Power Modeling and Optimization for GPGPUs

By

Zhi Li

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____

Chairperson Dr. Xin Fu

_____

Dr. Gary Minden

_____

Dr. Prasad Kulkarni

Date Defended: 04/08/2013

The Thesis Committee for Zhi Li

certifies that this is the approved version of the following thesis:

Power Modeling and Optimization for GPGPUs

_____

Chairperson Dr. Xin Fu

Date approved:

Abstract

Modern graphics processing units (GPUs) supports tens of thousands of parallel threads and delivers remarkably high computing throughput. General-Purpose computing on GPUs (GPGPUs) is becoming the attractive platform for general-purpose applications that request high computational performance such as scientific computing, financial applications, medical data processing, and so on. However, GPGPUs is facing severe power challenge due to the increasing number of cores placed on a single chip with decreasing feature size.

In order to explore the power optimization techniques in GPGPUs, I first build a power model for GPGPUs, which is able to estimate both dynamic and leakage power of major microarchitecture structures in GPGPUs. I then target on the power-hungry structures (e.g. register file) to explore the energy-efficient GPGPUs. In order to hide the long latency operations, GPGPUs employs the fine-grained multi-threading among numerous active threads, leading to the sizeable register files with massive power consumption. The conventional method to reduce dynamic power consumption is the supply voltage scaling. And the inter-bank tunneling FETs (TFETs) is the promising candidate compared to CMOS for low voltage operations regarding to both leakage and performance. However, always executing at the low voltage will result in significant performance degradation. In this study, I propose the hybrid CMOS-TFET based register file and allocate TFET-based registers to threads whose execution progress can be delayed to some degree to avoid the memory contentions with other threads to reduce both dynamic and leakage power, and

the CMOS-based registers are still used for threads requiring normal execution speed. My experimental results show that the proposed technique achieves 30% energy (including both dynamic and leakage) reduction in register files with negligible performance degradation compared to the baseline case equipped with naive power optimization technique.

Acknowledgements

I would like to thank my advisor Dr. Xin Fu for providing me an opportunity to study abroad and for her guidance, encouragement through my graduate study at University of Kansas.

I would also like to thank Dr. Gary Minden and Dr. Prasad Kulkarni for serving on my committee.

Besides, I want to say thank you to my friend Jingweijia Tan. Thank you for helping me and discussing the problems during my research.

Finally, I would like to thank my family and friends for their encouragement.

Table of Contents

1 Introduction

Graphic Processing Units (GPUs) are programmable parallel architectures primarily designed for graphics processing to exploit the massive parallelism inherent in graphics processing [1]. Because the massively parallel computing capabilities provided by the hardware, GPUs are becoming popular to accelerate the execution of general purpose parallel applications such as scientific computing, financial applications, medical data processing, and so on.

A lot of research has been done to solve the problems of executing general purpose applications on the GPUs such as performance, reliability and so on. Another important issue is power consumption. GPUs is facing severe power challenge due to the increasing number of transistors placed on a single chip with decreasing feature size [1]. For example, GeForce 8800 can consume 280W at its peak performance [2]. Because of the increasing power, researchers begin to explore the power reduction techniques in GPUs. However, we need a power model of GPGPUs first to provide a power evaluation methodology. In 2007, PowerRed is proposed to estimate the power consumption of GPGPUs based on analytical and empirical power models [3]. In 2011, Integrated Power and Performance (IPP) prediction model is established [4]. This model will predict power efficiency for GPUs with different number of cores, and select the optimal number of cores to save energy. Recently, GPUWattch is announced. It is also an architectural level power model modified based on McPAT (a popular CPU power model). These power

models are early research works to model the power of GPGPUs, and not released so not available to the research community. Thus, in my research, I propose an architectural level power model for major microarchitecture structures in GPGPUs, whose methodology is similar to the Wattch (an architectural level power model for CPU) [5].

The proposed power model provides me the infrastructure to explore power optimization in GPGPUs. I then explore the energy optimization techniques for a typical power hungry structure in GPGPUs, i.e., register file.

In GPGPUs, to hide the latency induced by the function unit computation and off-chip memory accesses, GPGPUs employs the fine-grained multi-threading that quickly switches among a large number of simultaneously active threads. As a result, substantial register files are required to keep the register context of each thread. For example, Nvidia Fermi GPUs supports more than 20,000 parallel threads and contains 2MB register files [6]. Accessing such sizeable register files leads to massive power consumption [7-11]. It has been reported that the register files consume 15%-20% of the GPUs stream multiprocessor's power [4]. Effectively optimizing the register files power consumption is critical and the first step towards the energy-efficient GPGPUs design.

Supply voltage scaling is the fundamental technique to reduce the dynamic power consumption, but it is limited by the leakage constraints in CMOS digital circuits. Recently, Inter-bank Tunneling Field Effect Transistors (TFETs) have shown to be the attractive candidates to operate at low supply voltages (e.g. 0.3V) with ultra low leakage and higher frequency than CMOS [12, 13]. However, at higher supply voltage, CMOS

devices are able to achieve much better performance than TFETs. The unique characteristics of CMOS and TFETs at different voltage levels provide great opportunity in GPUs power savings without hurting the performance.

In the GPGPUs applications, all threads in a kernel execute the same code [14], and exhibit similar execution progress in the fine-grained multi-threading environment. When one thread encounters an off-chip memory access, other threads are likely to issue the requests at approximately the same time, leading to severe memory contentions which extend the memory access time. And the pipeline in the GPU stream multiprocessor stalls when all threads stall due to the long-latency memory accesses. It has been found that the performance of numerous GPGPUs workloads are still bounded by the memory resources even modern GPUs provide very high memory bandwidth [15]. In order to alleviate the memory contentions and efficiently utilize the memory bandwidth, threads can run at different paces which effectively avoid the interferences among memory requests. It enables the implementation of the TFET-based registers in GPGPUs for a number of threads so that they can run at a lower frequency without any performance degradation, and meanwhile, both the dynamic and leakage power of the registers reduces substantially. On the other hand, applying TFETs for all registers in the GPGPUs will cause significant performance penalty since many threads still need to execute at high frequency to achieve the high computational throughput. In my research, I propose the hybrid CMOS-TFET based registers in GPUs to obtain optimal energy reduction with negligible performance penalty.

The contributions of this thesis are as follows:

I build a power model for GPGPUs, which is able to estimate both dynamic and leakage power of major microarchitecture structures in GPGPUs.

I observe that threads in GPGPUs workloads can be seriously delayed while executing in the GPGPUs streaming multiprocessors due to the memory access interference with others. Instead of stalling in the pipeline on the occurrence of serious memory contentions, threads can execute at a low speed by using TFET-based registers to postpone their memory requests. It helps to achieve the win-win scenario: preventing the interferences and achieving the attractive power savings.

I propose to build the hybrid TFET-based and CMOS-based register files, and perform the memory contention-aware register allocation. Based on the access latency of previous memory transaction, the thread stall time is predicted for its following memory access, and TFET-based registers are allocated to that thread to postpone its execution progress to the maximum degree without performance loss. By doing this, the utilization of the TFET-based registers is maximized, and the energy consumption is reduced while maintaining the performance.

The evaluation results show that the proposed register allocation technique in the hybrid register design exhibits the strong capability of reducing the register energy consumption (including both dynamic and static energy) by 30% compared to the case with naive power optimization technique (i.e. power gating the unused registers [7]). Especially, it achieves 42% energy reduction (16% dynamic saving and 26% leakage saving) in memory-intensive benchmarks with only 2.5% performance degradation.

2 Background

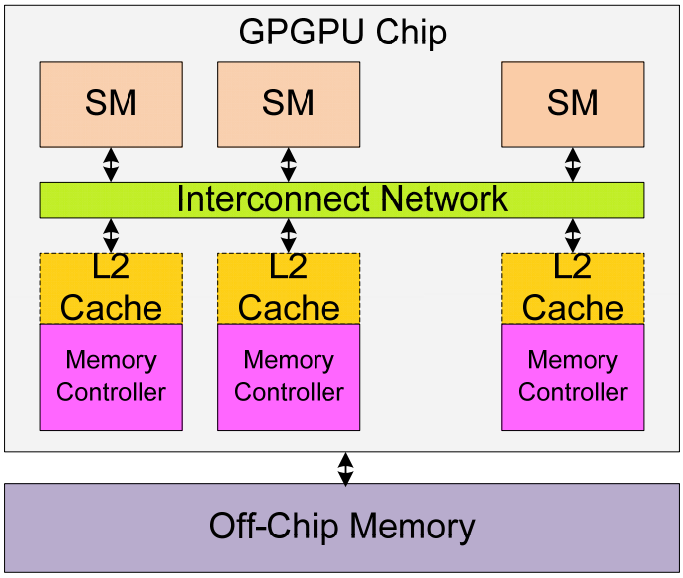


Figure 1 GPGPU architecture

This chapter will introduce the General-Purpose computing on Graphics Processing Units (GPGPUs) Architecture. Figure 1 illustrates an overview of the state-of-art GPUs architecture, which consists of a scalable number of in-order streaming multiprocessors (SMs). SMs can access to multiple on-chip memory controllers via an on-chip interconnection network [14]. GPUs have their own off-chip external memory (e.g. global memory) connected to the on-chip memory controllers. Some high-end GPUs also have L2 cache (shown as dotted line in figure 1).

Figrue 2 shows a detailed microarchitecture of the SMs. It contains the warp scheduler, register file, streaming processors (SP), constant cache, texture cache, and shared memory.

