In fact Intel and Samsung [17, 18] have already announced the production of FinFET chips or have placed them on their roadmap for the near future. FinFET devices use un-doped channels . Hence, these devices do not suffer from random dopant fluctuations. However, FinFETs still suffer from line edge roughness, which causes variations of the (effective) channel length L, as well as work function variations, which cause variations of the gate material properties [28]. Both types of variations affect the threshold voltage and the sub-threshold slope of FinFET devices. In this work we propose to use FinFET devices operating at NTV for building the GPU RF. Since SRAM cells are the basic building blocks for storage elements, such as registers, we performed a detailed yield analysis under aforesaid process variations for a wide range of SRAM cell designs. We designed 6T, 8T, 9T and 10T SRAM cells using FinFETs with an effective gate length of 7nm, with 1.5nm underlap on each side. The STV of these devices is 0.45V, and NTV is 0.3V. We then did a detailed Monte Carlo simulation of Hpsice models of these various SRAM cell designs. Our results show that FinFET based 8T SRAM cells provide an ideal design tradeoff between area and static noise margin (SNM) constraints. In particular, the SNM from hspice simulations is 0.144V at STV and 0.092V at NTV. On the other hand, the 6T SRAM cells even with a larger cell size than the 8T SRAM cells have 0.088V SNM at STV (more detailed results are presented in our evaluation section later). Hence, we conclude that these 8T FinFET SRAM cells are well suited for designing the proposed RF. As we show later the 8T SRAM cell design with FinFETs is exploited to design the binary back gate control of the FinFET devices.

B. SRF Design Using FinFET

The SRF is responsible for holding the registers that are not highly accessed. As a result, the access latency demand on the SRF is small. Hence, we chose to operate the SRF at NTV. The SRF operating in NTV has 3X longer access delay than the FRF operating at STV. Thus SRF saves dynamic and leakage energy by reducing the operating voltage but paying for it using additional access latency.

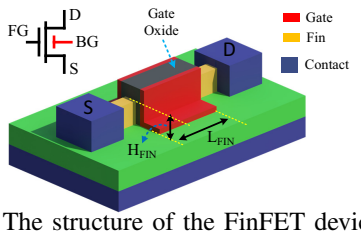


Figure 8: The structure of the FinFET device model

C. FRF Design Using FinFET

FRF is responsible for holding the highly accessed registers and hence its access latency must be minimized. As such, the FRF operates at nominal Vdd. The FRF is accessed in one cycle. Since almost 60% of the accesses are forwarded to the FRF, we look for further reducing the access energy of the FRF without affecting performance. To achieve that we should be able to switch the FRF between low power and high power modes, and at the same time we need a fast switching mechanism that enables the FRF to switch between the two modes. Hence, we introduce the adaptive FRF, which exploits the unique back gate control capability of FinFETs to further reduce the FRF energy without affecting performance.

1) Adaptive FRF Design: Figure 8 shows the basic structure of the dual gate (DG) FinFET transistor. As shown, the gate straddles the channel (fin) rather than connecting to the channel through a planar surface. This design provides better electrical characteristics and provides better control over the leakage and short channel affects. The DG FinFET has two gates that control the channel formation. For example, the channel formation in the DG device can be done by turning both gates ON by connecting them to V_{dd}, or by turning ON just the front gate. However, the channel formation ability of one gate is highly dependent on the voltage status of the other gate. When both gates are ON the channel is formed and the gate capacitance is C_g. On the other hand, if the front gate is ON and the back gate is disabled, then only the channel of the front gate is formed. As a result, the driving current is smaller. However, the gate capacitance of the device is half of the C_g. As a result, the dynamic energy of the device is lower. In addition, with the disabled back gate the threshold voltage of the device is higher and this results in reducing the leakage power. In the adaptive FRF we leverage the *binary back gate control* to operate the front gate and back gate separately. Hence we propose to design the FRF such that it can switch between the FRF low power mode (FRF_{low}) and high power mode (FRF_{high}).

