Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing

Reiji Suda
Graduate School of Information Science and Technology,
the University of Tokyo & JST, CREST
Tokyo, Japan
Email: reiji@is.s.u-tokyo.ac.jp

Da Qi Ren
Graduate School of Information Science and Technology,
the University of Tokyo & JST, CREST
Tokyo, Japan
Email: dren@is.s.u-tokyo.ac.jp

***Abstract*—Power dissipation is one of the most imminent limitation factors influencing the development of High Performance Computing (HPC). Toward power-efficient HPC on CPU-GPU hybrid platform, we are investigating software methodologies to achieve optimized power utilization by algorithm design and programming technique. In this paper we discuss power measurements of GPU, propose a method of automatic extraction of power data of CUDA kernels from long measurement sequence, and execute an exactitude and effective power analysis on CUDA kernels. By using the proposed method above, we measured a sample kernel that performs single precision floating point additions on GeForce 8800 GTS. Our results suggest that the power consumption by a non-working thread in underoccupied half-warp is 71% of the power consumed by a working thread.**

***Keywords*-GPGPU, CUDA, power dissipation, power modeling.**

I. Introduction

Power dissipation is the utmost problem in the next generation high performance computers. In order to realize ultra-scale high performance scientific computing, power dissipation of supercomputers must be reduced effectually without losing computing performance. This is also true for high performance workstations and personal computers: The imminent problem of processor heat demands serious efforts of power reduction in every possible factor, such as silicon device, cooling hardware, computer architecture, and software.

Now advantages of GPUs (Graphic Processing Units) as high performance computing accelerators are investigated by many researchers of various fields. As a massively parallel SIMD processor, GPU is not only high performance but also energy efficient. High-end GPUs consume more power than CPUs, however GPU/CPU performance ratio is higher than the power ratio, and thus GPU can complete more computations than CPU does by using fixed energy. This fact has been demonstrated by several researchers. For example, Nukada et al.[1] reported that energy efficiency of GPU is better than CPU by a factor of 4 in computing FFT (Fast Fourier Transform).

Our research aims to develop software methods for power optimized high performance computing on CPU-GPU heterogeneous platforms [2], and to minimize the overall energy dissipation by automatically adjusting the utilization of power related components in the system with automatic tuning mechanisms. Our approach to this end is to firstly build precise models of the power consumption and of the execution time, by which we are able to predict the energy dissipation and the time of a specific computation executed on the target CPU-GPU platform; next based on that information, to tune power related parameters such as clock frequency of CPU, GPU and memory, load distribution schemes, communication protocol etc.; finally to reach the best solution of power performance and computation performance for the given problem.

In this paper, we investigate a method of efficient and accurate power measurement of a video card under various computational loads. We use CUDA[3] for programming on the GPU. Using the proposed method, we could measure power consumptions of 2480 CUDA kernel calls in less than 30 seconds. From a sequence of measurement data, power consumption of each kernel call is extracted automatically by software. We will show preliminary measurement results and a power model of a kernel with single precision additions on GeForce 8800 GTS. From the measurements we suggest existence of non-working threads of the underoccupied half-warp whose power dissipation is about 71% of a working thread.

There has been several research works on power consumption of CPU-GPU platforms, but most of them measured the complex of CPU and GPU. Relatively few works measured video card separately from the host processor system. Rofouei et al.[4] measured two programs from CUDA SDK samples and compares power consumption of CPU and GPU. Their interest is in comparisons of CPU-only and CPU-GPU platforms. Collange et al.[5] measured various micro-benchmarks written in CUDA, which is similar to ours. However, they did not build any power model.

II. GPU AND CUDA

In this section, we provide a brief description of some technical details of GPU hardware and software that are related to our work. More about CUDA and NVIDIA GPU can be found in the CUDA programming guide[3] and related materials.

In this paper, a GPU is assumed to be provided on a *video card*. On a video card, video memory, fan and other circuit components are mounted in addition to GPU. We measured the power of a video card, so power dissipation of all circuit components on the video card is included.