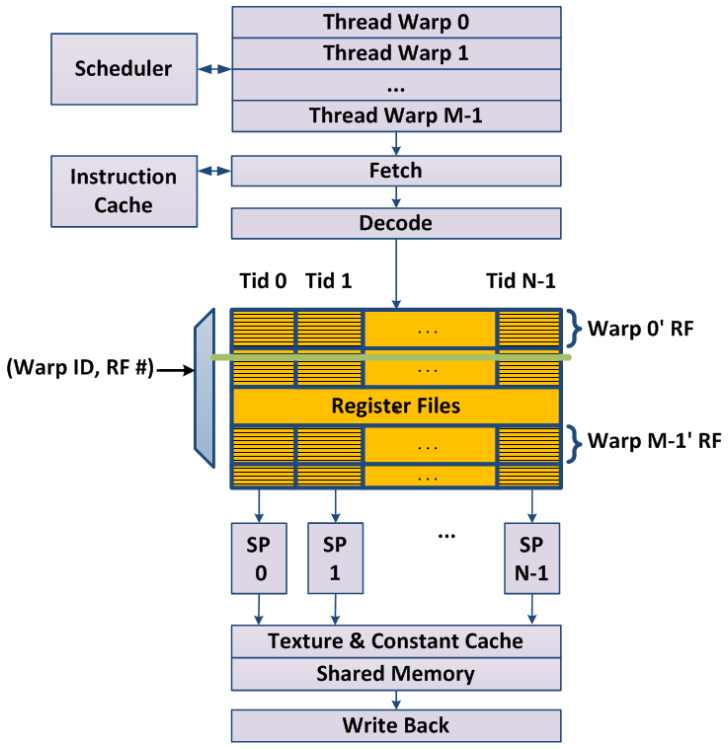


Figure 2 SM microarchitecture

In order to facilitate GPGPU application development, several programming models have been developed by NVIDIA and AMD. For this research, the NVIDIA CUDA programming model is used but some of the basic constructs are similar for most programming models. In CUDA, the GPUs is treated as a co-processor that executes highly-parallel kernel functions launched by the CPU. The kernel is composed of a grid of light-weighted threads; a grid is divided into a set of blocks; each block is composed of hundreds of threads. Threads in the kernel are assigned to the SMs at the granularity of blocks, and threads within a single block communicate via shared memory and synchronize at a barrier if needed. Per-block resources, such as registers, shared memory space, and so on, will not be freed until all threads in the block finish their execution. More than one block can be assigned to a single SM if resources are available.

Threads in the SM execute on the single-program multiple-data (SPMD) model. A number of individual threads (e.g. 32 threads) from the same block are grouped together, called warp. In the pipeline, threads within a warp execute the same instruction but with different data values. As Figure 2 shows, each warp has a dedicated slot in the warp scheduler. At every cycle, a ready warp is selected by the scheduler to feed the pipeline. The instruction is then fetched from the instruction cache based on the PC of issued warp, and further decoded. In the SM, a number of registers are statically allocated to each warp when the block is distributed. All threads in the warp access a number of registers (i.e. the register vector) simultaneously based on the warp ID and the register number, the register values are processed in parallel across the streaming processors (SP). During the memory stage, when one thread has a long latency off-chip memory access, all the threads in this warp will be stalled and can not proceed until the memory request is served.

3 Power Modeling of GPGPUs

As the processing technology continues to scale, more and more transistors are placed in the GPUs chip which result in higher computational capability and also larger power consumption. This provides new opportunities for researchers to explore the power optimization techniques to achieve optimal tradeoff between performance and power. To explore such opportunities, we need a power model of GPGPUs first. Several research works have been done to model power of GPGPUs such as PowerRed, IPP, and GPUWattch. However, all these power models are at the early stage of research and not released. So, in order to explore the optimization between power and performance in GPGPUs, I developed a power model for major microarchitecture structures (streaming multiprocessors) in GPGPUs. And this chapter will present the detailed power model in my study.

3.1 Background on Power

When talking about power in the past days, we just mean the dynamic power consumption, which is caused by switching activity of transistors. And leakage power is negligible so it is ignored. For example, Wattch just model the dynamic power and does not consider leakage. Recently, technology scaling lead to the increase in leakage power, so leakage is becoming more and more important. For example, the 2001 International Technology Roadmap for Semiconductors (ITRS) predicts that leakage may account for up to 50% of total power dissipation in the next several processing technology nodes [16]. Therefore, current power model should model both dynamic power and leakage power.

Caused by the switching of transistors to charge and discharge the load capacitance, dynamic power can be calculated by the following equation:

$$P_d = aCV_{dd}^{\ 2} f \tag{1}$$

Here, the activity factor, a, is a fraction between 0 and 1 indicating how often clock ticks lead to switching of the transistor on average; C is the load capacitance; V_{dd} is the supply voltage; and f is the clock frequency [5].

And leakage power is primarily the result of unwanted subthreshold current in the transistor channel when the transistors are turned off. The leakage power can be estimated using following equation:

$$P_l = KNV_{dd} I_l \tag{2}$$

Here, K is constant that related to technology characteristics; N is the number of transistors; V_{dd} is the supply voltage; I_l is leakage current in a transistor, which can be estimated using equation 3

$$I_l = U_0 \cdot C_{ox} \cdot \frac{W}{L} \cdot e^{b(V_{dd} - V_{dd0})} \cdot v_t^{\ 2} \cdot (1 - e^{\frac{-V_{dd}}{v_t}}) \cdot e^{\frac{-|v_{th}| - V_{off}}{n \cdot v_t}} \tag{3}$$

Here, detailed description of these parameters can be found in [17].

3.2 Power Modeling Methodology

The basic idea to calculate dynamic power of streaming multiprocessors in this model is similar to Wattch. I consider dynamic power of different structure blocks inside each SM,

and calculate the capacitance of each structure first. To calculate capacitance, the technology data is required. Wattch provides the technology data from 800nm to 100nm, while this research targets on 32nm processing technology node. Thus, the processing technology data of Wattch is scaled down to 32nm using the method mentioned in [18]. After the calculation of capacitance of each structure, its dynamic power P_{i_d} can be obtained using equation 1. Notice that we use the same values of activity factors as Wattch for different structure, and our supply voltage is 0.7V and the clock frequency is 1GHz. P_{i_d} is the power of accessing structure i for one time and we calculate the total power of this structure for the total execution duration using following equation:

$$P_{i\_d\_total} = P_{i\_d} N_{i\_access} \tag{4}$$

Here, N_{i_access} is the access count of structure i, which is obtained from the performance simulator GPGPU-sim. The ratio of $P_{i\_d\_total}$ and total execution cycles (obtained from GPGPU-sim) is the average power of this structure block.

$$P_{i\_d\_average} = \frac{P_{i\_d} N_{i\_access}}{N_{total\_cycles}} \tag{5}$$

Then, by summing up average power of all the structures of SMs, we can get average dynamic power consumption of SMs.

$$P_{chip\_d\_average} = \sum_{i} P_{i\_d\_average} \tag{6}$$

In order to estimate the leakage power of SM, HotLeakage is used to estimate leakage for memory structures based on equation 2 and 3, and empirical data are used for all other structures.

Following figure illustrates the overview structure of my power model integrated with the cycle-accurate performance simulator GPGPU-sim.

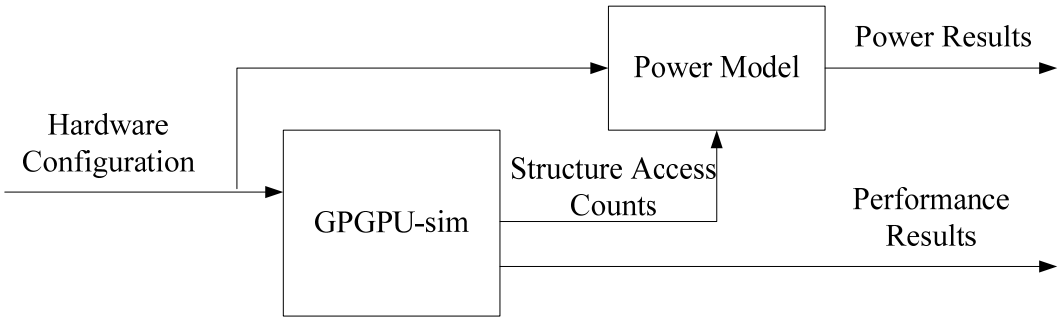


Figure 3 Structure of power model

GPGPU-sim is modified to count the accesses of each hardware structure. At the end of execution, the power model will use the access counts to calculate the average dynamic power of SMs following the equation 4, 5 and 6 based on the input hardware configuration. The power model will also estimate the leakage power for different structures using different methods based on its configuration.

3.3 Dynamic Power Modeling of GPGPUs

In this section, I will introduce dynamic power modeling for GPGPUs. I build the power model of SMs which is the major power consumer in GPGPUs. Figure 2 shows detailed architecture of SMs, and we model the dynamic power consumption of each structure inside SMs. The basic idea to estimate dynamic power of these structures is to estimate the capacitance of each structure first, and then calculate its corresponding dynamic

power using equation 1. This idea is the same with Wattch. Following is the detailed dynamic power calculation for different structures inside SMs.

3.3.1 Fetch, decode and issue logic

In GPGPUs, whenever the fetch logic is accessed, decode and issue logic will be accessed. Thus, they have the same access count number. To simplify the design, we combine the power of these three parts together.

Fetch logic mainly consists of instruction selection logic, fetch window and instruction cache. Therefore, I model the access power of these structures for the fetch logic. In Wattch, to calculate the capacitance of selection logic requires the number of selection entries. For our GPGPU configuration, we have 32 warps on each SM to select, so we set the selection entries as 32. Thus, we can obtain the total capacitance of this structure using the input entries and scaled technology data, then we can get the dynamic power. Fetch window is a regular array structure. According to Wattch, number of rows and columns are required to estimate the capacitance. In our GPGPU configuration, the fetch window has 32 entries and each entry has 32bit (because PC has 32 bit), so we model this window as a regular array structure whose size is 32bit by 32bit. Thus, we can calculate the total capacitance and its power. We should notice that this structure will be accessed twice every time because we need to read the current PC value and then write the new PC value into this structure. Instruction cache power calculation in Wattch utilizes the method from Cacti [28], which is a cache design tool. The required inputs include is number of sets, block size and associativity. Using these input parameters, Cacti will first divide the large cache into small subarrays to optimize the access time of each subarray.

