J Electr Eng Technol.2018; 13(1): 181-191
http://doi.org/10.5370/JEET.2018.13.1.181

ISSN(Print) 1975-0102
ISSN(Online) 2093-7423

# Power Modeling Approach for GPU Source Program

**Junke Li*, Bing Guo†, Yan Shen***, Deguang Li** and Yanhui Huang****

**Abstract** – Rapid development of information technology makes our environment become smarter and massive high performance computers are providing powerful computing for that. Graphics Processing Unit (GPU) as a typical high performance component is being widely used for both graphics and general-purpose applications. Although it can greatly improve computing power, it also delivers significant power consumption and need sufficient power supplies. To make high performance computing more sustainable, the important step is to measure it. Current power technologies for GPU have some drawbacks, such as they are not applicable for power estimation at the early stage. In this article, we present a novel power technology to correlate power consumption and the characteristics at the programmer perspective, and then to estimate power consumption of source program without pre-running. We conduct experiments on Nvidia's GT740 platform; the results show that our power model is more accurately than regression model and has an average error of 2.34% and the maximum error of 9.65%.

**Keywords**: Characteristics of source program, Correlation, Estimation, High performance computing, Power modelling

## 1. Introduction

Due to rapid development of information technology, our life becomes more and smarter. Smart city, smart home, smart geosciences and smart computing are emerging in our life, which offer us better living with better resources. To make our environment smart, collecting and processing increasing data are the first thing to do. These smart environments and increasing scale of data largely depend on the high performance computing and need higher requirement on computing speed than ever before. Before 2003, performance improvement mainly depends on increasing the frequency of processor. After the appearance of power wall, this trend stops. In order to better improve the performance, a variety of techniques have been proposed to address this problem, such as vector instructions, multi-core and hyper-threading. Among them, vectorization and multi-core are considered as key technologies to lead the future development of computer. At present, the processor has shifted from simply increasing processor frequency to multi-core for gaining continued performance improvement. Under this trend, the representative of heterogeneous multi-core architecture, such as graphic processing unit (GPU), comes into being. Current GPU has the feature of large number of parallel processing cores, less control logic and higher peak performance compared with that of CPU, which makes it be widely used in general purpose computing. Top 10 in Top500 list of 2015 are using heterogeneous systems [1]. Although heterogeneous multi-core GPU has higher performance, its power consumption is also generally higher. For example, the thermal design power (TDP) of Nvidia's Tesla GPUs is about 250Watt, while the TDP of current high-end quad core CPU is about 130Watt. The power of TianHe2 is up to 17.81 MW and electricity cost of working one hour is up to more than ten millions [1]. The problem caused by power consumption, such as rising cost, increasing the probability of IC's (integrated circuit) invalidation under high temperature (if temperature increase every 10 degrees, the system failure rate will be doubled typically), decreasing of the system reliability, has become the important obstacle that blocks the performance improvement. Therefore, it has important significance to build an effective mechanism to evaluate and understand the power consumption.

In researching power-related problems, acquiring power data of the research object is the basis for subsequent work. Currently, getting the power data can be divided into two ways, direct way and indirect way. The former way obtains the power data through the integrated circuit or multimeter, which needs additional hardware circuit and susceptible to environment. The latter way uses the relation between the power and the hardware performance counters or hardware events to estimate the power consumption of program, or uses the simulator to predict the energy. Usually, using simulator to acquire the energy data of program is used to evaluate the advantages and disadvantages of the system structure, to get the detail power consumption of each part

† Corresponding Author: College of Computer Science, Sichuan University, China. (guobing@scu.edu.cn)
* College of Computer Science, Sichuan University, China/School of Computer and Information, Qiannan Normal University for Nationalities, China. (ljk2006ljk@163.com)
** College of Computer Science, Sichuan University, China. (lideguang.00@163.com, huangyanhui@scu.edu.cn)
*** School of Control Engineering, Chengdu University of Information Technology, China. (sheny@cuit.edu.cn)
Received: March 5, 2017; Accepted: August 23, 2017



Copyright ⓒ The Korean Institute of Electrical Engineers
This is an Open-Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

or to save costs. The power model based on the hardware counters or events also show some deficiencies, such as complex modeling process, poor portability and pre-running. Currently, the field of GPU power modelling lacks power estimation model before program runs. Therefore, this article proposes an application independent and low cost power model based on the source program and compiler without pre-running the program. Our approach consists of the following steps.

***Program profiling and feature extraction***

We select typical programs from classical GPU benchmark suites that are written with CUDA and then run them on typical platform to extract characteristics of hardware resource used by program.

***Power measurement***

We got the power consumption from HIOKI 3334 AC/ DC power meter.

***Statistical analysis***

After getting characteristics and power of each sample, we first use neural network to model the relation between power consumption and characteristics; then compare it with the multiple linear regression (MLR) model.

***Verification***

Through the two models, we verify their accuracy using leave-one-out cross validation (LOOCV).

This paper makes the following contributions. First, we show the method of calculating utilization of hardware resource. Second, we propose a power estimation model from perspective view of programmers to analyze and predict GPU power dissipation. To the best of our knowledge, our model that uses the feature of source program and compiler without pre-running is first proposed.

The rest of paper is structured as follows. Section 2 introduces the architecture of GPU; in section 3, related work is illustrated; section 4 presents our proposed power model; section 5 details method of calculating utilization of hardware resource. The proposed model is detailed in section 6 and then it is compared and verified in section 7. Finally, we conclude our work in Section 8.

## 2. GPU Architecture

A large number of papers [2-7] show that GPU power consumption is closely related with the consumed resource. So, it is important to clearly detail the processing unit and memory organization of GPU that impact the power consumption and execution.

