An Accurate Power Model for GPU Processors

Qiyao Xie, Tian Huang, Zhihai Zou, Liang Xia, Yongxin Zhu, Jiang Jiang
School of Microelectronics
Shanghai Jiao Tong University
Shanghai, China
{xieqiyao, huangtian, zouzhihai, xialiang, zhuyongxin, jiangjiang}@ic.sjtu.edu.cn

***Abstract*—Albeit general purpose graphics processing unit (GPU) processor has gained strong momentums in high performance computing domain recently, power consumption stays as the critical constraint in GPU architectures. Among many efforts dedicated to power reduction, most existing power analytical models can achieve sufficient estimate accuracy for a new GPU architecture only after its layout or floor plan is finalized, when it is usually too late for software designers working on the new GPU architecture.**

This paper presents an accurate power model based on GPU native instructions to analyze and estimate the power consumption at an architecture level. Taking the latest FERMI GPU architecture as an illustrative example to apply our methods, our model is comprised of two parts, i.e. the computing section and the memory section. With an integrated view of the GPU architecture, our model is able to estimate the power consumption for the GPU architecture under a series of workloads with deviations less than 15%. To the best of our knowledge, our model should outperform all existing analytical GPU power models in terms of accuracy.

***Index Terms*—GPU architecture, native instructions, power model.**

I. INTRODUCTION

In recent years, many supercomputer system architects start to exploit the advantages of heterogeneous architectures comprising of generic CPU and GPU processors due to strong performance of GPU processors. But the power consumption of GPU also grows tremendously and thus occupies vast proportions of total power consumption of a computing system. Improving power efficiency of GPU processors consequently becomes a major research issue.

Thanks to the invention of next generation GPU architecture Fermi [1], GPU processors are enriched with new functionality features and computing power based on complicated structures. The transistor count of Fermi chips adds up to 3 billion as of GF100 processor, and the corresponding power increases dramatically even if the semiconductor process scales to 45nm. Optimizing power efficiency for GPU applications hence turns out to be a big challenge for application developers. Luckily, there are some high-level language tools such as CUDA[11][14] and OpenCL[12][15], with which we can evaluate the impacts of the power consumption on the performance of the application benchmarks on a physical processor.

In order to accurately project the power consumption on varieties of architecture candidates, in this paper, we propose an accurate analytical power model at an architecture level with architectural and application parameters as inputs. With our model, GPU application developers and system architects can easily and efficiently manage the power consumption for GPU applications without physical layout and implementation. The model is also beneficial for architects to decide tradeoffs for future GPU processors with early stage architectural level power estimations.

Though there have been a large body of research work on power analysis on CPU processors, very few analytical results on power consumption of GPU processors are available. In [7], Ma et al. relied on recorded power consumption readings, runtime workload signals, and performance data to build a power model, but the model is not integrated and the result is not accurate enough. In [9], Nagasaka et al. analyzed the application power consumption on GTX280 using the statistics about memory/cache/computing units. In [8], S. Hong and H. Kim proposed an outstanding power model for GTX280 GPU processor, which appears to be more accurate than that in [9]. However, GTX280 architecture is much simpler than Fermi and their analyses about the computing units appear to be insufficient. Hence the model would not be applicable to the next generation GPU architecture.

In this work, we present a novel scheme for analyzing and modeling the power consumption of GPU. Based on the recorded power consumption indicated in the profiling results of native instructions[6] executing on GPU processors, we build an energy model that is capable of estimating the power consumption of a runtime kernel (kernel in this paper refers to the kernel run on GPU). To the best of our knowledge, few people modeled the relationship between the power estimation and the native instructions on GPU processors. Our model for new generation GPU architecture should be an important reference to other researchers.

The rest of the paper is organized as follows: In Section 2, we introduce the principles of the energy estimate model. Then, every part of the power model will be explained in Section 3. In Section 4, the accuracy and applicability of the model will be verified by a set of benchmarks. In Section 5, we have a summary and describe our future work.

II. THE METHOD OF GPU ENERGY MODEL

A. The classic method of power model

Architectural level power analyses have been studied for more than ten years, but were only focused on the CPU. They present some power estimation frameworks based on some hardware performance counters.