CUDA is an extension of C language for programming on NVIDIA GPU. The computations on a GPU are programmed as *kernel* functions. A kernel program describes the execution of a serial *thread* on a GPU. The kernel is launched by the host CPU with specified numbers of blocks and threads, where a *block* represents a set of a certain number of threads, and all blocks in that kernel launch have the same numbers of threads. The total number of threads is the number of blocks times the number of threads per block.

16 consecutive threads in a block form a *half-warp*, and 2 half-warps form a *warp*. Threads in a warp are executed in SIMD, and warps in a block are executed concurrently in SPMD. Blocks may be executed in parallel or in serial.

Our target GPU (GeForce 8800 GTS) has 12 MPs (Multi-Processors) as computing cores, and each MP has 8 SPs (Streaming Processors) to execute threads.

A block is assigned to an MP and each thread in that block is assigned to an SP of that MP. Up to 16 warps or 512 threads can be executed concurrently on an MP. Scheduling of warps is done by the hardware, and is very fine grain and light weight.

III. POWER MEASUREMENTS OF VIDEO CARDS

A. Choosing the Target of the Power Measurement

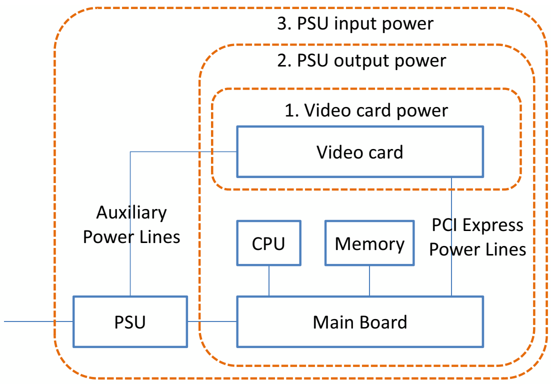

Figure 1. Three possible targets of power measurement

To measure the power consumption of a computing GPU, there will be three options, as shown in Fig. 1:

1) Measuring power consumption of the video card,
2) Measuring output of the PSU (Power Supply Unit), and
3) Measuring input of the PSU.

Measuring power consumption of the video card is a direct way. However, the other options have their own advantages.

The power output of the PSU includes the power consumption of CPU, memory etc. Since GPU does not run by itself and some CPU power is required to control the computing GPU, it is not precise to regard the power consumption only of the video card as the power of GPU computing. However, measuring PSU output power is not always ideal for GPU power analysis from various viewpoints, e.g., the CPU can be used by tasks other than the control of GPU.

The power input of the PSU includes the power dissipation (or loss) of the PSU itself. The power loss of a typical PSU can be 20% or more of the total power dissipation, and so it is too high to ignore. The power input to the PSU has relatively long latency (more than 10 ms) against the variation of the power load, and thus phenomena of durations shorter than that are unobservable through measurements of the PSU input.

The goal of the present work is to observe the relations between the power consumption of a video card and the programs running on them and to build a model of them. To this end, measuring power consumption of the video card will provide the most accurate data, undisturbed by the other parts of the platform. In our future works, CPU power for GPU control and power loss of PSU will be measured and analyzed.

B. Measuring Power Consumption of a Video Card

A video card on which CUDA programs run is connected via PCI-Express to the main board of the host processor. A part of power to a video card is supplied through the PCI-Express bus, but it is not enough for video cards with high performance GPUs, and additional power is supplied directly from the PSU (as shown in Fig. 1). Let us call the former **PCI-E power** and the latter **auxiliary power**, for brevity. PCI-E power is supplied in two voltages: **12V** and **3.3V**. The voltage of auxiliary power is 12V.

To measure the current through the auxiliary power line, a clamp probe is used. To measure the voltage, we remove parts of the coating of some (one of 12V and one of GND) of the auxiliary power lines, and attach a voltage probe to them.

To measure the power supply through the PCI-Express connector, we introduced a riser card. We separate 12V and 3.3V power supply lines of the riser card from the others. We use a clamp probe again to measure the current. To measure the voltage, we remove parts of the coating of some of the PCI-E power supply lines, and attach a voltage probe to them. Fig. 2 summarizes the connections of the probes.