GPU is connected to the CPU as a co-processor through the PCI-E bus. When the program will be running on GPU, CPU will prepare and copy processing data to GPU, then invoke the kernel which is the program running on GPU. In GPU, multiple streaming processors (short for SPs) will be grouped into streaming multiprocessors (short for SMs). The language to parallelize the program on Nvidia's GPU is the compute unified device architecture (CUDA) which uses C-liked fashion. In it, CPU and GPU are respectively called host and device (co-processor). Programmer can parallelize the program using above CUDA into three levels. In the highest level, a kernel can be scheduled by host to create a single grid that runs on GPU. The multiple GPUs can execute paralleling kernel simultaneously. Second, each grid has some thread blocks which can be specified as three-dimensional array. Third, each thread block also has some threads of three-dimensional structure. One or more thread blocks can be scheduled independently by the SM based on the resource requirement of kernel. In each thread block, parallel threads will be grouped into 32-thread which is a warp. In SP, the smallest scheduling execution unit is warp. Once the warp is executed, a half warp will be scheduled to all SPs in the SM. In SM, multiple warps can be simultaneously scheduled whether they are in the same or different thread blocks. This scheduling is limited by the available hardware resource in the SM. When the warp is ready and can meet the hardware resource, active warp will be assigned to the SPs for execution. This is the same for the block. There is no performance penalty in switching warps and blocks; on the contrary, more active warps and active blocks can effective hide the compute or memory access latency.

Memory hierarchy also has some levels in GPU. The global memory which is also called device memory is located off chip and has high latency. It can be accessed by all thread blocks in the grid. Shared memory is a high-speed memory and is located on chip. It can be read or wrote by all threads in the same block. Each SM has the fast on chip registers that can be accessed by all thread blocks. The off chip local memory can also be accessed by all thread blocks. Due to its high latency, it is used in the certain function. Constant and the texture memory are located off chip and are used for read-only data, and it is used as cache for quick accessing.

For the hardware resource constraint, limited numbers of thread blocks and threads can be scheduled on each SM.

**Table 1.** Resource specified by programmer

| Resource defined by program | Abbreviation |
|---|---|
| Threads per block | $N_{threads\_block}$ |
| Registers per thread | $N_{regs\_thread}$ |
| Shared Memory per block | $N_{smem\_block}$ |

**Table 2.** Hardware resource limitation

| Hardware Limitation | Abbreviation |
|---|---|
| Max # of blocks per SM | $H_{blocks\_SM}$ |
| Max # of threads per warp | $H_{threads\_warp}$ |
| Max # of threads per SM | $H_{threads\_SM}$ |
| Max # of registers | $H_{regs}$ |
| Max # of registers per block | $H_{regs\_block}$ |
| Max shared mem size per block | $H_{smem\_block}$ |
| Total global memory size | $H_{total\_gmem}$ |

Table 1 shows the resource specified by the programmer. Threads per block are the number of threads within each block, and registers per thread are the number of registers within each thread and shared memory per block is the size of shared memory assigned to each block. Table 2 represents the hardware constraint. Max # of blocks per SM are the maximum number of blocks can be assigned to each SM. This is the same meaning for max # of warps per SM, max # of threads per warp, max # of threads per SM, max # of registers per block and max # of registers per thread. Max # of registers is the maximum number of registers in GPU. Total shared memory size and total global memory size are the size of shared memory and the size of global memory in GPU. For the granularity constraint in allocating the register and warp resource, *Allocate_G*(*reg*) and *Allocate_G*(*warp*) are used to respectively express them. For the size constraint in allocating the register and shared memory resource, *Allocate_S*(*reg*) and *Allocate_S*(*smem*) are used to respectively express them.

## 3. Related works

Currently, more and more programmers and software developers use the modern GPU to run massively parallel application extensively. Although there are many researches [8-12, 31, 32] for analyzing and improving the performance of GPU, works on energy measuring approaches in energy problem that hindering the performance improvement and development are limited. Fig. 1 shows approaches of measuring the power of GPU. Among these approaches, build in on-broad power sensor is considered to be promising, but not all GPUs are integrated with them. External power instrument [4, 13] can provide accurate power data, but it needs extra power equipment and reserving the measuring point on GPU.

The simulation way can provide user with a more in-depth understanding of GPU hardware on energy consumption. [14] presents a powered tool and proposes an architectural power estimation framework primarily for GPU designers. [15] changes the configuration of CPU power simulator, McPAT, to make it can get the GPU power. It use multivariate linear regression method to build the model and authors use empirical data to achieve the parameter of model. Like [15], the approach of [16] is also use the linear regression to get the model. The difference is that [15] focuses more on the methodology of developing power models, whereas the [16] focuses on the GPU power model itself. [17] and [18] use the GPU-PowerSim to evaluate the power of GPU register files, which also uses McPAT. Like McPAT, GPUWatch is also used by [28] to get the power data. Although using the simulation can get the power, it usually has the problem of depending on architectural parameter, which is time consuming and cannot be widely used in the program scheduling.

