Power Estimating Model and Analysis of General Programming on GPU

Haifeng Wang [2 3] , Qingkui Chen [1 2*]
[1]University of Shanghai for Science and Technology, School of Optical-Electrical and Computer Engineering, Shanghai, China
[2]University of ShangHai for Science and Technology, , School of Management, Shanghai, China
[3]Lin Yi University, Linyi, China
Email: gadfly7@126.com, chenqingkui@gmail.com

***Abstract*—Energy efficiency is a major concern in the General Programming on Graphic Process Unit. Recent research focus on the measurement approach and energy optimization of Graphic Process Unit. Few studies provide insight to where and how power is consumed from the program perspective. The aim of this research was to build power consumption model to estimate the energy consumption for the application programmers. Program slicing was used to decompose the programs into slice set. The program slice as basic unit was to measure and analyze the program power consumption. We consider the computation intensity and the number of active SMs that have directly impact on energy consumption. Aiming to the sparseness-branch and denseness-branch programs, two power consumption prediction models were proposed. The experimental results show that the average relative error of the two prediction models are less than 6 percent. We conclude that the power consumption prediction models can effectively estimate the energy consumption of applications.**

***Index Terms*—Power Consumption Model, GPU Computing, CUDA, Program Slicing**

I. INTRODUCTION

With Compute Unified Device Architecture (CUDA) launched by NVIDIA and the development of applications of GPU (Graphic Process Unit) in the High Performance computing field the power consumption of GPUs increase rapidly. The energy measurement and optimization for GPUs has received much attentions in recent years due to the development of Green Computing[1]. Jiao. et al. showed the different power characterizations of GPU computing and investigated the relationship between power consumption and different computational patterns under various voltage and frequency levels[2]. Shaikh et al. measured the power consumption of different instructions in GPU computing. The results provide valuable data for the GPU power optimization[3]. One attempt to improve our understanding of power and performance is the research work to propose a prediction model for GPU computing. The prediction model can optimize the power consumption by setting the optimal number of active cores to execute the computing tasks[4]. In addition, some researchers have studied the power efficiency of GPUs clustering. Jeremy et al. used one technique to measure the GPU clustering power that is a very inexpensive, non-intrusive method for GPUs clustering monitoring system in order to improve the performance-per-watt of GPU applications[5]. PowerPack is a comprehensive hardware-software framework for performing an in-depth analysis of the energy consumption of parallel applications on a multi-core systems. The PowerPack no longer measures the componet power consumption alone and considers the correlation between the power consumption of a component and the software executing on the system[6].

To improve the energy efficiency of GPU computing systems and applications, it is critical to profile the power consumption from the software perspective. The instruction level is the first level to measure and optimize the power consumption. Sylvain et al. measured the power consumption of the arithmetic operations and memory accessing operations on three typical GPU architectures[7]. The function or procedure level is the second level to analyze the processor energy consumption. Some researchers had analyzed and quantified the CPU power consumption[8]. Wang et al. proposed kernel fusion method to reduce energy consumption and improve power efficiency on GPU architecture[9]. However, measuring and analyzing power consumption at the instruction level is fine-grained approach that is difficult to apply into the practice. On the other hand, it is course-grained power analyzing approach from the function level that has poor accuracy. The program slicing between the instruction and the function level is an effective way to analyze the power consumption of GPUs from the software perspective in practice. To our best knowledge, little attention has been paid to the program slice level of granularity to analyze the power consumption model.

Our goal is to build a power consumption model for the GPU applications and allow the programmers to estimate their program's power consumption by scanning the source code. So our research work belongs to the GPU power measurement and analysis from the software perspective. To address this issure, we build the power consumption model at the program slice level. Compared to the instruction and the function levels, our approach is

doi:10.4304/jsw.7.5.1164-1170

more practical than the former. And it is a more accurate way than the one from the function level. In this paper, the programs are decomposed by the program slicing technology. The program slice as power consumption unit is to measure the whole power consumption of GPU programs. Aiming to the sparseness-branch programs and denseness-branch programs, we proposed two different power consumption models that can support the power managment and the optimization of the GPU computing.