Figure 9 shows the FRF SRAM cell design. The red lines in the figure shows the extra connections that are needed to operate the back gate of the transistors. When operating in

the FRF_{high} mode the back gate control is set to V_{dd} and when operating in the FRF_{low} mode the back gate control is set to GND. To enable this control we modified the decode circuitry of the FRF as shown in Figure 9. The $mode$ signal determines if the FRF operates in the FRF_{high} or the FRF_{low} modes. To enable mode switch, signal buffers are added to drive the control signals to all SRAM cells in the same row. When the FRF is operating in the FRF_{high} mode it is accessed in one cycle and when it is operating in the FRF_{low} mode it is accessed in two cycles. Note that while SRF saves energy by reducing voltage, dual-mode FRF saves energy by reducing capacitance. But the potential for energy savings with voltage reduction, as is done with SRF, is much larger than reducing just the capacitance. However, the latency impact due to capacitance reduction is not as severe as the latency impact due to voltage. Hence, choosing to reduce capacitance in FRF, as opposed to voltage, provides a good tradeoff between performance and power.

The GPU pipeline uses the operand collectors in order to buffer the operands for the scheduled warps before being issued to the execute stage. Hence, the GPU pipeline is designed to deal with the variation in RF access time. For example, in the base machine the registers that are accessing the same bank are scheduled back to back to resolve conflict As such, the adaptive FRF design with multiple latencies may be handled by the operand collector logic effectively.

FRF switching policy: We propose to switch the FRF to operate in $FRF_{low}mode$ when the GPU's execution pipeline has limited instruction level parallelism. To detect this scenario we use a simple epoch based phase detection. During every 50 cycles the total number of instructions that are scheduled for execution are counted. The counting is done by a 9-bit counter which increments every time a warp is issued to the execution unit. For instance, in our baseline Kepler architecture, at most 8 instructions can be issued every cycle. Hence, at most 400 instructions can be issued in a 50 cycles epoch. At the end of the epoch if the number of instructions issued is less than certain threshold, then we assume the GPU is in a low compute phase. Accordingly, in the next epoch we simply use the back gate control of the FinFET device to put the FRF register in low power mode by setting the *mode* signal to *'0'*. As will be shown in the evaluation section, this simple technique is sufficient in predicting when to switch FRF into high power or low power modes.

V. Evaluation

We evaluated our proposed techniques using GPGPU-Sim v3.02 [29]. Table II shows the GPGPU-Sim configuration that we used in our simulations. In our analysis we ran all the workloads from Rodinia [30], Parboil [31] benchmark suites, and the workloads provided with the GPGPU-Sim [29] and we classified them into 3 Categories as shown in Figure 4 and presented the results for workloads in each category.

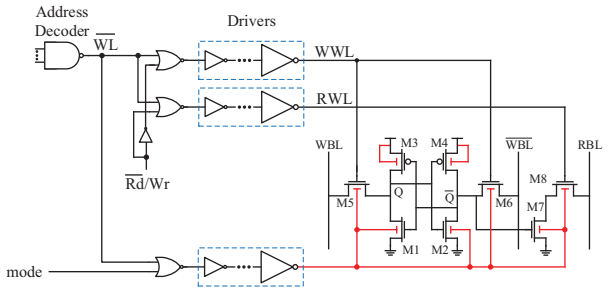


Figure 9: Schematic for the modified decoder and the SRAM cell structure

For the schedulers we used the two-level (TL) scheduler proposed to support the RFC design [3]. In addition we ran our experiments with the GTO and the fetch group schedulers [2]. Our technique shows a consistent performance across all the schedulers. In this section we show the results for the TL and GTO warp schedulers with the hybrid profiling technique as our preferred design choice. Also when applicable we compare with the compiler based profiling technique.

Category	Value
GPGPU-Architecture	
Architecture	Kepler-GTX 780
SMs	15
Warps per SM	64
SM-Architecture	
SIMT clusters	6
SIMT lanes per SIMT cluster	32
Register File-Architecture	
Size	256KB
Banks	24
Operand collector units	24

Table II: Experimental setup

A. Proposed Register File Characteristics

In order to evaluate the energy, timing and area characteristics of the proposed RF we updated the 7nm FinFET models used in [25] to model the DG FinFET devices and the back gate control used in our proposed RF. We used Synopsys Technology Computer-Aided Design (TCAD) tool suite [32] to generate the device characteristics like the gate capacitance and the ON/OFF currents. Also we modified the decoder circuitry of the RF in FinCACTI [25] to include the *mode* signal required to switch between the FRF_{high} and FRF_{low} modes and the additional signal buffers as shown in Figure 9. We modeled the FinFET partitioned RF by mapping 4 registers per warp to the FRF and the remaining registers to the SRF. Since Kepler supports 64 warps [22], the FRF size is 32KB and the SRF size is 224KB, which totals up to 256KB of RF, which is the size of RF in Kepler. Thus FRF is only 12.5% of the total RF size. We also evaluated the power-aggressive design with just one main register file (MRF) of 256KB operating at STV. This