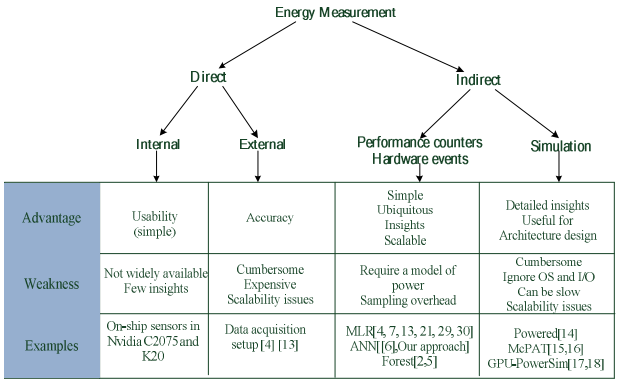


**Fig. 1.** Different Approaches of energy measurement[6]

It is feasible to use the performance counter to get the program or system energy consumption without power sensor and simulation method. Utilizing the counters to measure the power is first proposed by [19]. Subsequently, the method of measuring power using counters is continuously put forward. The best available GPU power model using hardware performance counters relies on statistics to correlate power to performance. [13] estimates power consumption using the proportion of computational instruction to total instruction from PTX code. [4] proposes a statistical power model based on linear regression to predict the energy consumption of GPU using the performance counters and its value is got from the CUDA profiler. Like [4, 7] propose pTop tool, which construct energy models for the main components (CPU, network, hard disk) of the computer system using the linear relationship between the power and the clock speed and then to acquire real-time power profiles. [29] uses multivariate linear regression to get the power. [30] also uses linear regression models to estimate the power of GPU programs. [6] extracts 10 performance events and uses back propagation artificial neural network to get the power consumption of GPU. [2] builts a high level power consumption model using a tree-based random forest method based on the performance variables and demonstrates that it can achieve better accuracy than regression-based methods. [5] also utilizes random forest methods with the profiling counters for AMD GPUs and analyzes the power consumption along with performance. [20] and [21] predict the GPU power consumption purely based on the GPU utilization. [20] uses utilization of the various GPU parts to build the power model. These parts include floating point unit, register file, ALUs and active degree of these parts is calculated by [25]. [21] dynamically predicts the runtime power of Nvidia GeForce 8800 using recorded power data and a trained statistical model. [22] proposes power model based on support vector regression using the counters from perfkit tool to estimate the GPU power. However, the purpose of perfkit tool is to debug the Opencl and direct3D application, so this model is more suitable for graphic application. Like [22], [23] is also used to predict the power of graphic application,

which adopts a different approach that builts the energy and power model from the unit energy consumed by each instruction and demonstrates that processing of geometry, fragments, and game logic consumes the most power in the pipeline.

Although past studies employing the above method have a very small error because they built the model from empirical data obtained from existing hardware, these methods are not applicable for power estimation at the early stage. Our work also utilizes empirical data, but it can give programmer an easy way to get the power and can be used for power-aware scheduling without pre-running.

## 4. Power model

$$E = \int_{t_1}^{t_2} P(t) \times dt = \overline{P} \times \Delta t = f(C_i) \times T, i = 1, 2 \cdots, n \quad (1)$$

The energy model can be expressed as Eq. (1). Where, $E$ is the energy consumption of program; $P(t)$ is the transient power. $\overline{P}$ represents the average power of program. $T$ is the running time of program. $C_i$ is the power characteristics. $\overline{P} = f(C_i)$ is called the power-characteristics correlation model(short for power model). It shows that $\overline{P}$ is the function of $C_i$. How to determine the power function f and how to get $C_i$ are the main research of source program power analysis. $\overline{P}$ is to be considered using characteristics got from source program and compiler and we can further get the following model based on Eq.(1).

$$\overline{P} = f(C_i) = f(O_s, R_{reg}, R_{smem}, R_{glob}, CTMR) \quad (2)$$

where, $O_s$ is the SM occupancy calculated by the SM resources used by source program and the total SM resource in GPU. $R_{reg}$ is the utilization of register. $R_{glob}$ is the utilization of global memory. $R_{smem}$ is the utilization of shared memory. $CTMR$ is the ratio of computation cycles to memory cycles. From the reason of causing power consumption [3, 4, 6, 15, 16, 20, 23, 24], a source program utilizing the hardware resources, such as processing unit and memory, are the most direct basis for determining the power consumption. In programming, how to use above hardware resource by programmer can directly impact the performance and the power consumption. Therefore, five characteristics of typical hardware resources are extracted as the parameters of the power model.

The procedures of establishing function $f$ to get accurate power model of source program are as follows.

1) Accurately analyze and measure the characteristics of the source program;

2) Get the $E$ and $T$ though the power meter and time function. Based on the Eq.(1), the value of $\overline{P}$ by

$$\overline{P} = E / T \quad (3)$$

3) This article assumes that five characteristics have the nonlinear relationship with the power(linear relationship can be seen as a special nonlinear function). For the back propagation (BP) artificial neural network can approximate any nonlinear function with high accuracy and can obtain the satisfying result, so this paper adopt the BP to approximate function $f$ after getting the input and output value. In order to verify the correctness of the assumptions, we also compare power consumption data got by the MLR with results obtained by BP.

## 5. Measurement of Characteristics

Characteristics of the source program can reflect the utilization of the hardware resources and then show the power consumption. The research scope of using source program and compiler to estimate power is to analyze the resources (processing resources and storage resources) occupied by the source program, and the influence of source program characteristics on power consumption.

### 5.1 Measurement of SM occupancy

SM occupancy is an important characteristic to measure the hardware resources used by source program. A large number of articles [2, 6, 15, 17, 20] indicate that it has a direct impact on the power consumption. Therefore, SM occupancy is chosen as a characteristic to achieve the power of source program. At present, the value of SM occupancy can be calculated by the dimensions defined by the kernel in the program. It can be got from Eq.(4).

$$O_s = warp_{active} / warp_{total} \quad (4)$$

Where, $Warp_{active}$ is the number of active warps in SM. $Warp_{total}$ is the maximum number of allowed active warp in SM. We can use nVIDIA's occupancy calculator tool to get this value.

### 5.2 Utilization of storage resource

Like SM occupancy, utilization of storage resources is also an important characteristic to reflect energy consumption of source program. Currently, measurement of storage resources utilization includes following aspect.