The remainder of this paper is organized as follows: Section 2 presents the power consumption measurement scheme for GPUs and the program slicing technology. The detailed power consumption models for the two different kind programs are provided in section 3. Next, we analyze the validity of the power consumption models through the experiments in section 4. Finally summarizes our research and future work.

II. RELATION WORKS

A. Measurement Description

The power measurement is the basic problem to the power consumption model. The power supplies of High-End GPU come from the PCI-E and the extra power. We only consider the external power supply due to inaccessibility to the PCI-E interface power pin. A simple current data acquisition card was designed and used to measure the energy consumption. The probe sensor in the current acquisition card converts the measured current to corresponding voltage signals through the current sensor ACS713-20T. The microprocessor Atmega168 of the data acquisition card is responsible to convey the AC signal to DC signal. And the USB controller FTDI 232RL send the signal data to the computer system to record the variation of the power of the GPUs.

B. Program Slicing

Program slicing is a well known program analysis technique that is widely used in program comprehension, testing, debugging, re-engineering and software maintenance fields[10]. A program slice is a statements set affected given variables. So the program slicing is a program decompose technique to reduce the scope of the program analysis and understanding.

The applications of GPUs is a set of Kernel functions, namely $P = \{K_1,K_2,\ldots,K_m\}$. And the kernel function is a set of program slice. $K = \{C_1,C_2,C_i,\ldots\}$, C_i represents the ith program slice, K denotes the kernel function running on the GPUs. C_i is triple (K, S, V), K represents the kernel function, S is the statements set relating to V and V is the variable affected in S. Then we explain how to slice a Kernel function by an concrete example.

We decompose the kernel function shown in Fig.1(a) by mean of the static sclicing. A slice is denoted as C_1 = (K, S, patlen) shown in Fig.1(b). Patlen is the variable in the slice C_1 and S is the statement set affected this variable. This kernel function may be decomposed as K = (C_1,C_2,C_3), C_1 = (K, S_1, patlen), C_2 = (K, S_2, charnum) and C_3 = (K, S_3, pchar). However the slicing form is not constant. This kernel function may be decomposed in other forms. Such as, K = (C_1,C_2), C_1 = (K, S_1, patlen),C_2 = (K, S_2, charnum, pchar).

```
if ( inner_id != block_per_group -1)
{
    charnum = tex2D(x1)-tex2D(x2);
    patlen = tex2D(x2)- tex2D(x3);
    pchar = tex2D(x4) * tex2d(x1);
}
else{
  if ( threadIdx.x < patNum-inner_group)
  {
     charnum = tex2D(x4)-tex2D(x6);
     patlen = tex2D(x5)- tex2D(x7);
     pchar = tex2D(x3) - tex2d(x1);

  }
  else
  {
     patlen = tex2D(x8)- tex2D(x9);
}
```

Figure1. (a). Source code of Kernel Function

```
if ( inner_id != block_per_group -1)
{
     patlen = tex2D(x2)- tex2D(x3);
 }
else{
  if ( threadIdx.x < patNum-inner_group)
  {
     patlen = tex2D(x5)- tex2D(x7);
  }
  else
  {
     patlen = tex2D(x8)- tex2D(x9);
  }
```

Figure1. (b). Program Slice

To determine the slicing form for the kernel function, we use the variable type to divide the slices. There are global memory, shared memory, constant memory and texture memory in GPUs and denoted as GM, SHM, CM and TM respectively. The variable set in the GPU program can be grouped into four different types according to the memory locations. Regardless the register file in GPUs, the kernel function should be decomposed as a constant form, namely K = (C_{GM} ,C_{SHM},C_{CM}, C_{TM}).

Being different in the energy consumption of accessing the different memory areas, we should measure the power consumption of memory accessing slices by experiments. The NVIDIA Geforce GTX280 GPU was selected in our experiments. For the sake of simplicity, we replace the program slices with specific kernel functions that contain the identical accessing statements. In order to record the experimental results accurately, the executing time of each kernel fucntion should be above 50ms. The four memory accessing kernels denote as GM,SHM,CM and TM, As shown in Figure 2. The peak value of shared memory accessing reaches to 135W. The global memory

accessing and texture memory accessing are between 71W and 75W. In the most case the energy consumption of texture memory accessing is lower than the global memory accessing due to the cache. The constant memory energy consumption fluctuates between 65W and 68W and is the minimum value among the four memory accessing patterns.