Bellosa was one of the first advocates of using hardware performance counters as links to processor power consumption. In [2], Bellosa demonstrates a high-level black-box model based on CPU performance counters. He used performance counters, to identify correlations between certain processor events, such as retired floating point operations, and energy consumption for an Intel Pentium II processor.

Brooks et al. provided a power analysis and optimization framework, Wattch [3]. Isci et al. later refined the framework, and then measured and modeled the Intel Pentium4 processor [4]. In [4], based on the hardware counters, total power is calculated in classic Equ. (1) and (2).

$$TotalPower = \sum_{i=1}^{n} Power_i + IdlePower \quad (1)$$

$$Power_i = AccessRate_i * ArchitecturalScaling_i * MaxPower_i + NonGatedClockPower_i \quad (2)$$

In a recent study [8], the authors defined another form of power statistical for GPU power consumption in Equ. (3) and (4),following Equ. (1) and (2).

$$GPU\_Power = Runtime\_Power + IdlePower \quad (3)$$

$$Runtime\_Power = \sum_{i=0}^{n} RP\_Component_i = RP\_SM_s + RP\_Memory \quad (4)$$

With results obtained through the GPU profiler, power analyzer and the cuobjdump, we put forward a more accurate and intuitive power model.

B. Our Energy model based on GPU Native Instruction

In our model, the power consumption of one application (E_{app}) comes in idle energy (E_{idle}) and dynamic energy ($E_{runtime}$), as shown in Equ. (5). The idle power (P_{idle}), as shown in Equ. (6), mainly involves: the clock circuit, on-chip bus, the Giga Thread global scheduler, etc. which are needed when the workload execute. But the dynamic energy comes mainly from the computing units and memory. For the computing units part, because the execution cycles and throughputs differ widely for different native instructions, the energy consumption is also different. So in our energy model, we collect the number of different native instructions in the kernel, and compute the native instruction's energy consumption based on the test of its energy cost. And the energy consumption for memory needs the count of L1/L2/DRAM by the GPU profiler. It concretely analyses as mentioned in section 4.3 and 4.4.

After the analysis we find the energy of one computing unit instruction and one memory/cache access is fixed, uncorrelated with IPC. So we can estimate the energy consumption of one application based on its native code and memory/cache access number. The formula is as shown in Equ. (7).

$$E_{app} = E_{idle} + E_{runtime} \quad (5)$$

$$E_{idle} = P_{idle} * Time_{exe} \quad (6)$$

$$E_{runtime} = \sum_i^n memory_i\_energy * count\_number_i + \sum_j^n Instruction\_energy_j * instruction\_number_j \quad (7)$$

To calculate the number of application instructions, we use a GPU PTX emulator, Ocelot [20]. By Ocelot, we get the procedure of PTX instructions. And we then count native instructions by cuobjdump.

III. MODEL VALIDATION

A. Experimental Setup

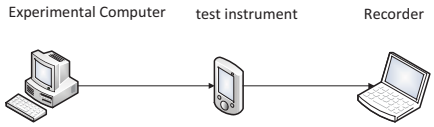

Fig. 1. Power Measurement Setup

As shown in Figure 1, our experimental computer runs Windows 7 on an Intel Core i7 processor with NVidia C2050 (Fermi) GPU Card, 6GB memory and 1TB hard disk. We use power test instrument to measure the power consumption of the applications running on the GPU.

We set CPU frequency and CPU fan speed at a fixed value to wipe off system power gap between idle and full load. For the GPU side, we fixed its fan speed by nVIDIA Inspector[13]. We find the power consumption of GPU increases while GPU temperature rises, of which the change rule is as shown in Figure 2. After adjusting these factors, our energy model can accurately test the energy consumption of one GPU kernel.

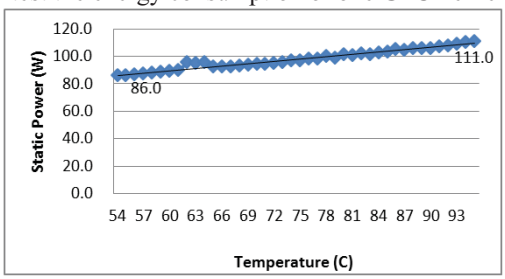

Fig. 2. Power Change with Temperature

The NVIDIA supports a prefect debugging environment, so we can easily catch some performance data by the NVIDIA Visual Profiler. The feedbacks of the profiler include: the kernel execution time, the bandwidth and hit number of caches and memory, IPC, the execute instructions number, and so on.