***Utilization of registers***

Register is an on-chip cache and it is widely used by programmers for its low access latency. The utilization of register resource has an impact on the energy consumption of GPU, so the utilization of registers can reflect the energy consumption of program. Utilization of registers can be got through Eq. (5).

$$R_{regs} = (N_{regs\_thread} * N_{threads\_warp} * warp_{active}) / H_{regs\_Block} \quad (5)$$

***Utilization of shared memory***

From researches [2-4, 15, 16, 20, 23], we can know that the power consumption generated by shared memory also cannot be ignored. The more shared memory used in the source program, the higher potential power consumption is. In this article, we use Eq.(6) to express it.

$$R_{Smem} = (N_{smem\_block} * Block_{active}) / H_{smem\_block} \quad (6)$$

***Utilization of global memory***

When the program is running, the data will be transferred between the main memory (usually called memory) and GPU memory. Like the shared memory and register, the utilization of global memory will also impact the energy consumption. In order to be able to estimate the power consumption of the source program, we also use utilization of global memory to describe the memory used by source program. Utilization of global memory is represented by Eq.(7).

$$R_g = N_{global\_memory} / H_{total\_gmem} \quad (7)$$

Where, $N_{global\_memory}$ is the size of global memory used by source program. $H_{total\_gmem}$ represents the total size of global memory in GPU.

This article emphasizes on measuring the space that program variables used. For a structural variable, the space occupied by each of its members can be accumulated and then we can get the required size of the variable in memory. Similarly, we can get the space required by an array of variables using the length of the array to multiply the space required by a single array element. As an *int* array of *A*[*rows*] [*cols*], if using the Visual C++ Microsoft compiler, the *int* data type occupied by the number of bytes is 4 and the array *A* takes up 4 × *rows* × *cols* bytes. The space of program code is usually determined by the compiler, and the compiler can provide various optimization options, such as code optimization, execution time optimization, so the space of binary code got by the different compilers or the same compiler on the same piece of code using different compilation and optimization is generally not the same size. Meanwhile, program drives the processor running, so this article uses the characteristic of *CTMR* to demonstrate the power consumption produced by the program code. In this article, utilization of global memory only contains the size of BBS, data, heap, and stack.

### 5.3 Ratio of computation cycles to memory cycles

Except the characteristics of SM resources and storage resources, the source program emphasize on SM or memory will also have an impact on the power consumption. For example, the program of emphasizing on SM and the program of emphasizing on memory has the different power behavior [4, 21, 22, 25]. Therefore, it is particularly essential to measure the degree of the source program emphasizing on which component and then reflects the power consumption of source program. [24] uses a rule of thumb named "ratio of computation cycles to memory cycles" to measure the density of memory access during program running. In this article, we take the *CTMR* characteristic as the feature of power behavior .*CTMR* value can be obtained by Eq. (8).

$$CTMR = \frac{Number(Computation\ Instructions\_cycles)}{Number(Global\ Memory\ Transactions\_cycles)} \quad (8)$$

Where, *Number* (*Computation Instructions_cycles*) and *Number* (*Global Memory Transactions_cycles*) respectively indicates time spent by executing computations instruction and the time spent by executing memory access. For calculating it, we get it through the way of compiler-assisted and program analysis model proposed in [25].

## 6. Nonlinear fitting of power consumption

To rational express nonlinear relationship of characteristics and the power consumption of GPU can estimate the power consumption of source program, so how to present the nonlinear relationship is important. The BP neural network is a kind of numerical approximation method without establishing mathematical equation; it can approximate any nonlinear function and has good fitting ability through learning the input vector and output vector [26]. To make BP achieve the best fitting effect, the structure of BP should be firstly determined (number of hidden layer, number of nodes in hidden layer, transfer function of each layer) .What's more, approximation error, convergence speed and learning rate of neural network are the factors that also need to be considered.

The number of hidden layer and the number of nodes in hidden layer will affect the prediction accuracy of network. Robert Hecht-Nielson in [26] proves that BP network of single hidden layer can approximate continuous function in

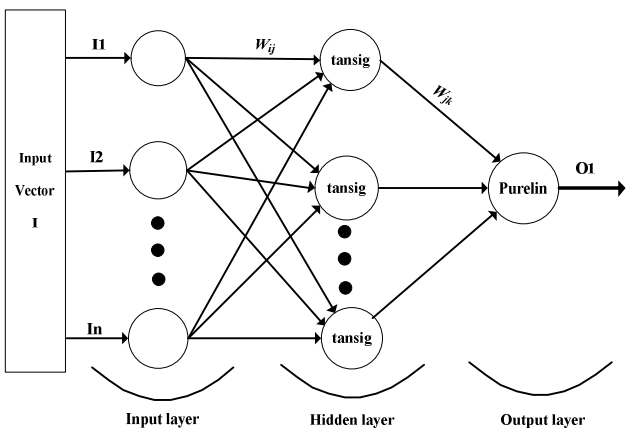


**Fig. 2.** Structure of BP neural network

any closed interval. The number of nodes in hidden layer should be also considered, but it lacks the guidance of the scientific method. Usually, the best numbers of nodes $l$ is got by Eq. (9).

$$l \le \sqrt{n+m} + a \tag{9}$$

In Eq. (9), $l$, $n$ and $m$ are respectively the number of nodes in hidden layer, the number of nodes in input layer and the number of nodes in output layer. $a$ is a constant number between 0 ~ 10. In this paper, BP has five inputs and one output (power), so the scope of $l$ is: 3 ~ 13.

The structure of the BP is shown in Fig. 2; where, $I$ and $O$ are respectively the input vector and the output vector; $w_{ij}$ and $w_{jk}$ are respectively the weight between input layer and hidden layer and the weight between hidden layer and output layer. The transfer function of hidden layer and output layer are got from experiment to get the best performance. From the experiment, the *tansig* and *purelin* in hidden layer and output layer can achieve satisfactory results in convergence speed and error.