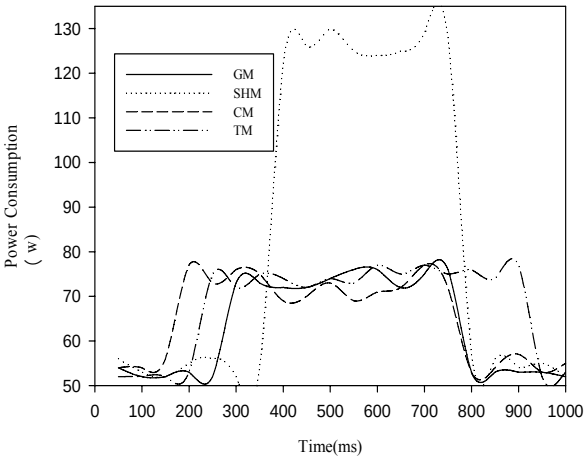

Figure 2 Energy Consumption of four Accessing Patterns

In order to normalize the four different memory accessing patterns, we unify the different memory accessing statements into unified form by mean of the proportional coefficient. The proportional coefficient for GM,SHM,CM,TM represent λ_0, λ_1, λ_2 and λ_3 respectively.. Setting the global memory accessing pattern is norm pattern, namely $\lambda_0=1$, since the global memory accessing operations occur most frequently in the GPGPU programs. Assume that a program slice C_i includes GM accessing statements x_1, SHM accessing statements x_2, CM accessing statements x_3 and TM accessing statesment x_4. Expressed in a formula, the number of unified memory accessing statements X can be written as .

$$X = \lambda_0 \ x_1 + \lambda_1 \ x_2 + \lambda_2 \ x_3 + \lambda_3 \ x_4 \qquad (1)$$

The proportional coefficients for GTX 280 obtained through the experiments are set as follows, $\lambda_1 = 1.67$, $\lambda_2 = 0.91$, $\lambda_3 = 0.95$.

C.Computation Intensity

The computation intensity was introduced to quantify the GPUs performance. It is defined by the ratio between the number of arithmetic operations and the number of memory accessing operations[11]. Here we take the power consumption as an important factor to redefine the computation intensity. Different in the power consumption incurred by the different memory accessing patterns, we don't treat the different memory accessing statements as the same one compared to the definition of computation intensity in literature[11]. So the computation intensity from the power consumption perspective can be defined as equation 2.

$$\tilde{A} = \frac{\#A}{\left(\#A+\lambda_0\sum GM + \lambda_1\sum SHM + \lambda_2\sum CM+ \lambda_3\sum TM\right)} \qquad (2)$$

Where $\tilde{A}$ denotes the computation intensity from the power consumption perspective, the number of the arithmetic statements represents #A and $\lambda_0\sum GM + \lambda_1\sum SHM + \lambda_2\sum CM+ \lambda_3\sum TM$ is the number of memory accessing statements.

III. POWER CONSUMPTION MODEL BASED ON SLICE

A. Regreesion Model of Program Slice

To build power consumption model for applications, the power consumption of program slicing should be considered and measured due to the fact that the program is composed of many slices. The computation intensity of applications changing, the power consumption will change as well. So we investigated the power consumption of slices and built a prediction model for the slices. It is difficult to directly measure the power consumption of program slices. Then we use the kernel function to substitute the program slice with the identical computation intensity. Many different computation intensity kernels are designed in the experiments, theirs executing time being greater than 50ms.