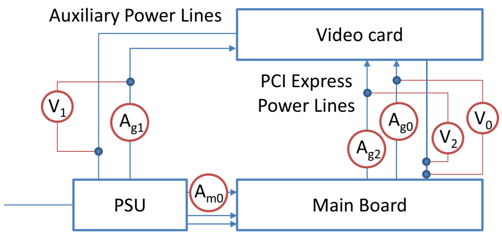

Figure 2. Diagram of measurements

Some factors of the measured platform are described in Table I: We use an old card of GeForce 8800 GTS with 640 MB of video memory, because the riser card we currently use does not work with PCI-Express gen. 2.0 or higher. We use National Instruments USB-6216 BNC as data acquisition, Fluke i30s / i310s as current probes, and Yokogawa 700925 as voltage probe. The room was air-conditioned in 23°C.

Table I
FACTORS OF MEASURED PLATFORM

Video Card	NVIDIA GeForce 8800 GTS
CPU	Intel Core i7 920 2.66 GHz
Memory	DDR3 SDRAM PC3-8500 3GB
M/B	GIGABYTE GA-EX58-UD3R
PSU	Seasonic SS-700HM
OS	ubuntu 8.10 (64 bit)
Compiler	CUDA 2.2 / gcc 4.3.2
CUDA Driver	version 185.18

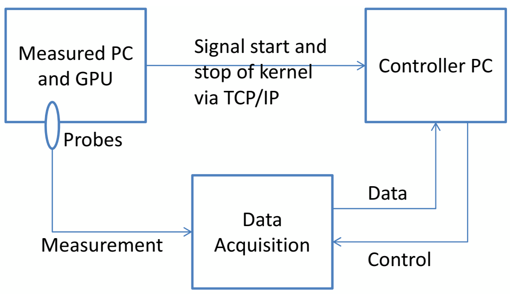

Figure 3. Schematic of measurements

Fig. 3 is a schematic figure of our measurements. We use a separate PC to control the measurement equipments and collect measurement data. This seems better for measurement accuracy, because it takes a considerable amount of CPU power to control the measurement equipments and to collect measurement data.

C. Equivalent Circuit Analysis of GPU Power Supply

We perform circuit analysis on the GPU power supply circuit by replacing them with a few notional components that have the same effect. Two circuits are equivalent if the voltage and current for one network have the same relationship as the voltage and current of the other network. If we are able find out the values of the equivalent notional components precisely, the abstract features of the target circuit become available. This method can also benefit us in practical applications. For example, the wires of the PCI-Express riser card are very thin, the diameter of each wire including its coating is about 0.6 mm, and the diameter of the conductor part in the center is only around 0.2–0.3 mm. The conductor wire can be considerably deformed when a voltage probe is attached on it, and the power input to the GPU card through this wire can be affected. Voltage measurements by probe may also cause some security issues: If one of the power wires is broken, or the exposed conductor wires accidently short, the GPU hardware may be seriously damaged. All these problems can be solved by using equivalent circuit that will reasonably minimize the number of voltage measurements.

The input voltages to GPU are defined as constants such as 3.3V and 12V, but the actual voltages will alter when circuit is closed. Directly using the nominal values of the voltage source will induce a considerable loss of accuracy. Based on our experiments, the errors caused from using the nominal voltages can be as large as 3W, which is much higher than the errors produced from using our method, which is estimated as 1W.