Design	Voltage (V)	ON current (A/um)	SNM (V)
NTV	0.3	7.505E-04	0.092
STV, BG=Vdd	0.45	2.372E-03	0.144
STV, BG=0	0.45	2.427E-04	0.096

Table III: Characteristics of the 8T SRAM cells built in FinFET 7nm technology

RF type	Access energy (pJ)	Leakage power(mW)	Size (kB)
FRF_{low}	5.25	7.28	32
FRF_{high}	7.65	7.28	32
SRF	7.03	13.4	224
MRF	14.9	33.8	256

Table IV: Size, access energy and leakage power for the power aggressive baseline and proposed register file

design gives the highest performance and we will compare our partitioned RF against this baseline.

Table III shows the operating voltage, the ON current, and the SNM for three different 8T SRAM cells used in this paper which were generated from the Hspice Monte Carlo simulations. The first row shows the details for the SRAM cell that operates at the NTV. The second and the third rows show the details for the SRAM cells operating at STV when the back gate control is enabled and disabled, respectively. When both the front and back gate are enabled (turned to ON state) the current is 9 times larger than enabling just the front gate. Hence, the delay of the gate can vary vastly depending on whether the back gate is enabled or disabled. However, the reduction in the C_g makes the degradation in the device delay smaller. The last column shows the SNM for the three different cells. As shown, the SNM for the NTV SRAM cell and the STV SRAM cell with back gate disabled is smaller than the SNM of the STV SRAM cell.

Table IV shows the access energy, leakage power, and the size for the partitioned RF structure with the above mentioned sizes. As shown the SRF that operates continuously at NTV has an access energy of 7.03 pico Joules (pJ), compared to monolithic MRF that operates at STV which has an access energy of 14.98 pJ. Due to its smaller size the FRF access energy is 7.65 pJ even when it operates in the FRF_{high} mode. When FRF_{low} mode is enabled the capacitance reduction leads to a further reduction in access energy to 5.25 pJ. We measured the area overhead for our additional connections and structures. This includes the additional wire connections from the RF decoder to the back gates of the FinFET transistors, the required buffers to drive the $mode$ signal, and additional layout area overhead for back gate connections of the SRAM cells. Based on FinCACTI [25] simulations, the area of the baseline RF is 0.2mm^2, whereas the area of the our proposed RF is 0.214mm^2, resulting in less than 10% area overhead. Also we calculated the area of the additional counters and the sort logic used by the *pilot warp*, which are negligible compared to the area of the RF.

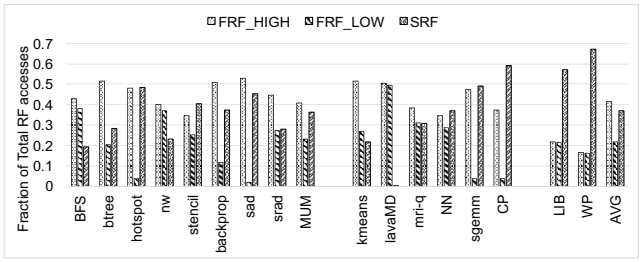


Figure 10: Partitioned register file access distribution

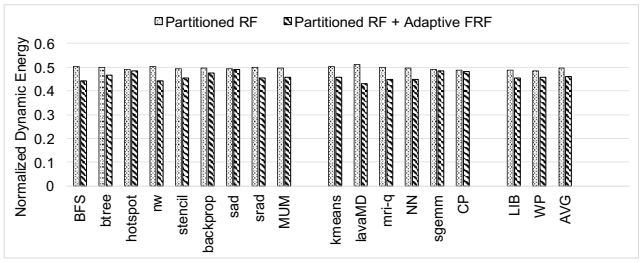


Figure 11: Energy savings

B. Energy Savings

We modified the GPGPU-Sim [29] to model the proposed RF. Based on the Hspice and FinCACTI simulations, the FRF_{high} access time is 0.08ns. Hence FRF_{high} can be accessed in 1 clock cycle even at high frequencies. In addition, when the FRF operates in the FRF_{low} mode then it is accessed in 2 cycles and the SRF is accessed in 3 cycles. In addition, we model the swapping table in GPGPU-Sim in order to forward the accesses to the correct RF as described earlier. Figure 10 shows the distribution of the accesses based on the RF type and the power mode of the FRF. The data is generated assuming four registers are in the FRF and the remaining registers are in SRF. As shown the proposed partitioned RF is able to forward 62% of the accesses to the FRF.