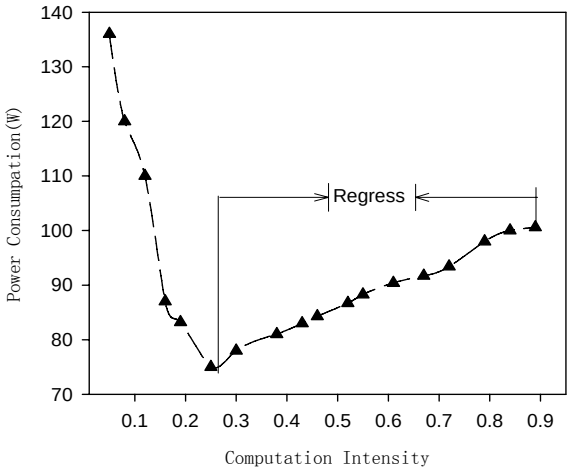

Figure 3 Variation of Power Consumption with Different Computation Intensity on GTX280

Fig.3 shows the variation of power consumption with the different computation intensity to the kernel functions. We obtain the following observations from this figure: 1) when the computation intensity is greater than 0.2, the power consumption increases proportionally. 2) when the computation intensity is less than 0.17, the power consumption drops noticeably from 138W down to 83W, indicating that these memory-intensive computation kernels include more shared memory accessing operations that consume much more energy. In general, there are few computation tasks in practice whose computation intensity are less than 0.2 because such computation tasks are unsuitable for the GPUs. So we neglect the tasks whose computation intensity are less than 0.2 and consider the computation intensity from 0.2 to 0.9 to build the power consumption model through linear regression analysis. Given the computation intensity is independent variable x and the power

consumption is dependent variable y. Then the regression function shown in Equ.3 is obtained with least square method.

$$f(x) = 69.4 + 34.5x \quad (3)$$

B. Active number of SMs

The number of active cores inside a chip, especially in GPUs, is an important factor to be considered for the energy consumption, because we can achieve speedup by launching more active cores but with increased total energy consumption. Then we investigate the correlation between the number of active stream multiprocessor and the power consumption. Currently, the number of Stream Multiprocessor (SM) in the GPUs is increasing dramatically. For example, NVIDIA GeForce GTX 280 has 30 streaming multiprocessors with 240 Stream Processors (SP). The computing performance increases with more SMs, but the power consumption increases accordingly[4]. To further improve the accuracy of the power consumption model, the number of active SMs should be considered and investigated the variation of energy consumption with changing the number of active SMs. In the experiment, the matrix multiplication whose computation intensity is 0.506 is selected to measure the power consumption with different SMs.

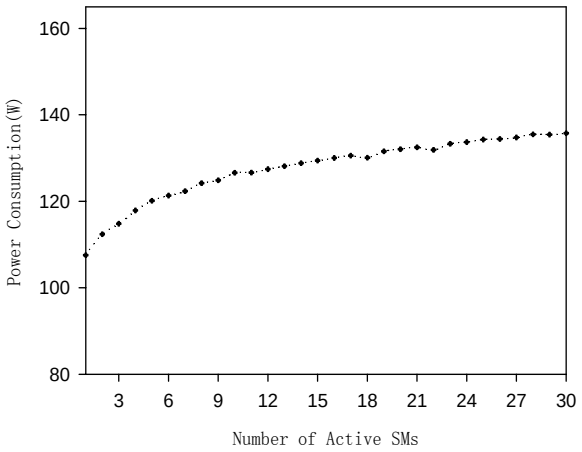

Figure 4 GTX 280 Power Consumption vs. Active SMs

As shown in Fig.4, the maxinum power delta between using only one SM and all SMs is about 30W. This can prove that the number of active SMs is an significant factor to the power consumption model. It is evident that the power consumption does not increase linearly as increasing the number of active SMs. The curve indicates a Non-linear relationship between the energy consumption and the number of active SMs. Consequently, the Non-linear regression function is built as follows.

$$P(n) = b_0 + \mu \log_{10}(\alpha n + \beta) \quad (4)$$

Where b_0 is the initial value and $b_0 = 90$, μ is scale factor that relies on the GPU architecture, n is the number of active SMs, the coefficients parameters α, β set respectively to 4.5 and 2.1. Then the same experiments were performed on the Low-End GPU NVIDIA GT200 with six SMs. The maxinum power delta between one SM and all SMs is only about 3.2W. Comparing Fig. 4 and 5 show that the number of active SMs has different impact on the power consumption to the different GPU architectures. The Nonlinear regression function for GT200 is as follows.

