


A statistic approach for power analysis of integrated GPU

Qiong Wang¹  · Ning Li¹ · Li Shen¹ · Zhiying Wang¹

Published online: 17 August 2017
© Springer-Verlag GmbH Germany 2017

Abstract As datasets grow, high performance computing has gradually become an important tool for artificial intelligence, particularly due to the powerful and efficient parallel computing provided by GPUs. However, it has been a general concern that the rising performance of GPUs usually consumes high power. In this work, we investigate the study of evaluating the power consumption of AMD's integrated GPU (iGPU). Particularly, by adopting the linear regression method on the collecting data of performance counters, we model the power of iGPU using real hardware measurements. Unfortunately, the profiling tool CodeXL cannot be straightforwardly used for sampling power data and as a countermeasure we propose a mechanism called kernel extension to enable the system data sampling for model evaluation. Experimental results indicate that the median absolute error of our model is less than 3%. Furthermore, we simplify our statistical model for lower latency without significantly reducing the accuracy and stability.

Keywords Integrated GPU · Power analysis · Statistical model · Kernel extension

Communicated by V. Loia.

✉ Qiong Wang
wangqiong@nudt.edu.cn

Ning Li
ln11@nudt.edu.cn

Li Shen
lishen@nudt.edu.cn

Zhiying Wang
zywang@nudt.edu.cn

¹ College of Computer, National University of Defense Technology, Changsha, China

1 Introduction

Due to the capability of powerful and efficient parallel computing, GPUs have been widely used in various computing systems, especially in high performance computing systems. Particularly, due to the parallel programming languages, e.g., CUDA (Corparation 2016c) and OpenCL (Stone et al. 2010), GPUs could significantly accelerate the solving of large-scale computation problems (Chitty 2016), such as video analytics, speech recognition and image classification (Li et al. 2016). It is reported that GPUs could be applied in running machine learning models for classification tasks in the cloud, and thus is able to handle far more data volume and throughput while consuming less power (Corparation 2017; Zhang and Xiao 2016).

However, the rising performance of GPUs is usually at the cost of increasingly high power consumption, especially when GPUs are integrated with massive transistors (e.g., NVidia GeForce GTX 280 contains 1.4 billion transistors). For example, an Nvidia GTX 280 could consume as high as 236 W power (Corparation 2016b), while it is usually less than 150 W (Corparation 2016a) for a typical multi-core CPU. As a result, more complex cooling solutions are needed to reduce the system temperature. This could compensate the benefits of improving the system performance. Therefore, reducing the GPU power consumption is highly necessary, among which analyzing and modeling the power consumption of GPUs is clearly a priority.

In this work, we aim at the accelerated processing unit (APU) (Branover et al. 2012), which is a revolutionary architecture released by AMD in 2011. In a APU, the CPU and integrated GPU (iGPU) are combined into one single chip so that they could share resources on the same chip. Therefore, to improve the APU efficiency, it is of significant importance

to figure out the kernel bottlenecks and determine the power consumption of iGPU.

Unfortunately, despite that there have been several power models built for GPU, they mainly focused on either the whole APU or the discrete GPU (Zhang and Owens 2011; Luo and Suda 2011; Bagsorkhi et al. 2010; Hong and Kim 2009; Wu et al. 2015; Diop et al. 2014; Zhang et al. 2011; Karami et al. 2015). The iGPU in the APU has received relatively little attentions. Since it is usually the case that iGPUs and discrete GPUs differ from each other, it is necessary for researchers to investigate the study of power consumption for iGPUs.

Our contributions To precisely analyse the power consumption of iGPU, we use a statistical model for power analyse. Our goal is to conduct a comprehensive investigation on the iGPU performance and its power consumption, providing an accurate predication for reducing the power consumption of iGPUs. Overall, our main contributions could be summarized as follows:

- We found that the widely adopted profiling tool, i.e., AMD CodeXL (AMD 2016) cannot be directly used to collect the energy consumption of OpenCL kernels. Precisely, the power sampling period of CodeXL is at least 100 ms while the execution time of most OpenCL kernels is less than 10 ms. By lengthening kernel extension time, we introduce a novel solution to sample data in one kernel execution. We insist that this mechanism is generally applicable for collecting data from OpenCL-based benchmarks.
- Relying on the multivariable linear regression model, we successfully build a statistical power model for the iGPU. We then adopt the proposed kernel extension for data collecting to evaluate our built model. The results show that our model is of higher accuracy than most existing models. Moreover, unlike previous models that collect data from GPU simulators, our experiment results are for real iGPUs and hence more meaningful in practice.
- Low latency is a desirable property for a power model especially when it is used to estimate power consumption. For this issue, we explore the possibility of simplifying our built model. Specifically, a sensitivity analysis is done to evaluate the role of each performance counter in the model, based on which we simplify our statistic model via removing some performance counters of less importance.

Related work Compared to the CPU, the internal interface of the GPU architecture is far less open. Many brands of GPUs only own a vendor internal debugging but cannot provide performance counter information. Therefore, the power consumption analysis for GPU was mainly carried out on simulators at the early stage. In 2012, Tor Aamodt Research Group designed a general-purpose GPU simulator called

GPGPU-Sim (Wang et al. 2012), which could be adopted to implement GPU operating states of clock level. The simulator integrates the GPUWattch (Leng et al. 2013) power module and each hardware event is assumed to consume the same energy. The total power consumption of the GPU is then calculated by collecting the performance counter information of all components. Diop et al. (2014) simulated the total power dissipation of the APU and used the power measurement unit connected externally to measure the processor power consumption. The Multi2Sim simulator is used to simulate AMD's EverGreen APUs. Similar to our work, it also, it also uses the regression model to estimate the processor power consumption while the model error is more than 20%. Although it was the first time to achieve power model in the AMD platform, the model accuracy is not desirable. Subsequently, some approaches were proposed to study the GPU power modeling on the real machine for more accurate analysis. Zhang et al. (2011) aimed at AMD's HD5870 and also adopted the regression method to study the models of both power consumption and performance prediction. Similarly, Karamiet et al. (2013) focused on the NVIDIA's GPU modeling. They used the GPGPU-Sim to predict and evaluate the performance and power consumption of OpenCL kernels.