B. Benchmark for the power model

The code we used to test the instructions executed on computing units(FADD 2 reg as an example) as is shown in Fig. 3(a).

The instruction a = a + b; repeats many times in one thread, which can be translated into ptx code as shown in Fig.3(b). And Fig.3(c) is the native code executed on GPU. We can see that the FADD instructions executed on GPU are all the same

```
void FADD_power(…)
{
…
#pragma unroll 500
        for (int i=0; i<repeat; i++)
        {
                a = a + b;
        }
        …
}
```

(a) The CUDA code of FADD benchmark

```
add.f32  %f3, %f2, %f1;
         add.f32  %f4, %f2, %f3;
         …
         add.f32  %f11, %f2, %f10;
         …
```

(b) The PTX code of FADD benchmark

```
/*0x08009c0050000000*/ FADD R2, R0, R2;
/*0x08009c0050000000*/ FADD R2, R0, R2;
/*0x08009c0050000000*/ FADD R2, R0, R2;
……
```

(c) The native code of FADD benchmark

Fig. 3. Benchmark of FADD Instruction

The code for memory (load data from memory as an example) is as shown in Fig.4.

With the help of CUDA profiler, we can get the number of L1 access, L2 access and DRAM reads. Modifying the code, we can separately get the times of different memory and test the power.

```
void L1Read_power(…)
{
        …
        for (int i=0; i<repeat; i++)
        {
                repeat512(j=*(unsigned int **)j;)
        }
        …
}
```

(a) The CUDA code of load benchmark

```
ld.u32   %r16, [%r15+0];
         ld.u32   %r17, [%r16+0];
         …
         ld.u32   %r25, [%r24+0];
         …
```

(b) The PTX code of load benchmark

```
/*0x00611ca584000000*/ LD.E.64 R4, [R6];
/*0x00419ca584000000*/ LD.E.64 R6, [R4];
/*0x00611ca584000000*/ LD.E.64 R4, [R6];
/*0x00419ca584000000*/ LD.E.64 R6, [R4];
……
```

(c) The native code of load benchmark

Fig. 4. Benchmark of Load

C. Power Parameter Extraction for Computing Units

Because the execution cycle and the throughput of different native instructions are different, we test the energy consumption for each native instruction separately.

Each SM can simultaneously execute 32 native instructions, and a total number of 14 SMs in C2050 consists of 448 native instructions that can be executed at the same time. In early versions of GPU architecture, each Thread Processing Cluster (TPC) consists of three SMs. But in Fermi architecture, there is only one SM in each TPC. So, each SM owns one SM controller and L1 cache/Shared memory separately, and the instruction operations are separated between the SMs. So, in theory, the power and energy consumption for each ALU/FPU instructions is in proportion to the number of SMs.

In GPU, the parameter that reflects the utilization of instructions is instructions per cycle (IPC). For FMA instructions, for instance, the throughput is 32, and the execution cycle is 16, so the IPC is 2 (32/16=2) at most. When some SMs stop work, a too small blocksize (< 448) may reduce the IPC. Using the different number of threads, we find that the power of instruction increases linearly with IPC, but the energy consumption of instruction is uncorrelated with IPC, as shown in Figure 5.

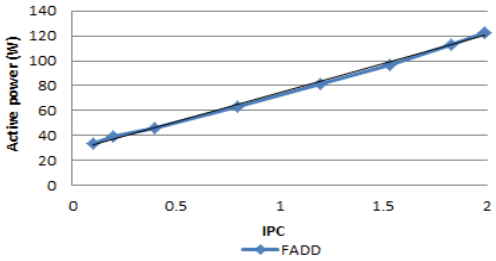

Fig. 5. FADD Power vs. IPC

The instructions executed on SFU are similar to those on ALU/SFU, but as described in section 2.2, SFU cannot execute one PTX instruction separately, and has to work together with FPU. And there are only 4 SFU for one SM, so IPC is smaller.

The latency, throughput and energy consumption of major ALU/ SFU/ FPU instructions is as shown in Table 1.