## 7. Experiment validation and analysis

In this paper, all of experiments are conducted on Intel i5-3230M quad-core processors (8 cores in total) and Nvidia's GT740 platform which is the Kepler architecture and it consists of two SMs and 2GB DRAM memory. Each SM contains 192 CUDA cores. The programming environment of GPU is CUDA6.5. To demonstrate effectiveness of our algorithm, we select 44 typical benchmarks from CUDA SDK to conduct typical experiments, such as *BlackScholes, fastWalshTransform*, *matrixMul, sortingNetworks*, etc, which are widely adopted by the existing works. For the program from CUDA SDK cannot reflect the influence of single characteristic variation on power consumption, we also modify some program to change the above characteristics to validate our proposed approach. The *Vecadd* program and the *scalaprod* program are modified to fully reflect the power model proposed in this paper. To measure the power, we use HIOKI 3334 AC/DC power meter to get the power of GT740 when the instances are running. We first measure the idle power of entire system (Pa). We also get the power consumption of whole system when a GPU application is running (Pb). Then, we get the CPU idle power by turning off the GPU module and rendering the CPU into an idle state(Pc). With the GPU module still off, we measure the power dissipation when running the GPU application with CUDA-related function calls removed (Pd); this value gives the CPU power. Consequently, Pa-Pc is the GPU idle power and Pb-Pd-(Pa-Pc) equals the GPU runtime power. To reduce error, we adopt the way of executing programs many times to get average power consumption. The compute capability of experimental platform is 3.5, so the *Allocate_G*(*reg*) is warp, *Allocate_G*(*warp*) is 4, *Allocate_S*(*reg*) is 256 and *Allocate_S*(*smem*) is 256. After getting the characteristics of source program and its power data, we use neural network tool in Matlab R2013a to build the BP, and adjust the weights and threshold in all layers to make the mean square error (MSE) meet the desired goal.

Fig. 3 shows the comparison between measured power and estimated power using our approach. For calculating error, we use leave-one-out cross-validation (LOOCV) method. It uses a single program from the 43 program as a validation data, and the remaining data as the train data. Program 1-12 is the benchmark form CUDA SDK; 13-22 is the modified *Vecadd* program; each different label behind experimental number indicates different *CTMR* value;23-25 is the modified *Vecadd* program to change utilization of global memory; each label behind experimental number indicates how to change utilization of global memory; 26-30 is the modified *Vecadd* program to change SM occupancy; each different label indicates the value of the SM occupancy;31-37 is the modified *Vecadd* program to change utilization of register; each different label behind experimental number indicates the value of register

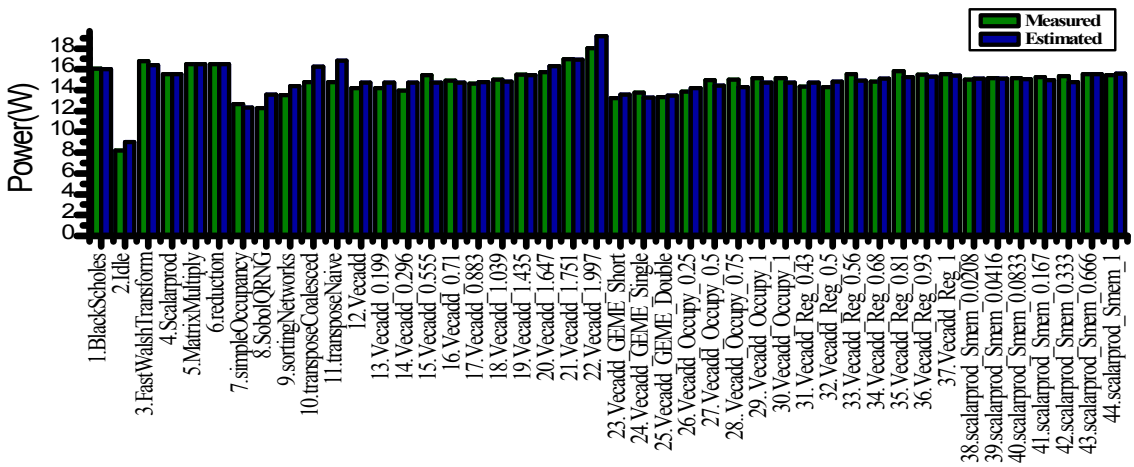


**Fig. 3.** Comparison between estimated and measured power

utilization; 38-44 is the modified *scalarprod* program to change utilization of shared memory; each different label behind experimental number indicates different shared memory utilization. From fig. 3, we can see that majority of the programs show high accuracy in their power prediction. In the experiment, the error is within 10%. The average error is 2.34% and the maximum error is 9.65%. Like most study, the accuracy of our proposed approach is under acceptance.

To validate relationship between the independent characteristics and dependent power consumption, we also do the experiments on individual characteristic to analyze power consumption. Figs. 4-7 and Fig. 8 respectively show the dependence plots for the utilization of register, SM occupancy, utilization of shared memory, utilization of global memory and *CMTR*. Fig. 8 shows the increase of power generally goes along with the variable ranking. This demonstrates that the power consumption has the similar direct correction with the *CTMR*. SM occupancy in fig. 5 shows that given higher occupancy, the more power is consumed. Given the fact that giving more thread blocks to hide memory or computation latency can generally make the execution fast, high occupancy consumes more power. The phenomenon also appears on utilization of shared memory in fig.6. However, at the highest utilization of shared memory the power decreases a little. This is because there are not enough active warps in the pipeline, and this may result in memory or computation latency and then slow down the instruction executed. In the experiment, we find that adjusting the utilization of register and utilization of global memory will affect the *CTMR* characteristic. These facts are shown in fig. 4 and fig. 7. From fig. 4, we can see the higher the register utilization, the bigger the value of *CTMR*. With the utilization of register enhancement, the power also increases. The experiment shows adjusting the utilization of register will change the *CTMR* and then these two characteristics jointly affect the power. Fig. 7 illustrates the relationship between utilization of global memory and the power consumption. To show how the utilization of global memory affects the power consumption, the program in each experiment is the same and we only change the data type. As we can see from the fig. 7, when the global memory utilization is changed, the *CTMR* characteristic is changed too. The two changed characteristics then make power consumption vary. From the above analysis, we can know these characteristics have