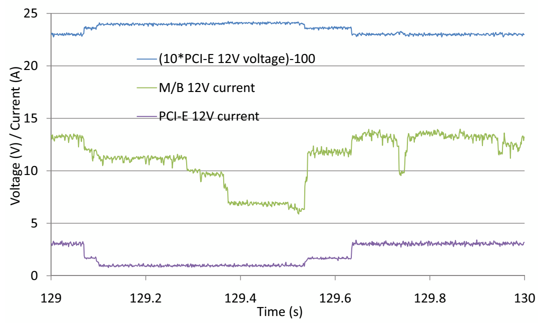

Figure 4. Voltage and current variation in time

Fig. 4 shows a measurement of the variations of the voltages of 12V PCI-Express power supply, the input currents of the video cards and the input currents of the main board. Note that the voltage is scaled and shifted to fit in the figure. In this figure, the 12V PCI-Express voltage drops when CPU or GPU power consumption increase. We presume the equivalent circuit is a resistive circuit. To verify this assumption, we use CUBLAS[8] and ATLAS[7] as loads on GPU and CPU, respectively, for the circuit testing. The voltages and the currents are probed every millisecond, and totally 146,000 samples were obtained. The approximation is derived from those data by using a standard linear least squares method. The resulting equivalent circuits are the followings (we omit units V, A and Ω):

$$\begin{array}{ll} V_0 \approx 12.471 - R_{g0}A_{g0} - R_{m0}A_{m0} & \text{at PCI-E 12V,} \\ V_1 \approx 12.514 - R_{g1}A_{g1} & \text{at Aux 12V,} \\ V_2 \approx \quad 3.554 - R_{g2}A_{g2} & \text{at PCI-E 3.3V,} \end{array}$$

where equivalent resistances are $R_{g0} = 0.044$, $R_{m0} = 0.003$, $R_{g1} = 0.020$, and $R_{g2} = 0.110$. V_0, V_1, and V_2 are the voltages of PCI-Express 12V line, auxiliary 12V line, and PCI-Express 3.3V line, respectively. A_{g0}, A_{g1}, and A_{g2} represent the current to the video card through PCI-Express 12V line, auxiliary 12V line, and PCI-Express 3.3V line, respectively. A_{m0} is the current to the main board in 12V (see Fig. 2). Because the PCI-E power is supplied from the main board, the linear expression of V_0 involves the term $R_{m0}A_{m0}$, which is related to the main board input current A_{m0} (cf. Fig. 4). However, we do not include similar terms to V_2 and V_1 because they improve the accuracy little.

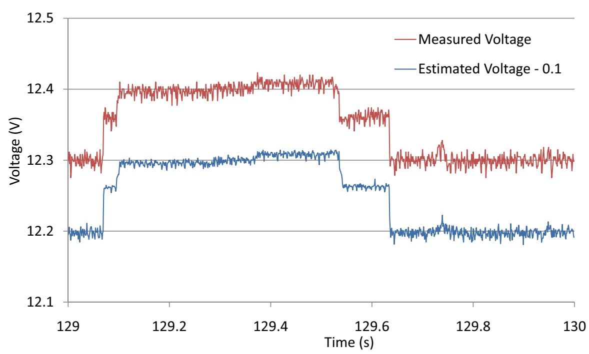

Figure 5. Observed Voltage and Estimated Voltage (shifted by 0.1)

Fig. 5 shows the fitting of the approximation from equivalent circuits of the PCI Express 12V for the same part of the measurement as shown in Fig. 4. (The estimated voltage is shifted by 0.1 in order to remove overlaps.) The estimated voltage approximates the observed voltage very well, and the good fitting is not only for the part shown in Fig. 5. The root mean square error over all measurement points of PCI-E 12V is 6.7mV. The root mean square error of the model of the PCI-E 3.3V is 5.1mV, and that of the auxiliary 12V is 21mV. As is shown in Fig. 5, the estimated voltage has less vibration than the measured one. Actually, a major part of the root mean square errors of our models comes from the perturbation of the *observed* voltage rather than the model inaccuracy. This fact implies that, using our model, one can estimate the power consumption with less statistical error. This is reasonable from statistical point of view: By building a precise model from a large set of samples, it effects as if a kind of average is taken, and the *variance* is reduced by introducing a small *bias*.

Our model is also validated in Table II. First, we run 1 to 7 processes of FFTE[9] (1D FFT of various sizes) and Lis[10] (sparse linear solver), and our own CUDA benchmarks reported in section V. Second, the same set of tasks as before (ATLAS + CUBLAS) is executed without air conditioning. Table II reports root mean square errors in voltage and in Watt (by multiplying with the current). Our model gives good approximations in those experiments.