TABLE I THE ENERGY OF INSTRUCTION

Native instruction	Energy (nJ/warp)	Native instruction	Energy (nJ/warp)
FADD (2 reg)	5.09	IADD (2 reg)	2.24
FADD (1 reg)	3.39	IADD (1 reg)	2.12
FADD32I	3.58	ISCADD	1.74
FMUL32I	4.65	SEL	3.11
FMUL (2 reg)	5.54	IMUL (2 reg)	5.12
FMUL (1 reg)	4.86	IMUL (1 reg)	4.87
FFMA (3 reg)	5.91	IMAD (3 reg)	5.99
FFMA (2 reg)	5.47	IMAD (2 reg)	5.24
FMNMX	2.98	IMNMX	1.66
FSETP	4.02	ISETP	1.14
DADD	10.63	LOP.AND (OR,XOR)	2.01
DMUL	12.25	SHL	3.66
DFMA	13.38	BAR	4.16
DMNMX	7.75	MOV	1.52
DSETP	5.93	MOV32I	1.17

There are some multifunk instructions executed by SFU and FPU. And these instructions can be cut into RRO, MUFU and other FPU instructions for native instructions. There are only the instructions of RRO and MUFU executed on SFU, but because the SFU instructions include a lot of judging instructions, the energy consumption of SFU instruction is between 12.9nJ to 25.1nJ.

D. Power Parameter Extraction for Memory

As we know, L1 cache is in SM, so its power increases with the SM number. For the relationship between memory ratio and bandwidth is most closely, we observe the L1 cache energy consumption change with the L1 bandwidth as the GPU is full, as shown in Figure 6. The first line (blue) is the L1 cache energy consumption increase with different bandwidth. But when the idle power is removed, the result is as line 2 (orange) shown. We can see from the figure that after reduction of the fixed energy consumption, the function relation between the energy consumption of one L1 access (E_{L1}) and its bandwidth can be fitted in formula (8):

$$E_{L1} = 0.0307 * bandwidth\_L1^{-0.578} \quad (8)$$

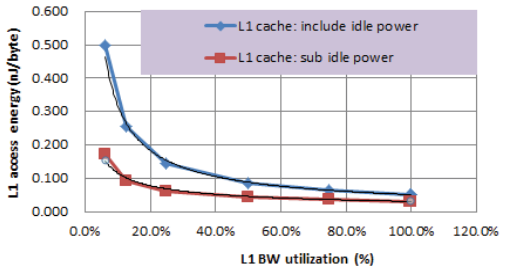

Fig. 6. L1 cache access energy vs bandwidth

And the energy consumption of L2 and shared memory access are similar to L1 access, so the formulas are all nearly the same in spite of different constant parameters. But the energy consumption of Dram is uncorrelated with bandwidth, and power is linearly with the number of Dram access times. The parameters of different memory and the energy cost of each access with the full bandwidth is shown in Table 2.

TABLE II THE ENERGY OF DIFFERENT MEMORY

Memory	Parameter a	Parameter b	Energy(nJ /byte)
L1 Cache	0.0307	-0.578	0.021
L2 Cache	0.145	-0.365	0.13
Read shared memory	0.0292	-0.596	0.021
Store shared memory	0.0297	-0.542	0.021
read Dram memory	--	--	0.54
write Dram memory	--	--	0.53
Texture cache	0.145	-0.299	0.14

IV. EXPERIMENTAL RESULTS

A. Task Kernels on GPU

There are many kinds of benchmark suite for GPU architecture. Among these, the representative workloads include: GPU Computing SDK for NVIDIA[17], the parboil[18] for the Illinois Microarchitecture Project Team, and the Rodinia[19] for the research staff from Department of Computer Science in University of Virginia. However, a big problem is which benchmark can help us have a real and comprehensive understanding of the GPU architecture. In [21], the authors proposed an ideal evaluation mechanism that must be accurate, thorough and realistic.

Many researches have analyzed the programming methods on GPU[14][22][26]. Among the factors affecting the performance of the workloads, the important parts include

branch divergence, barrier, type of memory, block number and size, the series of instructions and the series of memory/cache.

To verify the accuracy of our energy model, we choose a series of workloads from the three kinds of benchmark [17][18][19] and other third-part code. The workload includes: MonteCarlo (float, double), Reduction, MatrixMul (GPU Computing SDK), Sgemm, blackscholer, transpose, STREAM, histogram, Mersenne Twister, quasirandom Generator, sorting Networks.