The percentage of time the FRF is operating in the FRF_{low} mode depends on the selected threshold for the number of issued instructions in an epoch. We did a detailed design space exploration of this threshold to see the energy savings versus potential performance penalties. Our results show that any threshold around 85 works well (average performance overhead is less than 0.5%), namely if the number of issued instructions in a 50 cycle window is less than 85 out of a total of 400 issue slots the system is considered operating in a low compute phase. At this threshold 22% of the accesses to the FRF take place when the FRF is in the FRF_{low} mode. As expected, applications with high compute demand that do not waste issue slots, such as sad and hotspot, do not enter into low compute phase often and hence most of their FRF accesses occur during FRF_{high} mode.

Figure 11 shows the dynamic energy for our proposed partitioned RF and the partitioned RF plus the adaptive

FRF normalized to the baseline where a single large MRF operates at STV. This data is obtained by multiplying the number of accesses to each of the three register operating modes by the energy per access in the corresponding mode. The partitioned RF saves 54% of the RF dynamic energy across all the benchmarks. We also compared the partitioned RF design energy against the MRF that always operates at NTV. Our results show that when the monolithic RF operates at NTV it saves 47% of the RF energy, which is lower than the partitioned RF savings. The reason for that is the extra savings from the adaptive FRF technique that allows us to reduce the access energy further.

In addition to the dynamic energy, the partitioned RF is able to save leakage power. The leakage power savings come from the SRF where all the registers are operating in the near-threshold region. The leakage power of the SRF and the FRF are shown in table IV. The FRF leakage power is almost 21.5% of the MRF baseline leakage power since its size is smaller than the MRF. In addition, the SRF leakage power is almost 39.7% of the leakage power of the MRF because it is operating at NTV. Hence our proposed RF is able to save 39% of the RF leakage power.

C. Performance Overhead

Figure 12 shows the performance overhead of the proposed techniques using the GTO, and the TL [3] schedulers. The numbers show the normalized performance to the baseline with MRF running at STV and using the same scheduler (i.e. GTO shows the normalized performance to the base machine using the GTO scheduler and TL shows the normalized performance to the base machine using TL scheduler). As shown, our technique has less than 2% performance overhead when GTO scheduler is used. On the other hand, when the MRF in the baseline machine operates at NTV all the time then it suffers from 7.1% performance overhead. Hence the partitioned RF is able to achieve better energy savings with negligible performance loss. In addition, compared to the compiler based profiling, our hybrid profiling technique has 2% better performance. In addition, the hybrid profiling technique avoids the significant degradation in performance when the compiler based profiling is not accurate as in Category 2 workloads.

To measure the impact of the SRF speed on the overall performance in our proposed technique, we ran our workloads with longer SRF access latency . Our results show only 0.5% and 2.4% degradation in performance when the access delay to the SRF is 4 cycles and 5 cycles, respectively.

As mentioned before we used an epoch length of 50 cycles in our adaptive FRF technique. To verify that our results are not sensitive to the epoch length we ran our simulations for different epochs length. In our experiments we used the same threshold ratio. For example, when the epoch length is 50 cycles we used 85 out of 400 as the threshold value(i.e the workload issued less than 85 instructions out of the maximum possible issues). So we set the threshold to be 20% in all the sensitivity simulations. The results show that the epoch length has a small impact on performance.

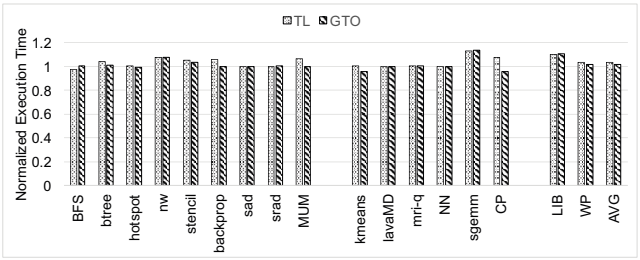


Figure 12: The execution time of the proposed ideas

D. Partitioned vs Hierarchal Register Files

One way to avoid accessing the MRF is to use the hierarchical RF where the multi-level RF is accessed similar to caches. The first level includes the recently accessed registers. If the register is not in the first level then the second level is checked. The work proposed in [3] applied the hierarchical RF in GPUs. Since multiple warps are concurrently running at the same time the authors proposed the TL scheduler [3] to reduce the size of the RFC.