Table II
Root Mean Square Errors of Voltage Estimation

	FFTE/Lis + CUDA		ATLAS + CUBLAS	
	Voltage	Power	Voltage	Power
PCI-E 12V	11 mV	0.03W	19 mV	0.03 W
Aux 12V	23 mV	0.11W	29 mV	0.11 W
PCI-E 3V	5 mV	0.01W	8 mV	0.01 W

In addition, another advantage of our voltage estimation is that we need a less number of probes in measurements. Ignoring the benefit of this in economic sense, the most important meaning is the time resolution of the measurements: The data acquisition probes each input in a round-robin fashion, thus the highest time resolution is reversely proportional to the number of probes. Based on the above advantages of our model, we continue our measurement in the following sections by observing the currents only, and estimate the voltages using the model built in this subsection.

IV. Automatic Extraction of Energy Consumption of CUDA Kernel from Measurement Sequence

To measure the power consumption of the video card during a specific CUDA kernel execution, we have to extract the measurement data corresponding to that kernel execution from a sequence of measurement data. To achieve this, we propose to use *markers* consists of executions of kernels with different intensities of memory accesses. Insertion of markers is not our idea: Nishikawa and Aoki[6] used a CPU sleep as a marker. Our marker allows higher time resolution and automatic data extraction by software.

A. The Proposed Marker

It is known that a kernel consumes high power when it accesses memory intensively[6]. So we design our marker with two kernels with different intensities of memory accesses. One kernel works on memory copy (`cudaMemcpy`) on the device memory, and the other one repeats floating point operations with one thread inside one block without accessing memory.

From experiments we estimated the time constant of the video card (described in Table 1) is between 20μs and 50μs. If a kernel takes a time shorter than 20μs, another kernel starts before the observed power changes in full range, and the difference of the observed power consumptions of two kernels with different memory access intensities becomes ambiguous. If the kernels take some times longer than 50μs,

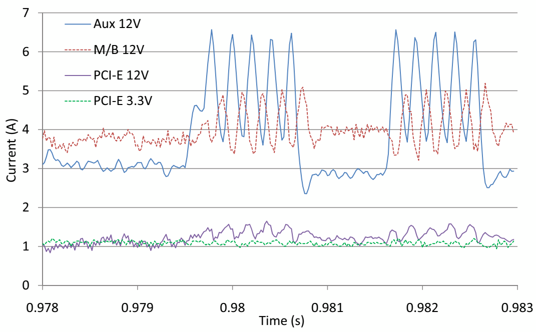

Figure 6. Measurement data with markers

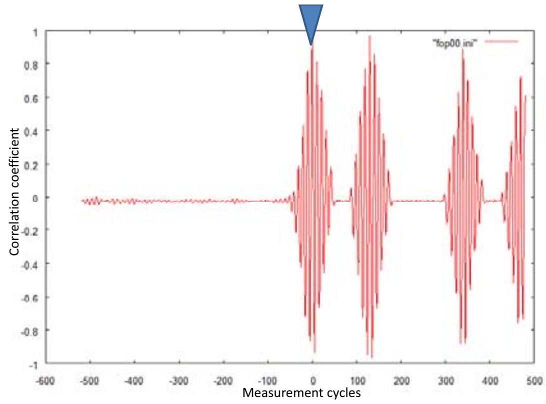

Figure 7. Identifying the first marker

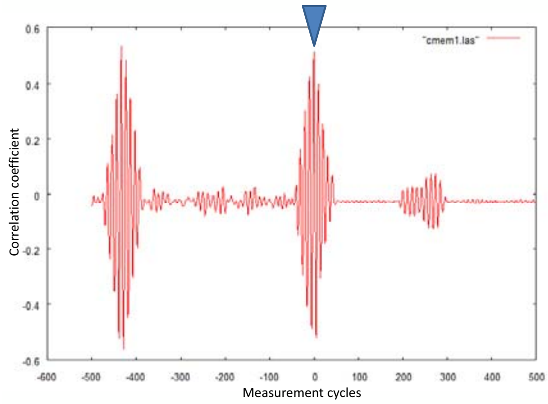

Figure 8. Identifying the last marker

then their power consumptions are clearly observed. We use kernels that take about 100μs.