Then, capacitance of decoder, wordline, bitline, senseamplifier, and tagarrays will be estimated for each subarray. Thus, the dynamic power of accessing one subarray can be estimated. Sum up the dynamic power consumption of instruction selection logic, fetch window and instruction cache, we get the dynamic power consumption for fetch logic of each access.

Decode logic includes decoder and instruction buffer. In Wattch, to calculate the decoder power, operation code length is required to estimate the capacitance of the decoder. The operator length of GPGPUs instructions is 32 bit. Thus, we can calculate the capacitance of the decoder and then power consumption. The decoded instruction will be written into the instruction buffer which has 32 entries and 34 columns (32 bit instruction, 2 bits for flags). Instruction buffer belongs to simple array structure, and its power consumption can be calculated like the instruction window power as mentioned above. Besides, the instruction buffer will be accessed twice every access to read the current instruction and write the new instruction.

The decoded instruction in the buffer will be issued by the issue logic, which is in fact a selection logic. We use the same method as instruction selection logic to calculate its power. The entries number is 32, because we have 32 warps to issue. Thus, we can estimate the capacitance and dynamic power.

The sum of the dynamic power of fetch, decode and issue logic can be found in Table1.

3.3.2 Branch unit

In GPGPUs, to solve the branch divergence issue, a branch unit is used to mask the not active thread. Branch unit also belongs to simple array structure, so the number of rows and columns are required to estimate the capacitance. In GPU, each warp in the SM has a mask table which has 32 entries and each entry has 96 bit (32 bit PC, 32 bit previous PC and 32 thread bit). We assume there is only one big table in each SM so it should has 32*32 rows and 96 columns. Then, the corresponding capacitance and power can be estimated.

3.3.3 Register file

To estimate dynamic power of registers, Wattch and Cacti will divide the large register files into small subarrays, which is optimized according to access time. For our design, according to the unique register file access characteristics of GPGPUs, I use the similar register file configuration mentioned in [4, 5]. And each SM has 64KB register file, which is divided into 32 subarrays, and block size of each bank is 64 byte. Because it is direct mapped structure, there are 32 sets in each bank. According to these parameters, I can estimate the decoder, wordline, bitline capacitance and power of accessing each subarray.

3.3.4 Shared memory and caches

The power calculation of shared memory is similar to register file. We should input the shared memory size first. In this study, the size of shared memory is 16KB. It is direct mapped structure and its block size is 4 Byte. Based on these inputs, Wattch and Cacti

will optimize the structure inside the shared memory and calculate the decoder, wordline and bitline capacitance, then dynamic power of accessing such subarray.

We have already talked about the power modeling of the instruction cache, and we use the same method to calculate the access power of constant and texture cache. The size of constant cache is 8KB with 64 Byte block size and its associativity is 2. And size of texture cache is 64KB with 128 byte block size and its associativity is 8.

3.3.5 Execution unit

Similar to Wattch, we use the empirical power data to estimate the integer adder and multiplier, floating point adder and multiplier power in GPGPUs. All these structures are 32bit.

3.3.6 Logic operation

The shift and other logic operation power consumption is considered also. All of them are 32bit operations and the dynamic power is also estimated using empirical data.

3.3.7 Clock power

As pointed out in the Wattch, clock power is a significant source of power consumption. So, I modified the corresponding parameters (such as the pipeline width, pipeline registers and so on) of Wattch to estimate the clock power in GPGPUs.

The following table shows the result of dynamic power consumption of accessing each structure based on the method mentioned above.

Table 1 Dynamic power of different structures

Structures	Power(W)
Fetch, decode and issue logic	0.974
Branch unit	0.287
Register file	0.00718
Shared memory	0.0111
Texture cache	0.0105
Constant cache	0.00436
Integer adder	0.0391
Integer multiplier	0.0603
Floating point adder	0.040
Floating point multiplier	0.120
Logic operation	0.00351
Clock power	27.926

For the power consumptions in above table, we will count access times of each structure (except clock power which is a constant) respectively in our simulator. Then, following the equations 4, 5 and 6, average dynamic power of SMs can be estimated.

3.4 Leakage Power Modeling of GPGPUs

In order to model the leakage power of GPGPUs, I use the open-source code of HotLeakage to calculate the leakage power of regular strutures, such as register file and cache, based on configurations mentioned above. The description of leakage calculation for such structures can be found in the technique report of HotLeakage [17]. The leakage power consumption of other structures is estimated and scaled based on available empirical data. The total leakage power of SMs is around 33W, which is a constant.

3.5 Power Results

Based on the dynamic and leakage power model of SMs discussed above, I run the benchmarks on the modified GPGPU-sim to get the power results. Detailed information of our benchmarks and configuration of GPGPUs can be found in chapter5. Table2 is the original dynamic power data of structures for different benchmarks.

Figure 4 shows the dynamic and leakage power of SMs for each benchmark. And dynamic power is composed of power of different structures. We can find that for different benchmarks, the total power consumption is largely different and the dynamic power consumption of structure blocks is also different between benchmarks. For example, memory intensive benchmarks (such as BFS, HY, NE, NN, NQU, NW, PR and MT) have low streaming multiprocessor dynamic power consumption due to the low computational operations. On the other hand, computation intensive benchmarks (such as CP, CS, and MM) consume a lot dynamic power because of the high computation operations. On average, the SMs dynamic power is around 67W. And dynamic power of register file is more than 10% of the SM dynamic power. And the average of dynamic and leakage power of SMs is around 100W.

Table 2 Dynamic power of structures for different benchmarks (Unit/W)

	Leak	fds	branch	register	shmem	text	const	iadd	imult	fpadd	fpmult	logic	clock	Total
BFS	33	6.5912	0.1839	0.8631	0	0	0	1.1611	0.2392	0	0	0	27.9263	69.9648
BP	33	16.748	0.2851	6.4622	0.232	0	0	5.5169	2.5917	0.4162	3.4884	0.1021	27.9263	96.7687
LPS	33	21.808	0.9358	7.7001	0.9932	0	0	6.9944	2.7283	1.2478	0.7487	0.0014	27.9263	104.0835
PNS	33	8.9499	0.1498	4.1239	0.0109	0	0	1.9083	2.0589	0.8502	0	0.546	27.9263	79.5242
SP	33	17.149	0.6961	8.1148	0.2446	0	0	8.5282	1.4182	1.1061	2.6552	0.4421	27.9263	101.2807
SRAD	33	12.586	0.2592	5.6208	0.3598	0	0	5.6203	1.7142	0.9963	3.3035	0.0229	27.9263	91.4089
HY	33	0.7975	0.0243	0.395	0.04	0	0	0.4385	0.028	0.0007	0.0022	0.0106	27.9263	62.6641
BN	33	14.899	0.6844	6.7156	0.6891	0	0	5.8278	1.495	0.8452	5.0145	0.0482	27.9263	97.1489
SLA	33	16.494	0.3732	3.5226	0.2934	0	0	3.1455	2.4864	0.2118	0	0.222	27.9263	87.6753
64H	33	20.692	0.086	10.4062	1.6641	0	0	12.6284	4.0855	0	0	1.9789	27.9263	112.4669
ST3D	33	17.627	0.1918	9.1719	1.4118	0	0	8.2557	1.2413	6.809	4.2556	0	27.9263	109.8915
LV	33	13.176	0.5261	6.1052	0.1555	0	0	5.9515	1.5695	0	0	0	27.9263	88.41
NE	33	5.0275	0.13	1.1347	0	0	0	0.9077	0.2907	0.2858	0.3832	0	27.9263	69.0859
NN	33	9.0185	0.3193	0.3549	0	0	0	0.5016	0.0417	0.1208	0.371	0.0002	27.9263	71.6564
NQU	33	0.3755	0.0145	0.084	0.0114	0	0	0.0882	0.0117	0	0	0.0063	27.9263	61.5179
NW	33	4.9496	0.13	0.8273	0.0826	0	0	1.0137	0.3996	0	0	0	27.9263	68.3291
PF	33	4.1986	0.0872	2.0399	0.133	0	0	1.7998	0.7728	0	0	0.0179	27.9263	69.9755
PR	33	14.483	0.4255	8.033	1.0297	0	0	14.3032	0.211	0	0	0	27.9263	99.4113
RAY	33	26.519	0.6779	12.5079	0	0	0.26	5.0776	0.4602	5.5406	17.0331	0.1277	27.9263	129.1347
BS	33	24.395	0.0758	13.6681	0	0	0	3.6608	0.0607	5.899	34.4126	0	27.9263	143.0984
CP	33	20.546	0.1447	14.6029	0	0	0.28	3.8156	0.0337	16.5977	32.5357	0	27.9263	149.4792
CS	33	25.67	0.1081	13.3985	2.5191	0	0.91	20.5401	2.0379	8.3735	25.1206	0	27.9263	159.6021
FWT	33	15.021	0.177	8.0965	0.5629	0	0	8.7568	3.2003	2.4726	0.2909	0.3561	27.9263	99.8606
LIB	33	17.247	0.1679	9.5803	0	0	0.07	6.2557	1.0618	2.485	18.1569	0	27.9263	115.9549
MM	33	25.432	0.1343	14.7813	4.9581	0	0	21.0689	0.8489	8.44	25.3201	0.033	27.9263	161.9428
MRIF	33	16.587	0.1047	9.9753	0	0	0.5	5.2411	0.3542	5.5465	22.186	0	27.9263	121.4224
MT	33	4.2997	0.0401	2.0701	0.0329	0	0	1.8625	1.4353	0	0	0.0156	27.9263	70.6825
STO	33	20.093	0.0495	11.206	0.6813	0	0	6.7551	0.2373	0	0	3.8991	27.9263	103.8479