$$P(n) = 64.5 + 3.3\log_{10}(2n + 2.1) \quad (5)$$

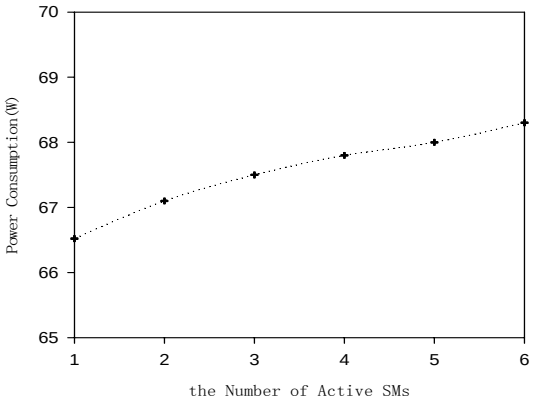

Figure 5 GT200 Power Consumption vs. Active SMs

C. Power Consumption Model of Slice

The approach used in this study aims to predict the power consumption of applications by the program slices. The program slice is a fine-grained power measurement level. Using the computation intensity based on energy consumption, we distinguished the power consumption between the computation instructions and the accessing instructions. Finally, the power consumption can be modeled by taking the computation intensity and the number of active SMs as shown in Equation 6.

$$P^i(x,n) = f^i(x) + \frac{\#SM_{active}}{\#SM_{bench}}[\mu^i \log_{10}(\alpha^i n + \beta^i)] \quad (6)$$

The final power consumption model is composed of two parts. The first part is obtained by the equation 3 and the input parameter is the slices computation intensity. The second part can adjust the power consumption accordingly to the number of active SMs for different applications and the input parameter is the number of active SMs. In the equation 6, $\#SM_{bench}$ is the number of SMs used in the experiments that determines the regression analysis function. $\#SM_{active}$ denotes the number of active SMs in the program slices.

D. Average Computation Intensity Model (ACIM)

The purpose of our study is to build power consumption model for programs. Suppose that the program $P = \{C_1, C_2, \ldots, C_n\}$, where $|C_i|$ is the slice length calculated by counting the statement number in the slice C_i. $\widetilde{A}^i$ represents the computation intensity of slice C_i. Then the power consumption of P is shown in Equation 7.

$$GPU\_P = \sum_{i=1}^{n} P^i(\widetilde{A}^i, n) \times t^i \quad (7)$$

From this equation, we can see that the program energy consumption is the sum of all the slices' power consumption. However this power consumption model is ideal scheme due to the fact that is difficult to apply into

the practice. The reason is that the executing time of each slices can not to be measured. To address this problem, the average computation intensity metric is introduced to evaluate energy consumption of programs. As shown in Equation 8, $\overline{A}$ denotes the Average Computation Intensity (ACI).

$$\overline{A} = \left(\frac{|C_1|}{\sum_1^n |C_i|}\tilde{A}^1 + \frac{|C_2|}{\sum_1^n |C_i|}\tilde{A}^2 + \ldots + \frac{|C_n|}{\sum_1^n |C_i|}\tilde{A}^n\right) \quad (8)$$

Suppose the executing time is T, the average computation intensity power consumption model (ACIM) is

$$GPU\_P = P(\overline{A}, n) \times T \quad (9)$$

TABLE I.

Identifier Maching

Target	Identifier
Kernel Function	__global__ , __device__
Shared Memory	__shared__
Constant Memory	__constant__
Texture Memory	texture,cudaArray
Global Memory	cudaMalloc 等
Arithmetic Operation	+,-,*,/,<<,>>,%
Memroy Accessing	=

In this work, we should explore an approach for calculating average computation intensity of program that need to solve three key problems as follows. (1) How to distinguish the different memory accessing patterns. Our apporach is to scan and match the identifiers which are used to direct the CUDA complier. These identifiers are listed in the table I [12]. For example, the variables located in the shared memory can be determined by the identifier __shared___. (2) How to distinguish the arithmetic and the memory accessing operations. In most cases the arithmetic operations and the memory accessing operations couple with closely. So the program statement may be transformed several logical statements accordingly to the operator symbols. Such as, (*Vect0) = m0 + m3. This statement may be decompsed two logical statements. The first one is a arithmetic logic statement, m0 + m3. And the other one is a memory accessing logic statement. (3) The last problem is to unroll the loop structure in a program in order to count the statements in loop body. Initially each slice is divided into many smaller blocks. Subsequently the loop body in each blocks are extracted by scanning the source code. After the unrolling process each statement in loop is added a count field recorded cycle index. The loop unrolling algorithm is as follows.