The kernel of MonteCarlo[23] uses Monte Carlo to simulate Single Asian Options. This is a typical computing bound kernel

Sparse matrix-vector multiplication (SpMV)[24] is singularly important in sparse linear algebra. This kernel tests the execution efficiency of SpMV under different storage requirements.

Matrixmul[17] is an important example in CUDA programming guide. The kernel reflects the features of NVIDIA GPU architecture and is memory bound.

The kernel of reduction[17] computes the sum of a large arrays of values. The kernel is a typical memory bound and includes a lot of access Dram.

Sgemm[25] is an important part of Linpack, and the kernel is optimization on CUDA programming. So the kernel proves the computing ability of GPU architecture.

BlackScholes[17] is an analytical means of computing the price of European options. Many kinds of multifunk instructions are included and the Dram is accessed repeatedly.

Transpose[17] demonstrates the process of matrix transpose. The kernel is optimization using shared memory.

STREAM is a pure kernel of testing access Dram. There are only few int instructions in the kernel.

Histogram[18] includes many divergence branches. And the kernel can show the impact of branch to the power model.

Mersenne Twister[17] random number generator is a typical application in CUDA. And the feature of the kernel is without complex computing to generation the random number.

Quasirandom Generator is the base method to random number generation. And the kinds of computing units are abundant.

SortingNetworks[17] implements bitonic sort and odd-even merge sort. There are many logic instructions and access Dram.

The instruction type of these kernels as shown in Table 3:

TABLE III INSTRUCTION TYPE OF KERNELS

kernel	Int	Float	Double	SFU	Shared mem	L1/L2/DRAM
MC(float)		Y		Y		Y/Y/Y
MC(double)		Y	Y	Y		Y/Y/Y
SpMV	Y				Y	N/N/Y
matrixmul		Y			Y	Y/Y/Y
Reduction	Y					N/N/Y
Sgemm		Y			Y	N/Y/Y
Blackscholer		Y		Y		N/N/Y
Transpose	Y				Y	N/N/Y
STREAM	Y		Y			N/N/Y
histogram	Y				Y	N/N/Y
MT	Y			Y		Y/Y/Y
Quasirandom Generator	Y	Y	Y			Y/Y/N
Sorting Networks	Y				Y	N/N/Y

B. Result Analysis

Through the analysis above, the workload include all the power resource part of GPU. The comparison of testing result and our energy model result as shown below:

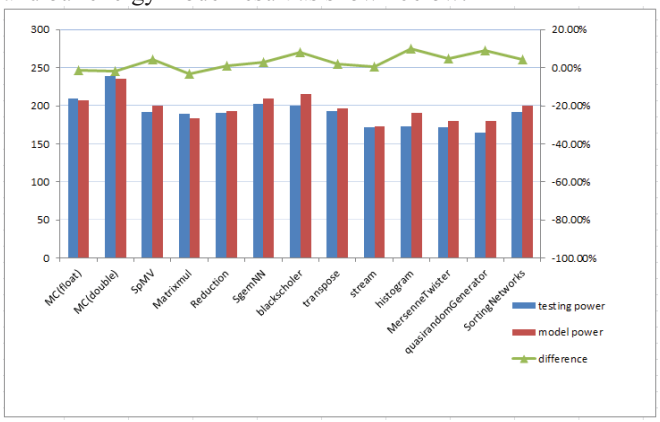

Fig. 7. Testing power and our model power comparison chart

The accuracy of our energy model is more than 90%. As shown in Fig 7, there are five workloads with the accuracy more than 98%; and five workloads with an accuracy of 92%-98%; but, three workloads' difference is larger than 8%. The results of MonteCarlo (float), MonteCarlo (double), Reduction, transpose and stream are precise, for the instructions of them are centred only on computing units or memory. SpMV, Matrixmul, SgemNN, MersenneTwister, SortingNetworks includes many kinds of computing units and memory, so that this decreases the accuracy of our model. There are a lot of divergence branch in histogram, the instruction execution efficiency of blackscholer and the measuring accuracy of quasirandom generator are low, so the accuracy of these three workloads are less satisfactory.