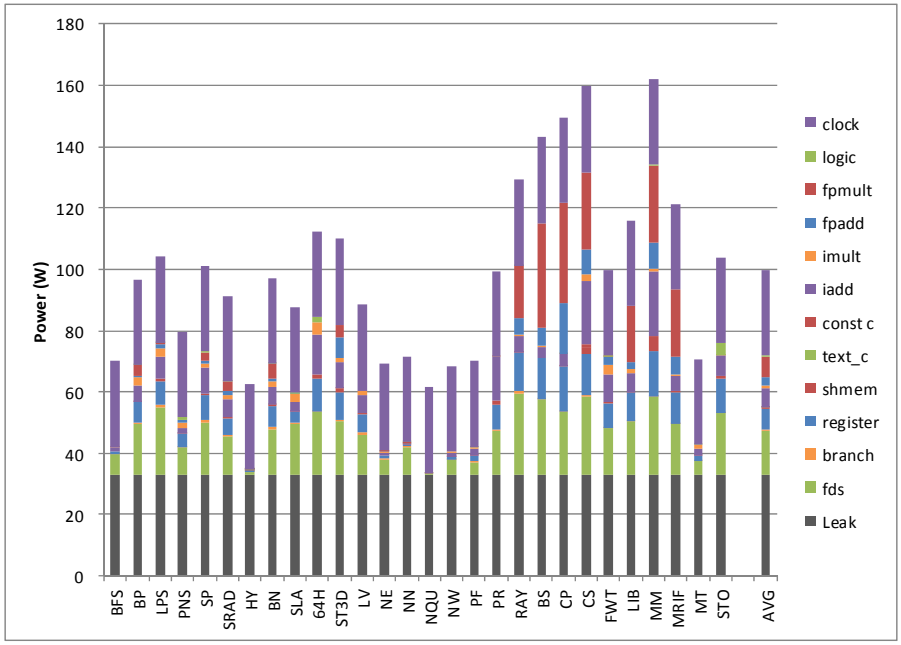


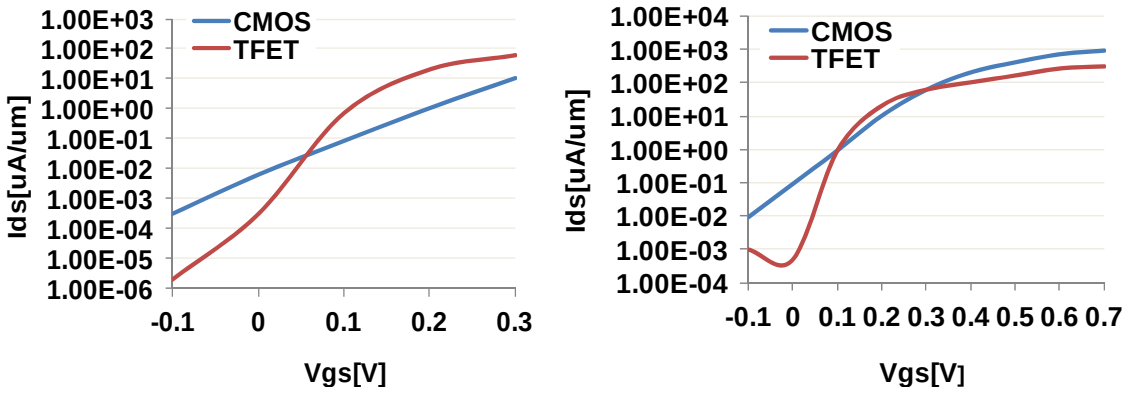
Figure 4 SMs dynamic and leakage power for different benchmarks

4 Hybrid CMOS-TFET based Register Files

This chapter will present our observation, proposed technique and finally the results of our technique to save energy in register file.

4.1 Tunneling Field Effect Transistors (TFETs)

Power consumption is becoming a challenge of the chip design. The conventional method by reducing the supply voltage becomes less effective due to the reduced performance and increased leakage power. The sub-threshold slope of the transistor is the key factor in transistor characteristics, and a steep sub-threshold device can achieve high on current at lower voltage with small leakage current. Traditional CMOS devices are limited to 60mV/decade sub-threshold slope which impede the voltage scaling [19]. While TFETs [12] exhibit sub-60mV/decade sub-threshold slope and achieve higher performance and very low leakage power consumption at low supply voltage compared with CMOS. Figure 5(a) compares the OFF-state leakage current (I_{OFF}) and ON current (I_{ON}) of the two kinds of devices when V_{CC} is 0.3V. As it shows, TFETs are able to obtain much lower leakage current and stronger driven current, therefore, ultra low leakage with high frequency. Thus, TFETs working at low voltage are promising for energy-efficient computing. On the other hand, as Figure 5(b) exhibits, although TFETs are still able to achieve low I_{OFF} at high supply voltage (e.g. 0.7V), CMOS devices have larger driven current and better performance than TFETs.



(a)0.3V (b)0.7V

Fig 5 Characteristics of TFET and CMOS at 0.3V and 0.7V

TFETs have the characteristic of uni-direction conduction which causes a challenge on designing the SRAM storage cell. Recently, many different TFET SRAMs have been explored to overcome this limitation [20-23]. By comparing those designs on several aspects (e.g. frequency, noise margins, power, and area), in this study, we apply the 6T TFET SRAM proposed by Singh et al. [20] to implement the TFET-based register files.

4.2 Observation on Memory Contentions

In GPGPUs, the off-chip memory requests from SMs need to go through the on-chip network routing to certain memory controller and wait there to be served. When numerous requests are issued at similar time by multiple SMs, both on-chip network and memory controllers will be severely congested which significantly increases the memory access time. Unfortunately, such congestion issue occurs frequently in GPUs due to the unique characteristic of the GPGPUs applications: all threads in the kernel across SMs execute the same instructions and proceed at similar rate in the fine-grained multithreading environment. Although there are up to thousands of active threads running in each SM, they are unlikely to fully hide the extremely long-latency memory

transaction caused by the memory contentions. As a result, the SMs suffers long-time pipeline stall. The GPUs memory bandwidth is already considered as one of the resource constraints for many GPGPUs workloads even modern GPUs provide pretty high memory bandwidth [15].

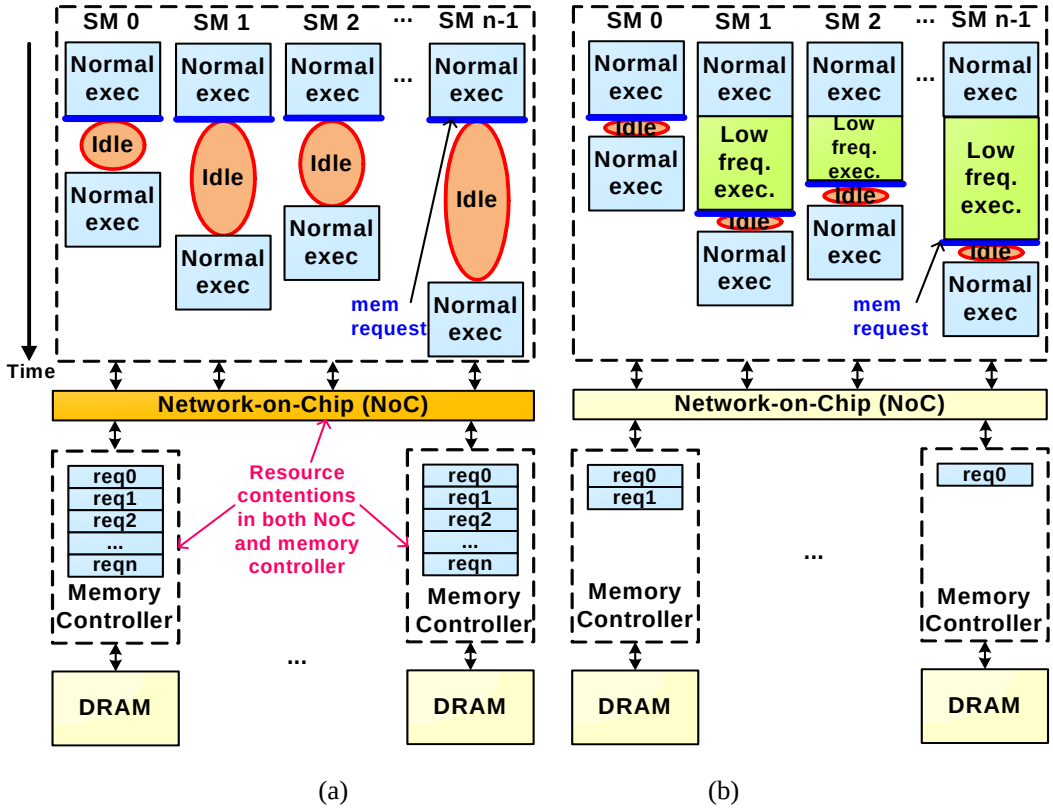


Figure 6 (a) SMs suffer long pipeline stall due to th77e severe memory contentions.

(b) Leveraging TFETs to absorb the long pipeline stall and alleviate memory contentions

Figure 6(a) shows an example of the memory resource contentions among SMs. Several SMs encounter the global memory access instructions and send out memory requests simultaneously. The buffers in network-on-chip (NoC) and memory controllers are quickly filled up by those requests and they have to be served sequentially to access the DRAM buffers (Figure 6(a) takes a snapshot on the NoC and memory controllers).

Therefore, the memory transactions spend longer time to finish, and the pipeline in SMs quickly turns to be idle (highlighted as red circles in Figure 6(a)) since other active threads in the SM will stall at the memory instructions in the near future as well.