With the increase in the number of the schedulers/SM and the number of issued instructions each cycle, the RFC must handle all the concurrent requests efficiently in future designs. The authors in [3] proposed using a multi-ported RF to be able to read all the operands in one cycle. This solution works well with one scheduler per SM issuing two instructions per cycle. However, when applied to newer architectures that issue up to 8 instructions per cycle the number of cache ports must also be scaled. To understand just the energy impact of adding more ports, we simulated RFC with varying number of read(R)/write(W) ports using the FinCacti [25]. When (R=2, W=1) ports are used the access energy of RFC that can hold 6 registers in the RFC per warp is 0.37X of the MRF. But when the port count is increased (R=8, W=4) to support 4 instruction issue per cycle the access energy of RFC is 3X the access energy of MRF access energy. Hence, the energy savings from accessing the RFC are compromised as we use more ports.

Another approach to reduce port contention is to use a banked RFC design. Our simulations show that the access energy of an 8 banked RFC is nearly the same as the access energy of MRF. Also distributing the RFC between the schedulers where each scheduler has its own RFC complicates the design significantly since the register writeback stage must then write to different RFC banks each associated with different scheduler.

Since increasing the number of ports is expensive, we did a quantitative comparison between the RFC and the partitioned RF when we increase the number of banks only. Figure 13 shows how the dynamic energy and the

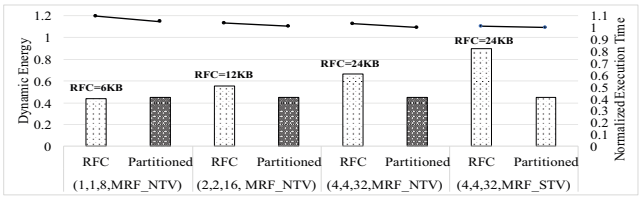


Figure 13: Scalability of the RFC and partitioned register file

execution time of the RFC and the partitioned RF scales as we increase the number of banks without increasing the number of read/write ports. In the experiments we used four different configuration parameters that follow the trend of GPUs scaling. These four parameters are listed under each set of bars in the figure inside the parenthesis. The first parameter in the configuration shows the number of schedulers/SM. The second shows the number of RFC banks and the third shows the number of active warps. The last parameter shows the operating region of the MRF when the RFC is used. The size of the RFC in each configuration is shown on top of the RFC bars in the figure. Note that as we increase the number of active warps the RFC size increases accordingly to accommodate the registers of all active warps. For fair comparison we assumed that the MRF is operating in NTV. As the first set of bars show, the dynamic energy savings of the RFC is close to the dynamic energy savings of the partitioned RF. However, as we increase the number of schedulers/SM and the number of active warps the RFC energy savings decreases as shown in the second and third sets of bars. On the other hand, the partitioned RF has constant energy savings all the time since its energy savings does not rely on the number of active warps. We can correlate the low energy savings when the RFC is used to the low RFC hit rate. Our simulations show that when we have 32 active warps the RFC hit rate is lower than 45%. Hence, more than half of the accesses are forwarded to the MRF. The lines on top of each configuration shows the execution time of the RFC and the partitioned RF. As shown the partitioned RF has lower performance overhead compared to the RFC. The RFC has 9.5%, 3.8% and 3.3% performance overhead when the number of active warps is 8, 16 and 32 ,respectively. The last set of bars shows the dynamic energy savings of the RFC when the MRF is operating at STV. In this region, the RFC and the MRF are accessible in one cycle. In such scenario there is no performance overhead. However, the RFC saves only 10% of the RF dynamic energy.