V. CONCLUSIONS

In this paper, we presented a novel energy model to estimate the power of new generation GPU architecture. With proper profilers and tools, we identified major power contributors in GPU architecture. These contributors can be divided into two groups, i.e. computing unit and memory access on which different native instruction were run and measured separately as a foundation for our energy estimate model.

It is exciting that our energy model is able to reach excellent accuracy of more than 90% for the applications executed on the Fermi GPU architecture. Our energy estimation model and corresponding methods should be of significant value to other researchers on other power models as well as software system architects.

In our future work, we will explore power consumption behavior in the process of graphics displaying and extend the valid range of our energy model. We will also incorporate new ideas such as control flow divergence in [29].

ACKNOWLEDGMENT

This paper is partially sponsored by the National High-Technology Research and Development Program of China (863 Program) (No. 2009AA012201).

REFERENCES

[1] NVIDIA Inc., NVIDIA's Next Generation CUDATM Compute Architecture: FermiTM, Version 1.1, 2009.

[2] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems, " Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, September 2000.

[3] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in International Symposium on Computer architecture (ISCA), June 2000.

[4] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," Proc. MICRO-36, 2003.

[5] NVIDIA Inc., PTX: Parallel Thread Execution ISA Version 2.1, http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/ptx_isa_2.3.pdf

[6] NVIDIA Inc., Cuobjdump, http://www.dahlsys.com/upload/cuobjdump.pdf

[7] X. Ma, M. Dong, L. Zhong and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-based Computing," Workshop on Power Aware Computing and Systems (Hot Power '09), 2009.

[8] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model," in International symposium on Computer architecture (ISCA), 2010.

[9] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo and S. Matsuoka, "Statistical Power Modeling of GPU Kernels Using Performance Counters," International Green Computing Conference (IGCC), 2010.

[10] Extech Instruments Corporation, Power Analyzer Model 380801, http://www.extech.com/instruments/resources/manuals/380801_803_UM.pdf

[11] NVIDIA Inc., CUDA, http://www.nvidia.com/object/cuda_home.html.

[12] Khronos OpenCL Working Group, OpenCL, http://developer.nvidia.com/opencl

[13] NVIDIA Inc., nvidia Inspector, http://www.geeks3d.com/20110305/nvidia-inspector-1-9-5-5-available/

[14] NVIDIA Inc., NVIDIA CUDA C Programming Guide.

[15] The OpenCL Specification.

[16] NVIDIA Inc., NVIDIA Visual Profiler, http://developer.nvidia.com/content/nvidia-visual-profiler

[17] NVIDIA Inc., GPU Computing SDK, http://developer.nvidia.com/gpu-computing-sdk

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

[19] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaer, S.H. Lee and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," In Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC '09), pages 44-54, Austin, TX, USA, 2009, IEEE.

[20] A. Kerr, G. Diamos, and S. Yalamanchili, "A characterization and analysis of ptx kernels," IISWC, 2009.

[21] N. Goswami, R. Shankar, M. Joshi and T. Li, "Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications", IISWC, 2010.

[22] S. Ryoo, C. I. Rodrigues, S. S. Stone, J. A. Stratton, Sain-Zee Ueng, S. S. Baghsorkhi, and W. W. Hwu, "Program optimization carving for GPU computing", Journal of Parallel and Distributed Computing, vol. 68 (10), 2008, 1389-1401.

[23] Craig Kolb and Matt Pharr, Option pricing on the GPU, GPU Gems 2, Chapter45. 2007.

[24] Nathan Bell, Michael Garland. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004. 2008.

[25] Y. Li, J. Dongarra, and S. Tomov, A note on auto-tuning GEMM for GPUs, Tech. report, LAPACK Working Note 212, 2009.

[26] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general purpose applications on graphics processors using CUDA," Journal of Parallel and Distributed Computing, 68(10):1370–1380, 2008.

[27] NVIDIA Inc., Tesla C2050, http://www.nvidia.cn/object/product_tesla_C2050_C2070_cn.html

[28] W. Dally. Power efficient supercomputing. Presented at the Accelerator-based Computing and Manycore Workshop. Nov. 30–Dec. 2, 2009.

[29] Z. Cui, Y. Liang, K. Rupnow and D. Chen, "An Accurate GPU Performance Model for Effective Control Flow Divergence Optimization", International Parallel and Distributed Processing Symposium, 2012.