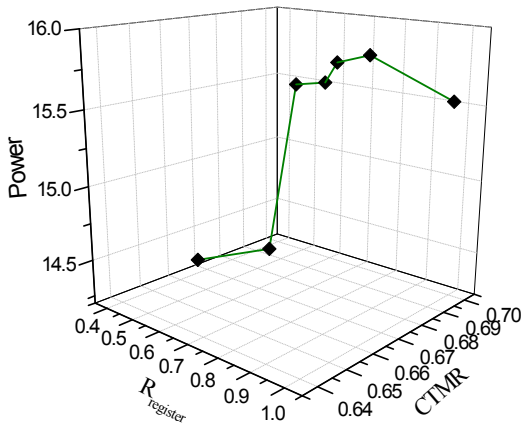


**Fig. 4.** Relationship between $R_{reg}$ and the power

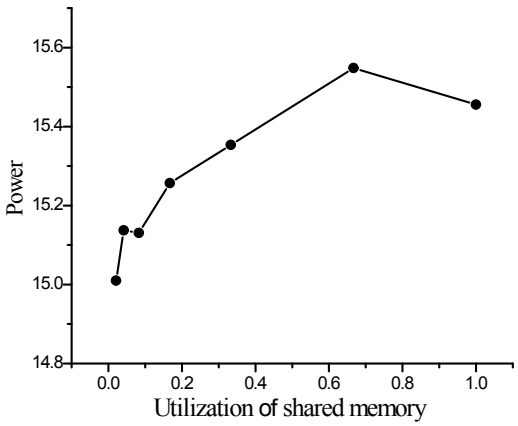


**Fig. 6.** Relationship between $R_{smem}$ and the power

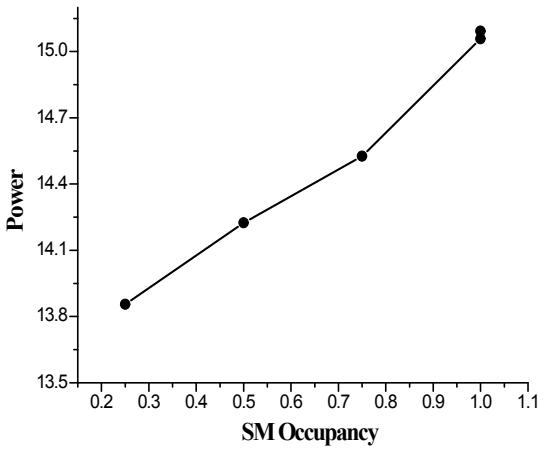


**Fig. 5.** Relationship between $O_s$ and the power

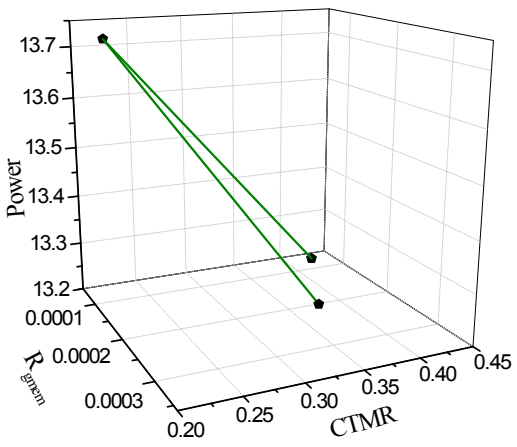


**Fig. 7.** Relationship between $R_{gmem}$ and the power

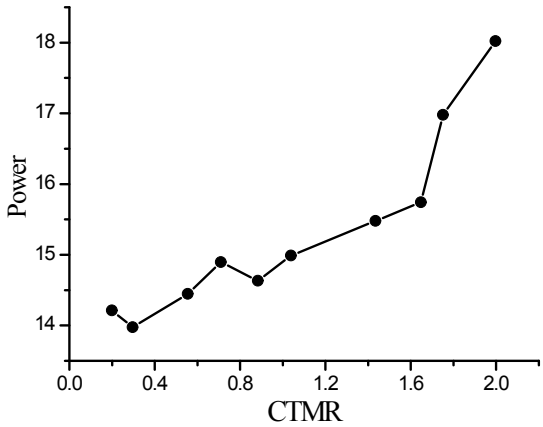


**Fig. 8.** Relationship between *CTMR* and the power

correction with the power consumption and they can be used in the power estimation.

We also conduct several experiments to verify the divergence impact on the total power consumption with different input. For reduction benchmark has six implementations with different control flow conditions and memory access pattern, so these are selected as the verification using different input size. Reduction1 version uses contiguous threads, but its interleaved addressing results in many shared memory bank conflicts. Reduction2 version uses sequential addressing which are no bank conflicts. Reduction3 version uses n/2 threads and performs the first level of reduction when reading from global memory. Reducion4 version uses the warp shuffle operation if available to reduce warp synchronization. Reduction5 version is completely unrolled. Reduction6 version adds multiple elements per thread sequentially. This reduces the overall cost of the algorithm while keeping the work complexity $O$(n) and the step complexity $O$(log n). Reduction1 and reduction2 have control flow conditions for reading data from global memory and performing reduction operation. Reductions3 performs a reduction operation when reading the data from global memory and does the whole reduction operation using control flow conditions. Reduction4 also performs a reduction operation when reading the data from global memory and do the warp shuffle operation using control flow conditions. Reduction5 performs the unroll operation using the using control flow conditions. Reduction6 reduces multiple elements per thread which is determined by the number of active thread blocks and does warp shuffle operation using control flow conditions. Three input size of each experiment are respectively 2M, 32M and 128M. The compared experiment of three input size and six reduction versions is illustrated in fig. 9. From it, we can see the estimated power can reflect the measured power with maximum error of 9.88% and average error of 2.61%. This phenomenon demonstrates that our proposed approach can reflect the power consumption of different control flow conditions. Except the different input size influences the