Algorithm3.1 Loop Unrolling in Slice

Input: Slice C = $\{B_1,B_2,...,B_k\}$

Output: Unrolled Slice C = $\{S_1,S_2,...,S_n\}$

1.For $B_i = B_1$ to B_k do
2. For each S_j in B_i do
3. S_j.count = 1;
4. Endfor
5.Endfor
6. For $B_i = B_1$ to B_k do
7. If B_i is LoopBlock then
8. Search the iteration_number in B_i;
9. For each S_j in B_i do
10. S_j.count = cycle index;
11. Endfor
12. Endfor
13. return C{ $S_1,S_2,...,S_n$}

In the loop unrooling algorithm, firstly each statement initializes the count field (line 1-5). Then matching the loop keywords is to extract the loop body (line7). When a loop is identified in the block, the cycle index is calculated by analyzing the source code (line 8). The next step is to set the count field of each statement by the cycle index (line 9-11). When the algorithm stops, the result is a set of the logic statements that have no loop body (line 13).

Algorithm3.2 Average Computation Intensity calculating

Input: Slice Set$\{C_{gm},C_{shm},C_{cm},C_{tm}\}$, Operator Sets OP^c,OP^m;

Output: average computation intensity $\overline{A}$;

1.For $C_i = C_{gm}$ to C_{tm} do
2. For each C_{ij} in C_i do
3. For each S_i in C_{ij} do
4. If S_i.op $\in OP^c$ then
5. #c = #c+ S_i.count;
6. Elseif S_i.op $\in OP^m$
7. #m = #m+ S_i.count;
8. Endfor
9. Endfor
10. $\overline{A}$ = #c/#m;
11. Return $\overline{A}$.

Assume that the program P consists of four different slice sets, P = $\{C_{gm},C_{shm},C_{cm},C_{tm}\}$, $C_{gm}^i \cap C_{gm}^j = \Phi$. The arithmetic operator set and the memory accessing operator set denote OP^c and OP^m respectively.

In the algorithm 3.2, the loop is to traverse each statements in each slice (line 1-3). If the statement has arithmetic operator, the arithmetic counter variable #c will be added (line 4-5). If the statement has memory accessing operator, the memory accessing counter variable #m will be increased (line 6-7). Finally, the average computation intensity is the ratio of the variable #c and the variable #m (line 10).

E. Probabilistic Slicing Model (PSM)

The average computation intensity model remains a significantly limitation that is unsuitable to predict the power consumption of the denseness-branch programs. In fact, there are plenty of denseness-branch programs due to the diversity and complexity of applications. The limitation of average computation intensity model will be discussed as follows.

```
1：int v1,v2,v3,v4
2:  v4 = input();
3:  v2 = 0;
4:  v3 = -2;
5：if (condition[1])
6：  then if (condition[2])
7:      then v1 =3;
8:      else v2 = 5 * v4/v1;
9:  else
10:     v1 = 10;
```

Figure 6. Denseness-branch Program

Figure 6 shows a denseness-branch kernel which will motivate the probabilistic slicing[13]. The energy consumption of this slice is the product between the execution time and the average computation intensity. The average computation intensity is calcuated the mean value of the statements in line 7, 8 and 10. Table 2 shows the branch outcome frequencies.

TABLE II.

Probabilities of Conditional Branch Outcomes

Line	Test	#executions	#then's	#else's
5	condition[1]	10000	1570	8430
6	condition[2]	1570	1560	10

From table II, the statement at line 8 has a executing probability of 0.001, so we may ignore the infrequent branch statement. However, this statement at line in ACIM make more contribution to the average computation intensity. This example illustrates that the average computation intensity model is not suitable for the denseness-branch programs.