VI. Related Work

GPU power efficiency:GPU power efficiency and performance have been widely studied. Several techniques tackle the inefficiency in the GPUs at the RF level [2, 4, 6] and the execution units level [5]. Several works [3, 4, 6] proposed techniques to save dynamic and static power of the GPUs RFs using circuit level and micro-architectural techniques. We already provided a detailed comparison of our work with the RFC [3]. Others explored the option of power gating and drowsing unused registers [4]. The authors in [33] proposed CMOS/TFET hybrid RF for GPUs. The authors proposed moving the warps registers between the CMOS and TFET register files to improve the RF energy efficiency and reduce the memory contention. A renaming table is used to store the mapping for each register. However, as the number of concurrent warps and the registers per warp increases ,as in kepler, the scalability and the size of the renaming table become the bottleneck because more warps will be running concurrently and the number of requests to the renaming table increases significantly. On the other hand, in our proposed techniques we relied on a low overhead profiling technique and added an 8-entry swapping table (104 bits) that stores the mapping for all the threads in the kernel and is updated once at the beginning of each kernel.

Power efficient SRAM cells: Several works have been proposed to improve the power efficiency of the SRAM structures. The authors in [25] show that the 8T SRAM cells and the FinFET technology enable building a stable SRAM at low operating voltage. In addition, the authors in [34] proposed using different SRAM cells sizes in the design of multi-Vcc caches. Also, multiple techniques have been proposed to reduce the failure probability of the SRAM cells at low voltage [34, 35]. In this work we take advantage of the proposed techniques to enable the design of the SRF.

Near threshold computing: Near threshold computing has been widely used to build different micro-architectural blocks and processors. The authors in [36, 37] designed complete processors that operate at NTV. In addition, the authors in [38] proposed mixing slow cores operating at NTV with faster L1 caches to improve the energy efficiency. The authors in [8] proposed building the caches using mix of NTV tolerant ways to reduce energy and traditional ways to maintain performance. The access policy of the cache is changed at runtime to minimize the overhead. The authors in [20] did a broad analysis on designing the execution units at NTV using different technologies nodes. In our paper we focused on designing low power RF for high throughput processors like GPUs. To achieve that we divided the RF into SRF that is used to hold the low accessed registers and FRF that is used to hold the most accessed registers. Also we proposed an adaptive technique that control the placement of the registers in the FRF and SRF.

VII. Conclusion

In this paper we proposed dividing the GPU RF into FRF and SRF. The partitioning is done based on a hybrid technique that combines the compiler based profiling and a novel pilot warp profiling techniques. The FRF holds the highly accessed registers and the SRF holds the remaining registers.

We target our RF design for future sub-10nm technology where MOSFETs are shown to be difficult to build. As such there is a strong industry trend to move towards FinFET designs. We build the partitioned RF using FinFET 7nm technology where the SRF is built to operate at NTV and the FRF to operate at STV. Our proposed techniques are able to save 39% and 54% of the RF leakage and dynamic energy with less than 2% performance overhead.

ACKNOWLEDGMENT

This work was supported by DARPAPERFECT-HR0011-12-2-0020 and NSF-CAREER-0954211 grants.

REFERENCES

[1] M. Abdel-Majeed, D. Wong, J. Kuang, and M. Annavaram, "Origami: Folding warps for energy efficient gpus," in *ICS 2016*.

[2] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving gpu performance via large warps and two-level warp scheduling," in *MICRO 2011*.

[3] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," in *ISCA 2011*.

[4] M. Abdel-Majeed and M. Annavaram, "Warped register file: A power efficient register file for gpgpus," in *HPCA 2013*.

[5] M. Abdel-Majeed, D. Wong, and M. Annavaram, "Warped gates: Gating aware scheduling and power gating for gpgpus," in *MICRO 2013*.

[6] W.-k. S. Yu, R. Huang, S. Q. Xu, S.-E. Wang, E. Kan, and G. E. Suh, "Sram-dram hybrid memory with applications to efficient rfs in fine-grained multi-threading," in *ISCA 2011*.

[7] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi1, "Gpuwattch: Enabling energy optimizations in gpgpus," in *ISCA 2013*.

[8] R. Dreslinski, G. Chen, T. Mudge, D. Blaauw, D. Sylvester, and K. Flautner, "Reconfigurable energy efficient near threshold cache architectures," in *MICRO 2008*.

[9] U. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "Energysmart: Toward energy-efficient manycores for near-threshold computing," in *HPCA 2013*.

[10] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N. Rohrer, "High performance cmos variability in the 65nm regime and beyond," in *IEDM 2007*.

[11] X. Wang, A. Brown, B. Cheng, and A. Asenov, "Statistical variability and reliability in nanoscale finfets," in *IEDM 2011*.