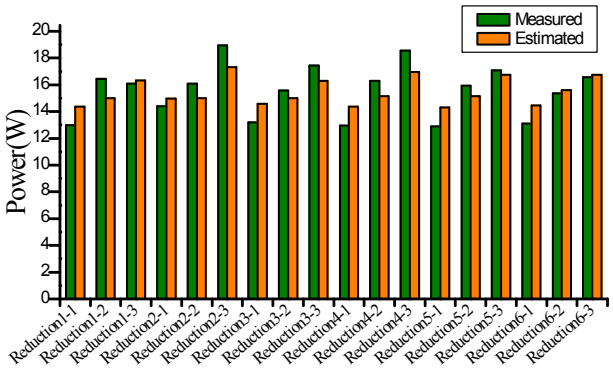


**Fig. 9**. Power comparisons under different divergence and input size

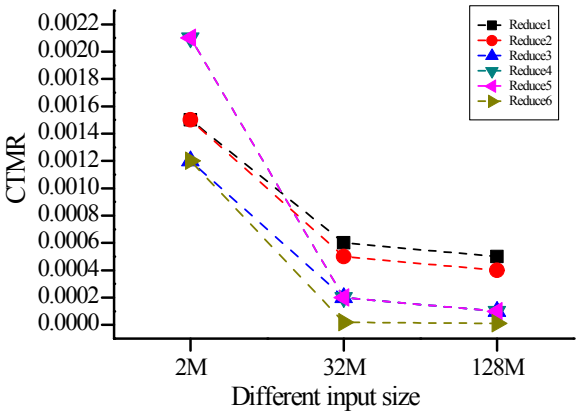


**Fig. 10.** Different input affect characteristics

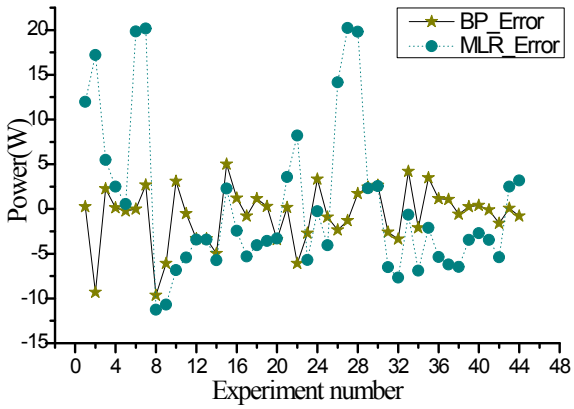


**Fig. 11.** Power error under different approaches

$R_{gmem}$ and $R_{smem}$, it can also affects other characteristic. Fig. 10 shows the different input and divergence affect the *CTMR* characteristic. From it, we can see that with the input size increasing, the *CTMR* is decreasing. The reason is that large input size is given, more instructions will be handled the divergence.

Among the approaches of power estimation, many articles use the MLR [4, 7, 13, 21] to predict the power consumption. Therefore, this paper also compares the

accuracy between the MLR and our proposed power model. Fig. 11 shows error between the MLR and our proposed model. From the figure, we can see the maximum estimated error is 20.231% and average error is 6.56% got by MLR. Comparison between estimated power got by MLR and our proposed model, the latter has some advantages compared with the MLR.

## 8. Conclusion

Energy consumption in the high-end GPU requires expensive power supply and cooling system. In this article, we propose a power estimation model of GPU source program to predict the energy consumption without pre-running the program. By extracting the characteristics of source program, we use BP neural network to present the model and then get the power data of source program. Experiments show that our model is effective. Our proposed novel method can provide basic power data for further research. For example, we can use this model to schedule kernel for energy saving. Given the GPU performance model, we can also do the energy-efficient scheduling. Furthermore, using this model, we can guide the programmer to optimize the energy consumption without pre-running.

## Acknowledgements

This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant No.61332001; the National Natural Science Foundation of China under Grant No. 61272104 and 61472050; the Science and Technology Planning Project of Sichuan Province under Grant No. 2014 JY0257, 2015GZ0103 and 2014-HM01-00326-SF; the Science and Technology Foundation of Guizhou Province under Grant NO.LH20147422. We also acknowledge group of embedded real-time system for their effective advice and constructive suggestions, and the reviewers who provided helpful suggestions which have improved the manuscript.

## References

[1] Top 500 Supercomputer Sites Webpage, November 2015. http://www.top500.org.

[2] Chen, J., Li, B., Zhang, Y., "Tree structured analysis on GPU power study," *in Proceedings of IEEE International Conference on Computer Design*, pp. 57-64, 2011.

[3] Ma, X., et al, "Statistical power consumption analysis and modeling for GPU-based computing," *in Proceedings of ACM SOSP Workshop on Power Aware Computing and Systems*, October 2009.

[4] Nagasaka, H., Maruyama, N., et al, "Statistical power modeling of GPU kernels using performance counters," *in Proceedings of IEEE International Conference on Green Computing*, pp. 115-122, 2010.

[5] Zhang, Y., Hu, Y., et al, "Performance and power analysis of ATI gpu: A statistical approach," *in Proceedings of IEEE International Conference on NAS*, pp. 115-122, 2011.

[6] Song, S., Su, C., et al, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," *in Proceedings of IEEE International Symposium on IPDPS*, pp. 676-686, 2013.

[7] Chen, H., Li, Y., & Shi, W., "Fine-grained power management using process-level profiling," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 1, pp. 33-42, 2012.

[8] Li, K., Yang, W., Li, K, "Performance analysis and optimization for SpMV on GPU using probabilistic modeling," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 196-205, 2015.

[9] Kreutzer, M., Pieper, A., et al, "Performance Engineering of the Kernel Polynomal Method on Large-Scale CPU-GPU Systems," *in Proceedings of IEEE International Conference on IPDPS*, pp. 417-426, 2015.