The Probability Slicing Model (PSM) is proposed to handle the denseness-branch programs. The main idea of PSM is to calculate the computation intensity of program by incorporating probability information. It will be possible to remove infrequent statements as well as irrelevant statements in calculating the computation intensity process. Assume that the slice C_i in the program P includes two sub-slices C_{i1} and C_{i2}. If the executing probability of C_{i2} is less than specific threshold. The computation intensity of C_{i2} will be ignored and only use the C_{i1}'s computation intensity as the whole program's computation intensity.

Then we discuss how to obtain the probability of the branches. The main idea of the approach is used the historical data to predict the future data. Firstly, select some data to calculate the braches probability of the specific programs. Assume that the probability results obtained by the test data will be available to future data that are in the executing phase of the specific programs.

Given P contains program slice sets, $P = (C_1,C_2,...,C_n)$. The probability vector $p_i <p_{i1},p_{i2},...,p_{im}>$ is the executing probability for each sub-slices C_i $(C_{i1},C_{i2},...,C_{im})$. The computation intensity of all the sub-slices for C_i represents $<a_{i1},a_{i2},...,a_{im}>$. Then the computation intensity calculating algorithm is as follows.

Algorithm3.3 Computation Intensity calculating of PSM

Input: $<p_1,p_2,...,p_i,...,p_n>$, $<a_i,a_i,...,a_i> \delta$;

Output: the computation intensity of Program A;

1.For $C_i = C_1$ to C_n do

2. For $C_{ij} = C_{i1}$ to C_{im} do

3. If $P_{ij} > \delta$ then

4. $A \mathrel{+}= p_{ij} \Big/ \sum_{j=1}^{m} p_{ij} \times a_{ij}$

5. End for

6.End for

Finally, PSM is as shown in Equation 10.

$$GPU\_P = P(A, n) \times T \quad (10)$$

Where T is the executing time of program, A is the Computation Intensity of the Probability Slicing (PSCI) obtained by the algorithm 3.3. And n is the number of active SMs.

IV. EXPERIMENTS

The goal of the experiments is to verify the accuracy of the power consumption model by comparing the prediction value and the measured one. Our testbed is an Intel Core 2 processors, 2G DDR RAM, 320G SeaGate HardDisk and NIVIDA GeForce GTX280 card with 602 MHZ core frequency and 1107 MHZ memory frequency. The operation system is Windows XP Profession with CUDA toolkit2.3 and the driver version is CUDA 190.29.

A.Average Computation Intensity Model

In this experiment, we selected four GPGPU applications to verify the power consumption model for the sparseness-branch programs[14]. As shown in Table III, these applications are the sparseness-branch programs. The computation intensity of these applications range from 0.18 to 0.93 on a 0 to 1 scale. The experimental results show that the relative error between the prediction value and the meassured value are not more than 6%. These data enable us to conclude that ACIM is available for the sparseness-branch programs.

TABLE III.

Experimental data of average computation intensity model

Bench mark	Description	ACI	ACIM	Messured
Dotp	Matrix dotproduct	0.574	114.6	119.5
Madd	Matrix multiply-add	0.921	101.2	105.6
Dmadd	Matrix double multiply add	0.927	102.1	108.2
Mtrans	Matrix Transpose	0.182	86.8	92.3

B. Probability Slicing Model

To verfiy the probability slicing model, these applications Discrete Cosine Transform (DCT), String greping and H.264 decoding were selected in this experiments. Compare to the DCT algorithm, the string greping algorithm and H.264 decoding algorithm have more branch statements[15]. From the table □, the computation intensity of DCT based on probability is less than the average computation intensity. So the estimated value of probability model is less than the ACI value and the relative error is 2.4% that is also less than the average

computation model 3.8%. The relative error delta in the string greping algorithm and H.264 decoding algorithm are more greater than the DCT algorithm due to more branch statements. We can conclud that PSM is prior to ACIM to the denseness-branch programs.

TABLE IV.