The thread throttling mechanism has been proposed recently to alleviate the memory contentions and shrink the pipeline idle time [24]. It dynamically stalls certain threads to restrict the number of concurrent memory tasks and avoid the interferences among memory requests. As can be seen, appropriately slowing down the threads before their memory accesses can even introduce positive effect on performance. Allocating the TFET-based registers to those threads and managing them to execute at low frequency during the register read/write operations provides the perfect approach to control the thread progress. Figure 6(b) demonstrates the example of intelligently leveraging the low frequency operations on TFETs to absorb the pipeline stall time (shown in green rectangles) and meanwhile, separate the memory requests from SMs. As it shows, both NoC and memory controllers have few queued requests, and the off-chip memory access time reduces significantly. More importantly, the benefit of TFETs on reducing both dynamic and leakage energy is effectively explored. Obviously, CMOS-based registers are essential during the normal execution. In this work, I propose the hybrid CMOS-TFET based registers, and use TFET-based registers to delay threads execution speed to the maximal degree so that achieve the goal of maximizing the energy savings without hurting the performance.

4.3 Memory Contention-Aware TFET Register Allocation

In GPGPUs, when launching threads to the SM, a number of registers are statically designated to them according to their resource requirements. The register ID encoded in the instruction is used as the index to the physical register being read/written. In other words, the mapping between the register ID and the physical register is fixed all the time. However, when applying the same mapping mechanism in the hybrid register, the use of the TFET-based registers cannot be managed at the run time.

In this work, a register renaming table is applied to record the physical register number corresponding to the register ID encoded in the instruction. A register renaming stage is inserted into the SMs pipeline following the decode stage. Note that this additional stage does not affect back-to-back instruction latencies. It only induces 1.5% performance overhead based on our evaluation across a large set of GPGPU benchmarks (detailed experimental methodologies are described in chapter 5), which also matches the observation made in [9]. During the register renaming stage, the destination register ID is renamed to a free physical register. The renaming table also provides the information of physical registers to be read according to the source register IDs. Therefore, the thread has the flexibility to map a register to either CMOS or TFET based physical register. A register in the renaming table is released after its last read, and the register lifetime information can be simply obtained by the compiler which indicates the last instruction reading the register. Since threads in a single warp execute the same instruction, they share the same renaming information. The execution of a branch instruction may cause

warp divergence, threads in a diverged warp execute in serial fashion. A physical register will not be released until the last read finishes across all threads in the warp.

The critical challenge in the hybrid register design becomes the runtime CMOS/TFET physical register allocation to the destination register ID in the warp. Aggressively utilizing the TFET registers degrades the performance significantly; on the other hand, too conservatively using the TFET registers fails to achieve the goal of maximizing the registers power savings. Moreover, the TFET utilization among warps needs to be different to well control the warp execution progress and avoid the interferences. As can be seen, it is crucial that the TFET-based register allocation adapts to the memory access pattern of the workloads. For example, randomly or periodically renaming the destination registers to TFET registers can easily hurt the performance as they are blind to the memory accesses. It is highly possible that the TFET registers are improperly used when there are few memory transactions and the high throughput is expected during that period of the workload execution. We propose the MEMory contention-aware TFET Register Allocation (named as MEM_RA as abbreviation) to achieve the optimal power savings with little performance penalty.

Recall that SM supports the SPMD execution model, threads from a warp exhibit the same progress and stall for the same amount of time, therefore, the stall time at warp level is the finest granularity can be considered. The warp stall time due to the off-chip memory access implies the severity of the memory contentions. A long waiting time means the occurrence of serious contentions, and if the memory request from the warp had been postponed by using the TFET registers, such contentions may be removed

successfully. Unfortunately, the waiting time is not available until the request has already been serviced and the contentions already take place. We use the last value prediction mechanism to predict the warp stall time in its next global memory transaction based on the previous memory access latency, and utilize TFET registers to absorb that predicted stall time before the warp sends out its memory request.

Note that the warp has already been slowed down to some degree in previous memory transaction, its following memory request might not interfere with others and it is unnecessary to further delay its progress. This happens in kernels with heavy computation tasks which help to separate the memory transactions and relief the memory contentions. However, the case is different in memory-intensive workloads. Even the warp has been delayed before, its following memory access can get involved with memory transactions from other warps due to the frequently issued memory requests, and further postponing its execution progress is desired.

In order to delay the warp appropriately across various types of workloads, we sample the memory access latency periodically at run time and introduce it into the warp stall time prediction. Eq.7 describes the analytical model to predict the stall cycles (represented as *SC*) of a warp based on its previous memory access latency (represented by *prev_acc*) and the latest sampled memory access latency (represented by *sample_acc*), where

$$SC = \begin{cases} 0, & if\ prev\_acc \le thr\_acc \\ \left\lfloor \frac{sample\_acc}{ref\_acc} \times (prev\_acc - thr\_acc) \right\rfloor, & \\ & if\ sample\_acc < ref\_acc, prev\_acc > thr\_acc \\ prev\_acc - thr\_acc, & if\ sample\_acc \ge ref\_acc, prev\_acc > thr\_acc \end{cases} \quad (7)$$

thr_acc is the threshold latency to determine whether the warp should be delayed in the near future. It is set as the memory access cycles under perfect memory system (e.g. 10 core cycles in our GPU machine configuration). When the *prev_acc* is no longer than the *thr_acc*, it implies that the previous memory transaction does not run into any congestion and the warp proceeds at good speed rate, so no delay is required. *ref_acc* is the referred memory access latency describing the memory access time with moderate resource contentions. When *sample_acc* is longer than *ref_acc*, it implies that the kernel currently exhibits the memory-intensive characteristics, the aggressive delay on the warp execution is preferred. The stall cycle is directly set as the extra waiting time in the previous memory access (i.e. *prev_acc* minus *thr_acc*). To the contrary, a short *sample_acc* compared to *ref_acc* means that the kernel involves heavier computation tasks, the predicted stall time is scaled down according to the ratio of *sample_acc* to *ref_acc*.

Once the stall time is calculated by using the analytical model above, the warp starts to allocate TFET-based registers to the destination register IDs in its following execution. Generally, the read/write time to TFET-based SRAM operating at low supply voltage is as twice as that of the CMOS-based SRAM at normal voltage [25]. The access time to TFET registers is modeled as 2 cycles in our study. In other words, one extra cycle is required to finish the TFET register read/write operation. The TFET register allocation is disabled when the predicted stall time is expected to be fully absorbed. Note that the register read time lasts 2 cycles as long as there is one TFET-based source register. When a warp diverges at a branch instruction, the extra delay is also modeled for all the sequentially executed threads if they use TFET registers. A warp issues multiple memory

transactions when a load instruction is executed and the load requests from threads belonging to that warp fail to get coalesced into fewer memory requests. Those transactions may complete at different time, as a result, the register write back cannot be performed concurrently. Writing values to TFET registers in a load instruction is likely to induce quite long delay which easily makes the warp over-postponed. The TFET register allocation for load instructions are skipped in MEM_RA.

4.4 Implementation

4.4.1 Number of the TFET-Based Registers

Since CMOS and TFET based SRAMs have similar size [26], we set the total amount of hybrid registers in each SM as the same as that (i.e. 16K) in the baseline case with default GPGPUs configuration for the fair comparison. The partition of CMOS and TFET based registers is important to the effectiveness of our proposed MEM_RA mechanism. Fabricating the sizeable TFET registers forces the use of TFET registers when there are insufficient CMOS registers, it reduces power by sacrificing the high computational throughput; while the small TFET registers cannot provide enough TFETs for the energy saving purpose. In the ideal case, the number of CMOS-based registers should perfectly matches their utilization under the impact of MEM_RA, which is largely determined by the warp waiting time during the off-chip memory accesses. The quantity of TFET registers may be more than required in the ideal case, it is better to have idled TFET based instead of CMOS based registers considering the extremely low leakage power consumed by TFET circuits.

Figure 7 shows the percentage of the warp stall time to its total execution time in various types of GPGPUs benchmarks, the detailed experimental setup is described in chapter 5. In the computation-intensive benchmarks (e.g. CP, LPS, MM, and RAY), there are few memory accesses, and the warp stall time is very close to zero. While the numerous memory transactions in the memory-intensive benchmarks (e.g. BFS, BP, MT, NE, and NW) causes much longer warp stall time. On average across all the benchmarks, the stall time is around 22%. In other words, CMOS register should be applied in the remaining 78% of the execution time. Therefore, the CMOS registers are designed to account for 78% of the total registers, and the remains are TFET-based registers. Our 16K hybrid registers are composed of 12.5K CMOS-based and 3.5K TFET-based registers. We also performed detailed sensitivity analysis on varying the size of CMOS registers (e.g. 6K, 10K, and 14K) in the total 16K hybrid design, and found that 12.5K CMOS register is the optimal design regarding to the total energy saving and performance overhead.

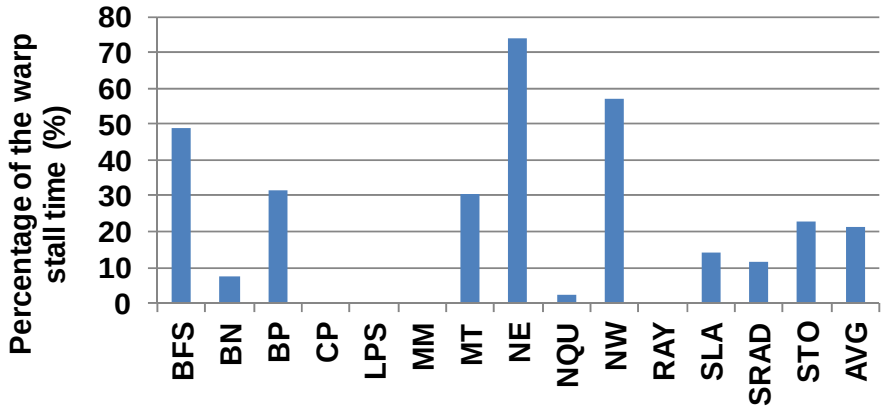


Fig 7 Percentage of warp stall time caused by off-chip memory accesses

In GOGPUs SMs, the per-block resources (e.g. registers, shared memory) are not released until all the threads in the block finish execution, they limit the number of blocks

that can simultaneously run in the SMs. Different per-block resources become the bottleneck for kernels that have different resource requirements. The bottleneck structure is prone to be fully utilized while other structures are usually underutilized. Therefore, a portion of CMOS registers may be free through the entire kernel execution, leading to the considerable leakage power consumption. In [7], the power gating technique has been introduced into GPGPUs SMs to remove leakage. We apply it to power off the unused CMOS registers in SMs. Information such as the maximum number of threads allocated to each SM, and the quantity of physical registers required per thread can be easily obtained during the kernel launch process. Hence, the total register utilization would not exceed the product of those two factors. The requirement on CMOS registers can be estimated by scaling down the total register utilization to 78%, and the power gating is enabled on the remaining idled CMOS registers for a long time until the kernel completes. The energy and time overhead caused by the power gating is negligible with regard to the large power reduction by keeping those registers in the power-gated mode during the entire kernel execution period.