[12] Z. Guo, S. Balasubramanian, R. Zlatanovici, T.-J. King, and B. Nikolic, "Finfet-based sram design," in *ISLPED 2005*.

[13] C. Wann, K. Noda, T. Tanaka, M. Yoshida, and C. Hu, "A comparative study of advanced mosfet concepts," *ED 1996*.

[14] L. Chang, K. Yang, Y.-C. Yeo, Y.-K. Choi, T.-J. King, and C. Hu, "Reduction of direct-tunneling gate leakage current in double-gate and ultra-thin body mosfets," in *IEDM 2001 (Technical Digest)*.

[15] B. Yu, L. Chang, S. Ahmed, H. Wang, S. Bell, C.-Y. Yang, C. Tabery, C. Ho, Q. Xiang, T.-J. King, J. Bokor, C. Hu, M.-R. Lin, and D. Kyser, "Finfet scaling to 10 nm gate length," in *IEDM 2002*.

[16] A. Brown, A. Asenov, and J. Watling, "Intrinsic fluctuations in sub 10-nm double-gate mosfets introduced by discreteness of charge and matter," *NANO 2002*.

[17] "Samsung shows 14nm chip," http://www.eetimes.com.

[18] "Intel and ibm to layout 14nm finfet strategies on competing substrates," http://electroiq.com/blog/2014/10/intel-and-ibm-lay-out-14nm-finfet-strategies-on-competing-substrates-at-iedm-2014/.

[19] X. L. Q. X. S. Chen, Y. Wang and M. Pedram., "Performance prediction for multiple-threshold 7nm-finfet-based circuits operating in multiple voltage regimes using a cross-layer simulation framework," *IEEE S3S 2014*.

[20] Q. Xie, X. Lin, Y. Wang, M. Dousti, A. Shafaei, M. Ghasemi-Gol, and M. Pedram, "5nm finfet standard cell library optimization and circuit synthesis in near-and super-threshold voltage regimes," in *ISVLSI 2014*.

[21] J.-L. Cruz, A. González, M. Valero, and N. P. Topham, "Multiple-banked register file architectures," in *ISCA 2000*.

[22] "Nvidia next generation cuda compute architecture: Kepler tm gk110," Nvidia, Tech. Rep., 2012.

[23] "Uda pro tip: Do the kepler shuffle," http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-kepler-shuffle/.

[24] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim, "Minimizing total area of low-voltage sram arrays through joint optimization of cell size, redundancy, and ecc," in *ICCD 2010*.

[25] X. L. Alireza Shafaei, Yanzhi Wang and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *ISVLSI 2014*.

[26] S. Lin, Y.-B. Kim, and F. Lombardi, "A low leakage 9t sram cell for ultra-low power operation," in *GLSVLSI 2008*.

[27] I. J. Chang, J.-J. Kim, S. P. Park, and K. Roy, "A 32kb 10t subthreshold sram array with bit-interleaving and differential read scheme in 90nm cmos," in *ISSCC 2008 (Digest of Technical Papers)*.

[28] K. Patel, T.-J. K. Liu, and C. J. Spanos, "Gate line edge roughness model for estimation of finfet performance variability," *ED 2009*.

[29] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *ISPASS 2009*.

[30] M., A., M. Goodrum, J., A. Trotter, S. Aksel, T., K. Acton, and Skadron, "Parallelization of particle filter algorithms," in *EAMA workshop 2010*.

[31] "Parboil benchmark suite," http://impact.crhc.illinois.edu/parboil.php.

[32] "Synopsys technology computer-aided design (tcad)," http://www.synopsys.com/tools/tcad.

[33] Z. Li, J. Tan, and X. Fu, "Hybrid cmos-tfet based register files for energy-efficient gpgpus," in *ISQED 2013*.

[34] H. Ghasemi, S. Draper, and N. S. Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *HPCA 2011*.

[35] S. Schuster, "Multiple word/bit line redundancy for semiconductor memories," *JSSC 1987*.

[36] B. Zhai, R. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester, "Energy efficient near-threshold chip multi-processing," in *ISLPED 2007*.

[37] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "A 300mv 494gops/w reconfigurable dual-supply 4-way simd vector processing accelerator in 45nm cmos," in *ISSCC 2009(Digest of Technical Papers)*.

[38] R. Dreslinski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester, "An energy efficient parallel architecture using near threshold operation," in *PACT 2007*.