Our marker consists of five calls of memory copy and four calls of computation kernels between memory copies. Fig. 6 shows the measurement data of markers, where GPU is idle in the period between the markers (CPU waits for 1ms to pass). The marker is most clearly seen in PCI-E 12V current. So we use only PCI-E 12V current to identify markers in the measurement sequence.

B. *Automatic Marker Identification*

As shown in Fig. 6, the marker can be identified so clearly in the measurement data that we can identify the markers automatically by software.

To this purpose, we calculate an average measurement sequence of several markers, and then use a simple pattern matching method (computing correlation coefficients) to find markers in a sequence of measurement data. The correlation coefficients are calculated with a sliding window, and the position where the correlation coefficient takes its maximum value is assumed to be the position of the marker. Fig. 7 plots the correlation coefficients in a measurement sequence. The x-axis represents the position of the sliding window relative to the estimated first marker. The triangle ($x = 0$) shows the estimated position of the first marker in that sequence.

The host CPU records the wall-clock times of the calls of the markers and the kernels. Using that timing information, we can predict the position of the next marker very precisely from the previous one, and identify the next maker in the measurement data easily. This scheme allows us to identify thousands of markers from a measurement sequence efficiently and correctly.

In order to check whether all the markers are correctly identified or not, we plot data around the first and the last markers in the measurement data, as shown in Fig. 7 and Fig. 8.

C. *Calculation of Power Consumption of CUDA Kernels*

After identifying the markers, we calculate the energy consumption of the executions of CUDA kernels from the measurement data between markers.

However, the observed power can be affected by the inserted markers. To minimize influences of the markers in the energy calculation of the kernels, the experiments are done as follows: Each CUDA kernel is written with a loop, and called twice. In the second call, the number of repeats in the loop is two times the repeats in the first call. We regard the difference of the energy consumptions of the two calls as corresponding to the difference of the computations of the two calls. The number of repeats is predefined so that the kernel call takes 500μs – 1ms.

We measure the energy consumption of each kernel 10 times, remove the maximum and the minimum values, and take an average of the remaining 8 measurements. Dividing the result by an average execution time, average power consumption is calculated. We estimate the overall absolute accuracy as 1–2W, including probe accuracy, statistical variance, and errors of the circuit model discussed in section III-C.

V. Preliminary Results of Measurement and Power Model

We now start to measure CUDA kernels by using the above-mentioned method. Note that, in this work we conduct preliminary measurements in order to find what kind of kernels need to be measured, how much accuracy is required to derive a power model of CUDA kernels, and how many measurements are enough to attain that accuracy. In this section, we will show some preliminary results.

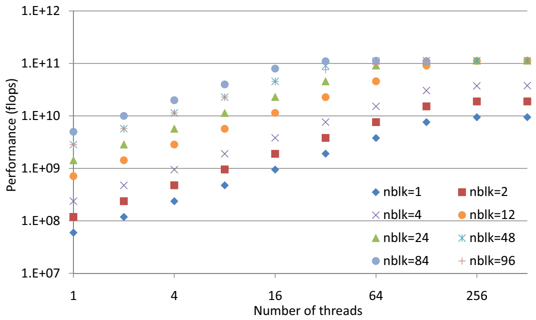

Figure 9. Flops performance versus number of threads

To study the power features of GPU computation, we launch a series of CUDA kernels with various numbers of blocks and threads that repeatedly perform single precision floating point additions. Fig. 9 shows flops (floating operation per second) performance of kernels. To obtain those data, the CUDA kernel is called 2480 times, interleaved by 2481 calls of markers. The measurement took 21 seconds. In Fig. 9 and some of the following plots, the numbers of blocks are shown with "nblk=."

The theoretical peak performance of the measured GPU is 114 Gflops (for floating point addition). With large numbers of blocks, the resulting performance was almost exactly same as the theoretical peak. However, with 96 blocks, the performance is same as 48 blocks (half of 96 blocks), which implies that some of 96 blocks are executed after others.