4.4.2 Implementation of MEM_RA

Figure 8 demonstrates the implementation of our proposed memory contention-aware TFET-based register allocation in the hybrid register design. A counter is attached to each warp slot in the warp scheduler. When a warp encounters an off-chip memory access, its counter is re-set as zero and starts the auto increase every cycle to record the memory access latency. Upon the completeness of the memory transaction, the cycle number stored in the counter is sent to an ALU for the warp stall time prediction, meanwhile, the

sampled memory access cycles with the static information (i.e. threshold latency, referred access latency) also input to the ALU. The output is written back to the counter, it will be read when the warp enters into the pipeline, and a larger-than-one value in the counter implies the necessity of writing to the TFET-based register. In [9], Gebhart et al. found that 70% of the register values are read only once in GPGPUs workloads. It implies that most TFET register values are read once, therefore, renaming the destination register to the TFET register usually causes 2-cycle extra delay: one additional cycle during the value write back, and another one when it is read by a subsequent instruction. As can be seen, one TFET register allocation takes two cycles of the warp stall time in most cases. And the counter value will decrease by two upon a successful TFET register allocation. Note that the counter decrease is just used to estimate the possible delay to the warp when renaming to the TFET registers. For TFET registers being read multiple times, the warp stall time will be taken more than two cycles. Moreover, the counter auto-increases occurs at warp waiting time while its value decrease is performed at the normal execution time, there is no overlap between the counter auto-increase and decrease processes.

Considering that the ALU is used once a warp completes a memory instruction, and the major computation in it is division (as shown in Eq.7.) lasting for tens of cycles, we set the referred access latency as 2 to power of n and translate the division into logical shift. It will operate based on the product of the sampled memory access latency and the previous stall time. We performed the detailed sensitivity analysis on the referred access latency, and found that MEM_RA achieves optimal trade-off between power and performance when setting it as 2^7=128 cycles. Note that the warp stall time estimation

occurs in parallel with the write back stage, it does not introduce any extra delay to the critical path in the pipeline.

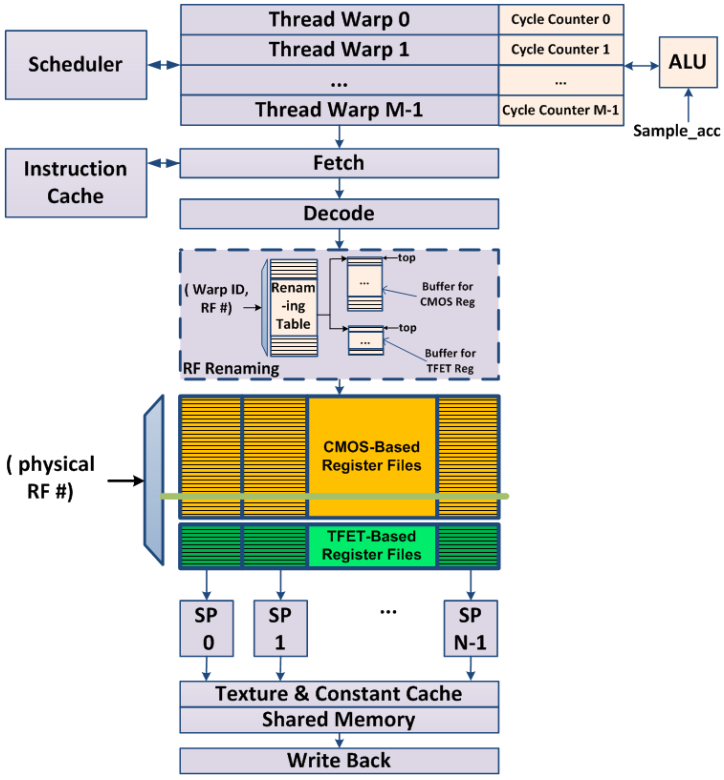


Fig 8 Memory contention-aware TFET-based register allocation

As Figure 8 shows, register files are partitioned into CMOS based and TFET based registers, and each physical register vector has a unique identification number. Two power supply lines are used to support the high (low) voltage operations on CMOS (TFETs) registers. The register renaming stage is added into the SM pipeline (shown as the dotted rectangle), during which the register renaming table is accessed. It is indexed by the warp ID and register number encoded in the instruction, and each entry holds the corresponding physical register vector number which will be used for register access in the following stage. Two FIFO buffers are attached to the renaming table to keep the released CMOS and TFET register vectors, respectively. The top register in each buffer is

consumed for the renaming, while the bottom is filled by the newly released register. In the case that a CMOS register is requested while the buffer for CMOS registers is empty, the buffer for TFET registers will provide a free TFET register instead, and vice versa. Note that there is always at least one free CMOS/TFET register available for renaming since the required resources have already been well estimated when the block is assigned to the SMs.

4.4.3 Hardware and Power Overhead

The major hardware added into the SMs is the register renaming pipeline stage including the register renaming table, two buffers for the released CMOS and TFET register vectors, and some simple combinational logics. In order to keep the renaming information for all physical registers, the number of entries in the renaming table is equal to the amount of register vectors which is 512 in our default GPU configuration. Similarly, the total size of the two buffers is 512 as well. Each entry in those three structures contains 9 bits. The hardware in the renaming stage causes around 2% area overhead to the register files in the SMs. In addition, to predict the warp stall time, thirty-two 11-bit counters (we set the maximum memory access time as 2048 cycles), and the unit performing simple integer arithmetic and logic operations are added in the SMs. The overall hardware overhead to the SM register files is 3%. We develop the power model (including both dynamic and leakage power) for the added hardware, and find that it induces around 2.9% power overhead to the register files by running a large set of GPGPUs benchmarks.

4.5 Experimental Methodology

I implement the MEM_RA technique on the cycle-accurate, open-source, and publicly available simulator GPGPU-sim [27] to obtain the GPGPUs performance statistics. I use the power model of CMOS-based register file built in chapter 2 and build another power model for TFET-based register file using similar method [28]. This model of TFET is based on technology data for TFETs, and it is from the research group of Dr. Narayanan at Pennsylvania State University. We set the high supply voltage as 0.7V and low supply voltage as 0.3V. The read/write times to CMOS- and TFET-based registers and the total execution time are collected from the modified GPGPU-sim to evaluate both RF dynamic and leakage power consumption. Our power estimation is consistent with previous studies [4, 8, 9].

Our baseline GPGPUs configuration is set as follows: there are 28 SMs in the GPU, SM pipeline width is 32, warp size is 32, each SM supports 1024 threads and 8 blocks at most, each SM contains 16K 32-bit registers, 16KB shared memory, 8KB constant cache, and 64KB texture cache, the warp scheduler applies the round robin scheduling policy, the immediate post-dominator reconvergence [29] is used to handle the branch divergences; the GPU includes 8 DRAM controllers, each controller has a 32-entry input buffer, and applies out-of-order first-ready first-come first-serve scheduling policy [27]; the interconnect topologies is Mesh, and the dimension order routing algorithm is used in the interconnect, the interconnect router contains two virtual channels, and flit size is 16B.

We collect a large set of available GPGPUs workloads from Nvidia CUDA SDK [30], Rodinia Benchmark [31], Parboil Benchmark [32] and some third party applications. The

workloads show significant diversity according to their kernel characteristics, branch divergence characteristics, memory access patterns, and so on. Following is the description of benchmarks used in this research. Breadth-First Search (BFS) is a graph algorithm, which performs breadth-first search on a graph; Binomial Options (BN) is a numerical method for the valuation of options; Back Propagation (BP) is a method of training artificial neural networks; Columbic Potential (CP) is useful in the field of molecular dynamics; 3D Laplace Sover (LPS) uses Jacobi iterations for a finance analysis; Matrix Multiplication (MM) is a algorithm to calculate the multiplication of matrix as indicated by the name; Matrix Transpose (MT) is method to calculate the transpose of matrix as indicated by the name; Nearest Neighbor (NE) is a algorithm in pattern recognition; N-Queen Solver (NQU) solves a puzzle problem by searching all possible solutions; Needleman Wunsch (NW) is a sequence alignment algorithm for protein or nucleotide sequences; Ray Tracing (RAY) is a graphics algorithm for generating high visual realism image; Scan of Large Arrays (SLA) is an implementation of parallel prefix sum (also known as “scan”) for arbitrary-sized arrays, given an array of numbers, scan computes a new array in which each element is the sum of all the elements before it in the input array; Speckle Reducing Anisotropic Diffusion (SRAD) is a method for image processing; StoreGPU (STO) is an algorithm to accelerate hashing-based library primitives designed for middleware.