[10] Chitty, D. M., "Improving the performance of GPU-based genetic programming through exploitation of on-chip memory," *Soft Computing*, vol. 20, no. 2, pp. 661-680, 2016.

[11] Dastgeer, U., Kessler, C., "Performance-aware composition framework for GPU-based systems," *The Journal of Supercomputing*, vol. 71, no. 12, pp. 4646-4662, 2015.

[12] Angerer, C. M., et al, "A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware," *Sustainable Computing: Informatics and Systems*, vol. 12, pp. 72-82, 2015.

[13] Luo, C., Suda, R., "A performance and energy consumption analytical model for GPU," *in Proceedings of IEEE International Conference on DASC*, pp. 658-665, 2011.

[14] Ramani, K., Ibrahim, A., Shimizu, D., "PowerRed: A flexible modeling framework for power efficiency exploration in GPUs," *in Proceedings of the Workshop on General Purpose Processing on GPUs*, October 2007.

[15] Lim, J., et al, "Power modeling for GPU architectures using McPAT," *ACM Trans. on Design Automation of Electronic Systems*, vol. 19, no. 3, pp. 26, 2014.

[16] Leng, J., Hetherington, T., et al, "GPU Wattch: enabling energy optimizations in GPGPUs," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, ACM, 2013.

[17] N. Goswami, Cao, B., Li, T., "Power-performance co-optimization of throughput core architecture using

resistive memory," *in Proceedings of IEEE International Conference on HPCA*, pp. 342-353, 2013.
[18] N. Goswami, A. Verma, and T. Li, Gpu-powersim, 2012.http://www.ideal.ece.ufl.edu/main.php?action=gpupowersim.
[19] Tiwari, V., Malik, S., Wolfe, A., "Power analysis of embedded software: a first step towards software power minimization," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 2, no. 4, pp. 437-445, 1994.
[20] Hong, S., Kim, H., "An integrated GPU power and performance model," A*CM SIGARCH Computer Architecture News*, vol. 38, no. 3, ACM, 2010.
[21] Kim, Y. G., Kim, M., Kim, et al, "A novel GPU power model for accurate smartphone power breakdown," *ETRI Journal*, vol. 37, no. 1, pp. 157-164, 2015.
[22] Ma, X., Dong, M., et al, "Statistical power consumption analysis and modeling for GPU-based computing," *in Proceedings of ACM SOSP Workshop on Power Aware Computing and Systems*, 2009.
[23] Pool, J., Lastra, A., & Singh, M., "An energy model for graphics processing units," *in Proceedings of IEEE International Conference on ICCD*, pp. 409-416, 2010.
[24] Ryoo, S., Rodrigues, C. I., et al, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," *in Proceedings of ACM SIGPLAN Symposium on Principles and practice of parallel programming*, ACM, pp. 73-82, 2008.
[25] Hong, S., Kim, H., "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *ACM SIGARCH Computer Architecture News*, vol. 37. no. 3, ACM, 2009.
[26] Sarajedini, Amir, R. Hecht-Nielson, "The best of both worlds: Casasent networks integrate multilayer perceptrons and radial basis functions," *in Proceedings of IEEE International Joint Conference on Neural Networks*, vol. 3, pp.905-910, 1992.
[27] Wu, G., Greathouse, J. L., et al, "GPGPU performance and power estimate on using machine learning," *in Proceedings of IEEE International Conference on HPCA*, pp. 564-576, 2015.
[28] Leng, J., Hetherington, T., et al, "GPUWattch: Enabling Energy Optimizations in GPGPUs," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, ACM, 2013.
[29] Bailey, P. E., Lowenthal, D. K., et al, "Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems," *in Proceedings of IEEE International Conference on ICPP*, pp. 371-380, 2014.
[30] Ma, K., Li, X., et al, "GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures," *in Proceedings of IEEE International Conference on ICPP*, pp. 48-57, 2012.
[31] Baghsorkhi, S. S., Delahaye, M., et al, "An Adaptive Performance Modeling Tool for GPU Architectures," *in Proceedings of IEEE International Conference on PPoPP*, pp. 105-114, 2010.
[32] Madougou S, Varbanescu A L, De Laat C, et al. "The landscape of GPGPU performance modeling tools," *Parallel Computing*, vol. 56, pp. 18-33, 2016.

**Junke Li** He received his BS degree in Computer Science from the Henan Polytechnic University in 2010, and he received his MS degree in Computer Science from Southwest University in 2013, he received his PHD degree in Computer Science from Sichuan University. He is currently an associate professor in the School of Computer and Information at Qiannan Normal University for Nationalities, China.

**Bing Guo** He received his BS degree in Computer Science from the Beijing Institute of Technology in China, and MS and PhD degrees in Computer Science from the University of Electronic Science and Technology of China, China, in 1991, 1999, and 2002, respectively. He is currently a professor in the School of Computer Science at the Sichuan University, China. His current research interests include embedded real-time system and green computing.

**Yan Shen** She received her MS degree in Mechatronics Engineering and PhD degree in Measuring and Testing Technology and Instruments from University of Electronic Science and Technology of China in 2001 and 2004 respectively. Currently she is a professor in the Control Engineering College, Chengdu University of Information and Technology. Her main research interests include distributed measurement systems, embedded system development, wireless sensor networks, robotics.

**Deguang Li** He received his BS degree in Computer Science from the PLA Information Engineering University, in 2010, and he received his MS degree in Computer Science from Northeastern University, in 2012. He is currently a PhD candidate in the

School of Computer Science, Sichuan University. His research interest includes green computing.

**Yanhui Huang** He received his BS and the MS degree in Computer Science from Sichuan University in 1997 and 2002 respectively. Currently he is a lecture in the school of computer science at Sichuan University. His current research interest includes green computing.