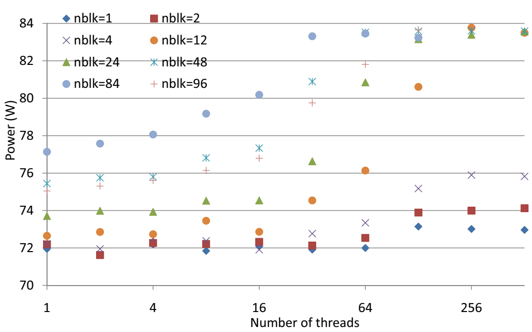

Figure 10. Average power versus number of threads

Fig. 10 shows average power consumption of the kernel executions. More power is consumed by a kernel call with larger numbers of blocks and threads. When the number of blocks is 96, the average power is lower than some other cases, possibly because a part of blocks are executed serially.

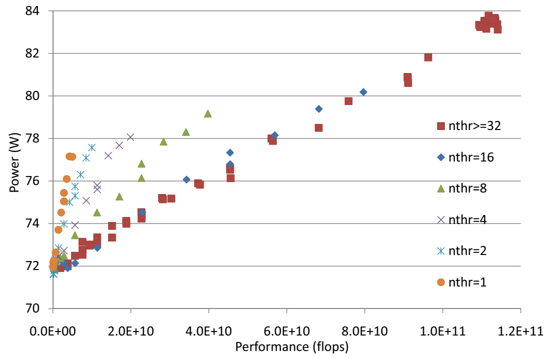

Figure 11. Average power versus flops performance

Fig. 11 plots the average power in the y-axis and the flops performance in the x-axis, where we add data with 3, 6, 36, 60, and 72 blocks to those plotted in Fig. 9 and Fig. 10. Here "#nthr>=32" includes the cases with 32, 64, 128, 256, and 512 threads. When the number of threads is 16 or larger, the relation between power and performance is almost precisely affine. However when the number of threads is less than 16, the kernels consume relatively higher power.

The results in Fig. 11 can be analyzed as follows. Let the number of threads per block be θ, the flops performance be f (flops), and the average wattage be w (W). Then w can be approximated as

$$\begin{aligned} w &\approx 72.0 + 1.02 \times 10^{-10} \rho f, \\ \rho &= \begin{cases} 1 & \theta \equiv 0 \pmod{16} \\ 1 + 0.71 \frac{16-(\theta \bmod 16)}{\theta} & \text{otherwise} \end{cases} . \end{aligned}$$

The root mean square error of this approximation was 0.29(W). Fig. 12 plots the observed wattage in the x-axis and the above approximation model in the y-axis. The model approximates the actual power well, as all points are close to the line $x = y$.

The above model can be interpreted as follows: If θ (the number of threads) is a multiple of 16, that is, all half-warps are filled with working threads, $\rho = 1$. In such a case, the wattage is the sum of base power 72.0W plus 1.02×10^{-10}W per flops. Otherwise, $\rho > 1$ and GPU consumes more power. In such a case, the GPU adds *non-working threads* to fill the underoccupied half-warps (though operations of those non-working threads are ineffectual). Here, the number of the non-working threads is $16 - (\theta \bmod 16)$. Thus the ratio of the numbers of the non-working and the working threads is $[16 - (\theta \bmod 16)]/\theta$. The above model suggests that a non-working thread consumes 71% of the power dissipation of

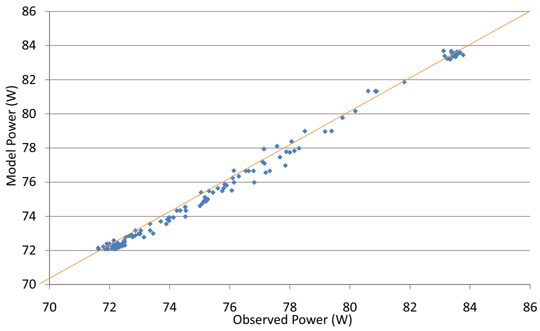

Figure 12. Observed power versus model power

a working thread.