4.6 Results

In order to justify the effectiveness of MEM_RA, I compare it with several power reduction techniques. The baseline case studied in this thesis is employing only

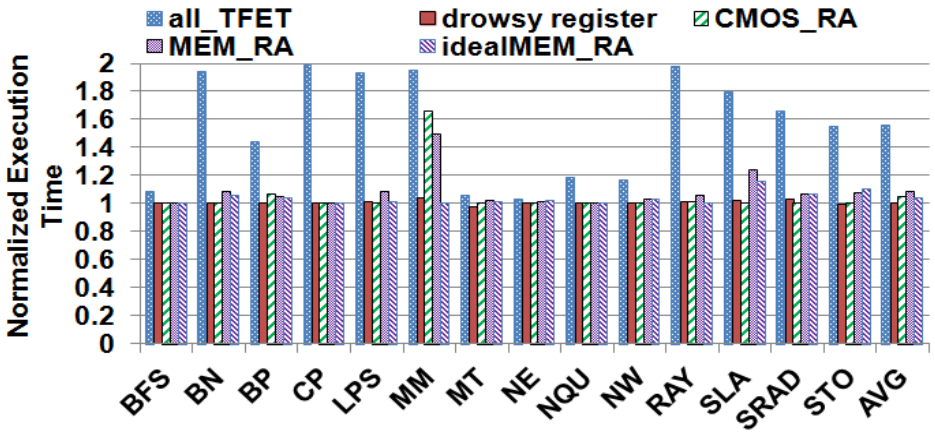
CMOS-based registers and power gating the unused registers during the kernel execution. Another naïve mechanism for power saving is simply applying TFETs to all SM registers, it is named as all_TFET. In previous work, the drowsy cache has been proposed to reduce the cache leakage power [33]. Similarly, registers belonging to a warp can be put into the sleep mode when the warp stalls in the pipeline, but it takes couple of cycles to wake them up for further accesses. We also investigate the effect of drowsy register from the performance and power perspectives. In the hybrid register design, the long access time to TFET registers may largely degrade the performance when they are randomly used. A straightforward technique to maintain performance is to avoid the allocation of TFET registers if possible. In other words, the CMOS register is selected for renaming as long as there is any one free. We name this technique as CMOS_RA, it is applied on the hybrid 12.5K CMOS registers and 3.5K TFET registers. Note that the power gating technique is integrated into drowsy register and CMOS_RA, respectively, for the fair comparison. Since TFET has extremely low leakage power, the power gating is not triggered in all_TFET mechanism.

As discussed previously, ideally, the size of CMOS registers would exactly match their usage. I further investigate the effectiveness of MEM_RA when the CMOS registers size is set ideally, called idealMEM_RA. (I name the MEM_RA using 12.5K CMOS and 3.5K TFET registers as MEM_RA for short.) Since benchmarks exhibit different memory access patterns, their requirements on CMOS registers vary greatly. Although designers rarely fabricate a GPU with certain number of CMOS(TFET) registers to specifically satisfy a single benchmark’s requirement, the results of idealMEM_RA provide a more

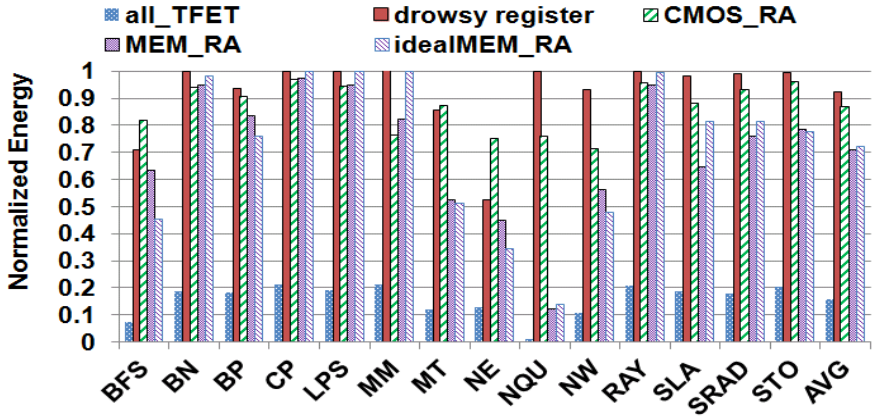
accurate evaluation on the capability of MEM_RA on power optimizations while maintaining the performance.

Figures 9 describes (a) the execution time and (b) the overall energy when running the investigated benchmarks under the impact of several power reduction techniques described above. The results are normalized to the baseline case. Note that the performance and energy overhead caused by each technique is also included in the results. As Figure 9(a) shows, all_TFET hurts the GPU performance significantly, the execution time is almost doubled in several benchmarks (e.g. BN, CP, MM, and RAY). On average, all_TFET degrades the performance by 56%. Although it reduces the energy consumption significantly (total energy decreases to 16% as shown in Figure 9(b)), it is not worth to scarify such large portion of throughput to achieve the low energy consumption. Interestingly, the kernel execution time under drowsy register mechanism remains the same although there is time overhead to wake up registers staying in the sleep mode, because the wake up time is trivial with regard to the hundred-cycle long memory access. Moreover, the energy reduction achieved by drowsy register is small, only around 7%.

CMOS_RA is performance friendly which causes 5% performance penalty on average. Because it uses the CMOS registers in majority of the time, and there is no performance loss when the benchmark needs less than 12.5K registers. However, the performance penalty is high for benchmarks requesting a large amount of registers, as TFET registers are consumed in that case. For example, the execution time for MM increases 65% under CMOS_RA because the RF utilization in that benchmark is 100%.



(a)



(b)

Fig 9 (a) normalized execution time (b) normalized energy consumption

As Figure 9 shows, the energy reduction under CMOS_RA is 13%, while MEM_RA is able to achieve 30% energy savings with similar performance loss (i.e. 8%). Such 30% energy reduction contributes to around 5% energy savings to the entire SM, which is already considered as the noticeable energy optimization as discussed in [9]. Different from CMOS_RA, MEM_RA intelligently migrates the resource usage from CMOS to TFET registers which reduces the total dynamic energy, and meanwhile, the extra access delay in TFETs absorbs the warp waiting time and prevents the interferences among memory requests which minimizes the impact on performance. Especially for the

memory-intensive benchmarks, such as the BFS, BP, MT, NE, and NW, MEM_RA generally reduces the power by 42% with only 2.5% performance loss.

One may notice that MEM_RA introduces the long execution time in MM as well. Because MM fully utilizes the RF resources and contains quite few memory accesses to trigger the memory-contention aware TFET register allocation. As Figure 9(a) demonstrates, the performance of MM maintains the same under idealMEM_RA since the GPU will be equipped with all CMOS registers if running such type of benchmarks, and it cannot reduce the energy. On average, idealMEM_RA slightly outperforms MEM_RA on performance but meanwhile, obtains less energy savings. In summary, MEM_RA successfully explores the energy-efficient GPGPUs and its effectiveness is quite close to that of idealMEM_RA.

The performance degradation in SLA is noticeable under MEM_RA and idealMEM_RA. Because they use the last memory access latency to predict the warp waiting time and enable the TFET register allocation correspondingly, the prediction accuracy is affected when the next memory access pattern differs greatly from the last one. As a result, the TFET registers are excessive utilized which hurts the performance. Generally, the last value prediction mechanism achieves pretty high accuracy for most benchmarks and helps MEM_RA to minimize the performance penalty.

I further split the normalized overall energy obtained by MEM_RA into the dynamic and leakage portions and present them in Figure 10, the energy partition under the baseline case and CMOS_RA is also included in the figure. As it shows, CMOS_RA can barely

optimize the dynamic power since the CMOS register are frequently accessed. MEM_RA exhibits strong capability in reducing not only leakage but also dynamic energy. On average, the dynamic energy reduction compared to the baseline case is 10%, while the leakage decreases 20%. In addition, the dynamic (leakage) energy savings in memory-intensive benchmarks is 16% (26%).

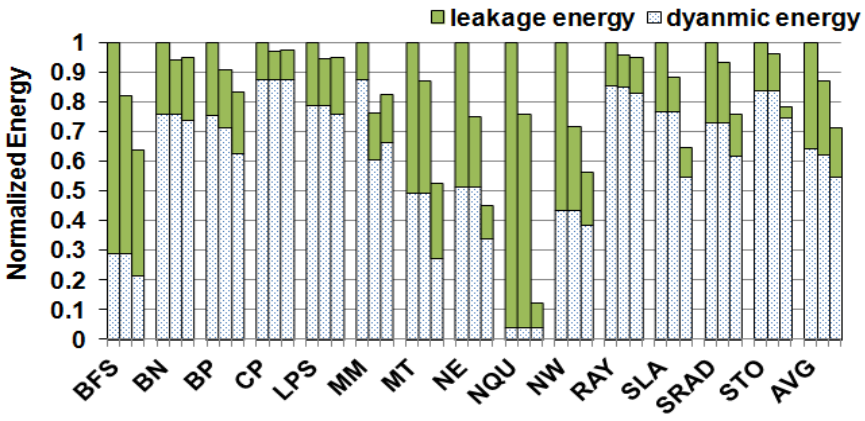


Fig 10 Dynamic and leakage energy consumptions

5 Related Work

There are several studies on building the power model of GPGPUs architecture. In 2007, PowerRed, a modular architectural level power model, is proposed to estimate the power consumption of GPUs using analytical and empirical based models. And they proposed two power optimization techniques to demonstrate the utility of their power model. In 2010, Integrated GPU Power and Performance model (IPP) is established to save energy by selecting the optimal number of cores based on the predicted power efficiency. Recently McPAT, a popular CPU power model, is modified and extended to model the power of GPUs architecture and this modified power model, GPUWattch, is driven by cycle-accurate performance simulator GPGPU-sim to estimate the power consumption of GPGPUs.