Experimental Data of PSM

Benchmark	Description	ACI	PSCI	ACIM	PSM	Messured
DCT	Discrete Cosine Transform	0.724	0.676	125.4	123.5	120.6
cugrep	String greping	0.532	0.725	110.5	117.1	114.2
cuh264	H.264 decoding	0.623	0.9478	108.14	119.7	122.3

V. CONCLUSION

We presented program slicing method for power profiling and evaluation of General-Computing GPU applications at the program slice granularity. With the aid of static program slicing technique, we quantified the power consumption of applications on the current and emerging GPU high-performance computing field. Two important factors that are the number of branch statements and active SMs are considered. The GPGPU programmers can use this power consumption model to analyze theirs application's energy consumption profile. So it also be used to optimize the power consumption in the GPGPU field. In our future work, we will improve the power consumption model to adapt to various GPU architectures and further enhance the prototype functionalities. Such as extracting the code area that comsume more energy than other areas will be designed.

ACKNOWLEDGMENT

We would like to thank the support of National Nature Science Foundation of China (No.60970012), the Innovation Program of Shanghai Science and Technology Commission (No.09511501000, 09220502800), and Shanghai leading academic discipline project (S30501).

REFERENCES

[1] Patrick Kurp, "Green Computing, Commons". Of the Association for Computing Machinery, 51(10):11-13,2008.

[2] Y.Jiao, H. Lin, P. Balarji.et al. "Power and Performance Characterization of Computational Kernel on the GPU". IEEE/ACM Int'l Conference on Green Computing and Communications& Int'l Conference on Cyber, Physical and Social Computing. pp:221-228,2010

[3] M.Z.Shaikh, M.Gregoire, W.Li, M. Wroblewski, S.Simon, "In situ Power Analysis of General Purpose Graphical Processing Unit", In 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing. 2011

[4] Sunpyo Hong, Hyesoon Kim. "An Integrated GPU Power and Performance Model", in ISCA'10. 2010..

[5] Jeremy Enos, Craig Steffen, Joshi Fullop, Michael Showerman, et al. "Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters".

[6] R. Ge, X. Feng, S. Song, H. Chang, D.Li, K.Cameron, "PowerPack: energy profiling and analysis of high-performance systems and applications". IEEE Transactions on Parallel and Distributed Systems, Vol. 21, No.5, pp. 658-671,2010.

[7] S. Collange, D.Defour, and A. Tisserand, "Power consumption of gpus from a software perspective", in Proceeding of the 9th International Conference on Computational Science. Berlin, Heidelberg: Springer-Verlag，pp.914-923,2009.

[8] Tan, T. K. Raghunathan, A. Lakshminarayana,G. et al. "High-level software energy macro-modeling" In Proceeding of Design Automation Conference,2001, pp:605-610.

[9] Guibin Wang, YiSong Lin, Wei Yi. "Kernel Fusion: an Effective Method for Better Power Efficiency on Multithreaded GPU". IEEE/ACM Int'l Conference on Green Computing and Communications& Int'l Conference on Cyber, Physical and Social Computing. pp:344-349,2010.

[10] M. Weiser. "Program slicing" IEEE Transactions on Software Engineering,10(4):352-357, 1984.

[11] Pharr M, Fernando R. GPU Gems2. Boston: Addison Wesley, 2005:493-495.

[12] NVIDIA_Corporation, CUDA_3.0 Programming Guide, 2010, http://www.nvidia.com/ (accessed May 2010)

[13] Jeremy Singer. "Towards Probabilistic Program Slicing. In Beyond Program Slicing, Dagstuhl Seminar Proceedings". July,2006

[14] S. Hong , H. Kim. "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness". In ISCA,2009

[15] Moecke,M. Seara, R. "Sorting Rates in Video Encoding Process for Complexity Reduction" IEEE Transactions on Circuits and Systems for Video Technology. 20(1):88-101,2010.

Haifeng Wang. He received his diploma in computer science from the Shandong University in 1995, China. He is currently a PHD candidate in the University of Shanghai for Science and Technology. His current research interests include GPU Computing, Network Computing(gadfly7@126.com).

Qingkui Chen. Received the MS degree in computer science from the JILIN university, and PhD degree in University of Shanghai for Science and Technology. He is a full professor of computer science at the University of Shanghai for Science and Technology, China, where he is the head of the Network Computing Group and the Wireless Sensor Network Research Center. His research interests include GPGPU, Network Computing, and WSN. (chenqingkui@gmail.com)