However the above formula is applicable only to the measurements shown Fig. 11, this precise modeling has become possible by our method discussed in the previous sections. Our method enables accurate measurements of CUDA kernels as short as a few 100 μs and automatic extraction of CUDA kernel parts from a measurement sequence. Using our method, we could measure thousands of CUDA kernel calls in a short time, and thus build a precise power model by statistical analysis using abundant sample data.

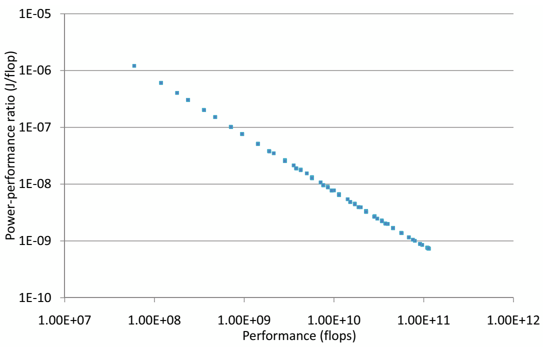

Figure 13. Joule per flop versus flops performance

Fig. 13 plots the average energy for an execution of a single floating addition (Joule per flop) versus flops performance. It is clear that higher computational performance implies higher energy efficiency. Note that both the flops performance and the energy efficiency vary in three orders. Thus performance and power optimization by software tuning can have drastic effects.

VI. Summary and Future Works

In this paper we have discussed efficient and accurate power measurement of GPU with CUDA programming. The proposed method allows measurement of 2480 kernel calls in 21 seconds, and automatic calculation of their power dissipations. We have also shown preliminary measurement data, and have built a power model that fits to the measured data. From the experiments, existence of non-working threads to fill underoccupied half-warps is suggested, and the power dissipation by a non-working thread has been estimated as 71% of the power dissipation of a working thread.

Power modeling and optimization of GPU execution is a sub-goal of our research. We are considering further level of power reduction by optimizing allocations of computing loads to CPU and GPU. To this purpose, we will measure and analyze power dissipation of CPU and the whole complex of CPU-GPU computing platform. Based on them, we will investigate low-power high-performance computing by software automatic tuning approach.

Acknowledgements

This work is partially supported by Core Research of Evolutional Science and Technology (CREST) project "ULP-HPC: Ultra Low-Power, High-Performance Computing via Modeling and Optimization of Next Generation HPC Technologies" of Japan Science and Technology Agency (JST) and Grant-in-Aid for Scientific Research of MEXT Japan.

References

[1] A. Nukada, Y. Ogata, T. Endo and S. Matsuoka, "Bandwidth Intensive 3-D FFT kernel for GPUs using CUDA," *Proceedings of SC2008* (electronic), 11 pages, 2008.

[2] D. Q. Ren and R. Suda, "Modeling and Estimation for the Power Consumption of Matrix Computations on Multi-core CPU and GPU platform," Proc. IEEE International Workshop on HPC and Grid Applications (IWHGA 2009).

[3] NVIDIA, "NVIDIA CUDA Programming Guide," version 2.2, 2009.

[4] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh: "Energy-Aware High Performance Computing with Graphic Processing Units," USENIX Workshop on Power Aware Computing and Systems (HotPower '08), Poster.

[5] S. Collange, D. Defour and A. Tisserand: "Power Consumption of GPUs from a Software Perspective," International Conference on Computational Science (ICCS 2009), Lecture Notes on Computer Science 5544, pp. 914–923, 2009.

[6] T. Nishikawa and T. Aoki, personal communication, 2008.

[7] C. Whaley et al., "Automatically Tuned Linear Algebra Software (ATLAS)," `http://math-atlas.sourceforge.net/` [accessed June 29, 2009].

[8] NVIDIA: CUDA CUBLAS Library, 2008.

[9] D. Takahashi, "FFTE: A Fast Fourier Transform Package," `http://www.ffte.jp/` [accessed June 28, 2009].

[10] H. Kotakemori et al., "Lis: a Library of Iterative Solvers for Linear Systems," `http://www.ssisc.org/lis/index.en.html` [accessed June 28, 2009].