There have been several studies on building hybrid storage-cell based structure and furthermore, heterogeneous multi-core processors based on CMOS and TFET to achieve the good trade-off between performance and power [25, 34, 35, 36]. For instance, Narayanan et al. [25] developed the hybrid cache architecture that uses a mix of TFET and the non-volatile memory. Swaminathan et al. [34] proposed to replace some of the CMOS cores with TFET alternatives, and dynamically migrate threads between CMOS and TFET cores to achieve significant energy savings with negligible performance loss. We build the hybrid registers in GPGPUs and leverage its unique characteristics to fully explore the benefit of TFETs for the energy-efficient GPGPUs design.

Many methodologies have been proposed recently to reduce the GPGPUs registers dynamic power. Gebhart et al. [9] proposed register file caching and two-level thread scheduler to reduce the number of reads and writes to the large main register file and save its dynamic energy. The authors further extended their work to the compiler level and explored register allocation algorithms that are targeted on improving register energy efficiency [10]. Yu et al. integrated embedded DRAM and SRAM cells to reduce area and energy [8]. In addition, several works have been done on GPGPUs register leakage power optimization. Chu et al. [11] explored the fine granularity clock gating scheme for registers. Wang et al. [7] adopted the power gating technique at architecture level for leakage reduction on GPGPUs. Our technique targets on both dynamic and leakage savings and it is orthogonal to the techniques discussed above.

6 Future work

There are three major works we should continue with in the future.

First, continue with the building of the power model for GPGPUs. Currently, detailed dynamic power model for GPGPUs SMs is established, however accurate power model for interconnect and memory controller is necessary to estimate the dynamic power of the entire chip. And the power consumption of the off-chip DRAM memory should be integrated in our model. Finally, we should build more accurate leakage power model for all these structures.

Second, based on the power model, DVFS can be implemented to optimize the performance and energy consumption. For example, the opportunity of memory contention observed in this study can be utilized to improve performance and also save energy by migrating the power between GPGPUs SMs and DRAM. When the GPUs SMs encounter long off-chip memory access, we can move the power from GPGPUs SMs to DRAM to speed up the execution, vice versa. Thus, we can improve the performance and save energy by reducing execution time to reduce the leakage power.

Third, based on the power model, we can explore performance optimization under power constraints. We can first use linear programming or other methods to optimize the performance and get the optimal power for different structures under the power constraint. Then, to control the actual power consumption of each structure to stay at the assigned power level, simple control theory methods can be implemented. Thus we can get the optimal performance under given power constraint.

7 Conclusions

Power consumption of modern general-purpose computing on graphics processing units (GPGPUs) is becoming more and more important. To enable the optimization of the power consumption in GPGPUs, I build a power model of GPGPUs, which is able to estimate both dynamic and leakage power of major microarchitecture structures in GPGPUs. I then target on the power reduction in register files which is a power-hungry structures.

GPGPUs employs the fine-grained multi-threading among numerous active threads which leads to the large register files consuming massive dynamic and leakage power. Exploring the optimal power savings in register files become the critical and first step towards the energy-efficient GPGPUs design. The conventional method to reduce dynamic power is to scale down the supply voltage which causes substantial leakage in CMOS circuits. The inter-bank tunneling FETs (TFETs) are the promising candidates for low voltage operations regarding to both leakage and performance. However, always executing at the low voltage (so that low frequency) will result in significant performance degradation. In this thesis, I propose the hybrid CMOS-TFET based register files. I leverage the unique characteristics of GPUs during the off-chip memory accesses, and explore the memory contention-aware TFET register allocation (MEM_RA) to make use of TFET registers in alleviating the memory contentions, and meanwhile gaining the attractive energy optimization. Experiment results show that MEM_RA obtains 30% energy (including both dynamic and leakage) reduction in register files compared to the

baseline case with power gating technique. Especially, it achieves 42% energy savings in memory-intensive benchmarks with only 2.5% performance loss.

References

[1] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. IEEE Micro, 31:7–17, September 2011.

[2] “GeForce 8800 & NVIDIA CUDA: a new architecture for computing on the GPU,” www.gpgpu.org

[3] K. Ramani, A. Ibrahim, and D. Shimizu. PowerRed: A Flexible Modeling Framework for Power Efficiency Exploration in GPUs. In Worskshop on GPGPU, 2007.

[4] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In Proceedings of ISCA, 2010

[5] D. Brooks, V. Tiwari and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In Proceedings of ISCA, 2000.

[6] D. Kanter. Inside Fermi: Nvidia’s HPC Push, 2009. http://www.realworldtech.com/page.cfm?ArticleID=RWT093009110932.

[7] P. Wang, C. Yang, Y. Chen, and Y. Cheng. Power Gating Strageties on GPUs, ACM Transaction on Architecture and Code Optimization (TACO), Volume 8 Issue 3, October 2011.

[8] W. Yu, R. Huang, S. Xu, S.-E. Wang, E. Kan, and G. E. Suh. SRAM-DRAM Hybrid Memory with Applications to Efficient Register Files in Fine-Grained Multi-Threading. In Proceedings of ISCA, 2011.

[9] 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 Proceedings of ISCA, 2011.

[10] M. Gebhart, S. W. Keckler, and W. J. Dally. A Compile-Time Managed Multi-Level register File hierarchy. In Proceedings of MICRO, 2011.

[11] S. Chu, C. Hsiao, and C. Hsieh. An Energy-Efficient Unified Register File for Mobile GPUs. In Proceedings of IFIP 9th International Conference on Embedded and Ubiquitous Computing, October 2011.

[12] S. Mookerjea, D. Mohata, R. Krishnan, J. Singh, A. Vallett, A. Ali, T. Mayer, V. Narayanan, D. Schlom, A. Liu, and S. Datta. Experimental Demonstration of 100nm Channel Length In0.53ga0.47as-based Vertical Inter-Band Tunnel Field Effect Transistors (TFETs) for Ultra Low-Power Logic and Sram Applications. In Proc. IEEE Int. Electron Devices Meeting (IEDM), 2009, pp. 1–3.

[13] N. N. Mojumder and K. Roy. Band-to-Band Tunneling Ballistic Nanowire FET: Circuit-Compatible Device Modeling and Design of Ultra-Low-Power Digital Circuits and Memories. IEEE Transactions On Electron Devices, vol. 56, pp. 2193–2201, 2009.

[14] NVIDIA. CUDA Programming Guide Version 3.0., Nvidia Corporation, 2010.

[15] V. Sathish, M. J. Schulte, and N. S. Kim. Lossless and Lossy Memory I/O Link Compression for Improving Performance of GPGPU Workloads, In Proceedings of PACT, 2012.

[16] SIA. International Technology Roadmap for Semiconductors, 2001.

[17] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. In Tech. Rep, Department of Computer Sciences, University of Virginia, 2003.

[18] J. Meng, C. Chen, A. K. Coskun, and A. Joshi. Run-Time Energy Management of Manycore Systems Through Reconfigurable Interconnects. In Proceedings of GLSVLSI, 2011.

[19] Y. Taur and T. H. Ning. Fundamentals of Modern VLSI Devices. Cambridge University Press, 2009.

[20] J. Singh, K. Ramakrishnan, S. Mookerjea, S. Datta, N. Vijaykrishman and D. Pradhan. A Novel Si Tunnel FET based SRAM Design for Ultra Low-Power 0.3V Vdd Applications. In Proceedings of ASPDAC, 2010.

[21] V. Saripalli, S. Datta, V. Narayanan and J. P. Kulkarni. Variation-Tolerant Ultra Low-Power Heterojunction Tunnel FET SRAM Design. In Proceedings of International Symposium on Nanoscal Architectures, 2011.

[22] D. Kim, Y. Lee, J. Cai, I. Lauer and L. Chang, S. J. Koester, D. Sylvester and D. Blaauw. Low Power Circuit Design Based on Heterojunction Tunneling Transistors (HETTs). In Proceedings of ISLPED, 2009.

[23] X. Yang and K. Mohanram. Robust 6T Si tunneling transistor SRAM design. In proceedings of DATE, 2011.

[24] H. Cheng, C. Lin, J. Li, and C. Yang. Memory Latency Reduction via Thread Throttling. In Proceedings of MICRO, 2010.

[25] V. Narayanan, V. Saripalli, K. Swaminathan, R. Mukundrajan, G. Sun, Y. Xie, and S. Datta. Enabling Architectual Innovations Using Non-Volatile Memory. In Proceedings of GLSVLSI, 2011.

[26] A. Pal, A. B. Sachid, H. Gossner, and V. R. Rao. Insights Into the Design and Optimization of Tunnel-FET Devices and Circuits. IEEE Trans on Electron Devices, vol. 58, NO. 4, April 2011.

[27] A. Bakhoda, G.L. Yuan, W. W. L. Fung, H. Wong, Tor M. Aamodt, Analyzing CUDA Workloads Using a Detailed GPU Simulator, In Proceedings of ISPASS, 2009.

[28] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.1. HP Labs, Tech. Rep. 2008.

[29] S. S.Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmanns, 1997.

[30] http://www.nvidia.com/object/cuda_sdks.html

[31] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing, In Proceedings of IISWC, 2009.

[32] Parboil Benchmark suite. URL: http://impact.crhc.illinois.edu/ parboil.php

[33] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In Proceedings of ISCA, 2002.

[34] K. Swaminathan, E. Kultursay, V. Saripalli, V. Narayanan, M. Kandemir, and S. Datta. Improving Energy Efficienty of Multi-Threaded Applications Using Heterogeneous CMOS-TFET Multicores. In Proceedings of ISLPED, 2011.

[35] V. Saripalli, A. Mishra, S. Datta and V. Narayanan. An Energy-Efficient Heterogeneous CMP based on Hybrid TFET-CMOS Cores. In Proceedings of DAC, 2011.

[36] V. Saripalli, G. Sun, A. Misha, Y. Xie, S. Datta and V. Narayanan. Exploiting Heterogeneity for Energy Efficiency in Chip Multiprocessors. IEEE Trans on Emerging and Selected Topics in Circuits and Systems, vol. 1, pp. 109-119, June 2011.