Organization The rest of this paper is organized as follows. In Sect. 2, we briefly introduce the APU architecture, OpenCL execution model, AMD CodeXL and Rodinia. We then describe our proposed statistic model in Sect. 3. Our experimental results and analysis are then presented in Sect. 4. In Sect. 5, we show how to simplify the model for lower latency. Finally, we draw the conclusion in Sect. 6.

2 Preliminaries

2.1 APU architecture

In this work, AMD A10-7850k is chosen as the target APU, which consists of four Steamroller CPU cores, 8 computing units of Graphics Core Next (GCN) and 512 shade processors. The GCN architecture is depicted in Fig. 1. Precisely, in each GCN computing unit, one could find a CU scheduler, a branch & message unit, 4 texture filter units, 4 SIMD vector units, 1 scalar unit, 4 64KB VGPR files, 16 texture fetch load/store units, a 64 KB local data share, a 4 KB GPR file and a 16 KB L1 Cache.

2.2 OpenCL execution model

OpenCL (Stone et al. 2010) is an open programming framework which generally consists of a programming language for running kernels on heterogeneous platforms, such as CPU, GPU, FPGA and other accelerators. OpenCL can provide two types of parallel computing capabilities, i.e.,

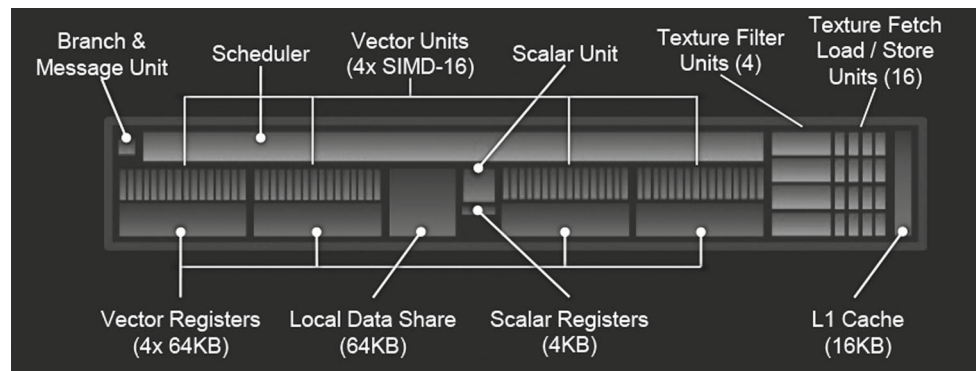


Fig. 1 AMD GCN architecture

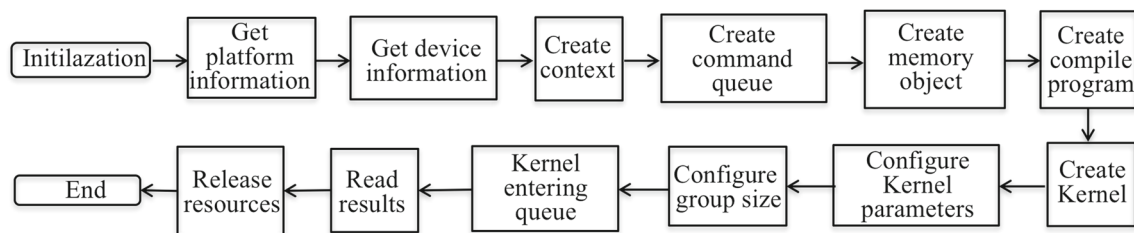


Fig. 2 Kernel execution on OpenCL

task-based and data-based, for different applications. An OpenCL program may belong to either the host program or kernels. We usually call a running kernel as a work item, several of which on a computing unit are set as a work group. A local ID is assigned for each work item and each work group has a global ID, due to which we can exclusively figure out each work item in the global space. As shown in Fig. 2, for a running kernel, the host program would create a context to define the operating environment of kernels, which includes a device list, memory objects, kernels and program objects.

2.3 CodeXL

In our work, we use the CodeXL (AMD 2016) for profiling power and performance counter information of the target iGPU during the benchmark execution. CodeXL is an open software development tool suite for AMD processor. Particularly, it can help discover the bottleneck of the program running on CPU, GPU and APU.

A CodeXL consists of several components, i.e., GPU debugger, GPU profiler, CPU profiler, static shader/kernel analyzer, power profiler and graphic frame analyzer. Here we mainly introduce the GPU profiler and the power profiler. The GPU profiler offers hotspot analysis for AMD GPUs and APUs via collecting and visualizing hardware performance counters data, kernel occupancy and application trace. Particularly, the profiler can be used to discover performance bottlenecks for kernel execution optimization by gathering data from the OpenCL runtime, and from the GPU/APU

itself during the kernel execution. The power profiler could gather the real-time power consumption of both the single GPU component and the whole APU. Furthermore, it can also profile the CPU and GPU frequency and provide two modes, i.e., command mode and visible interface mode, for power analysis.

2.4 Rodinia benchmark

Rodinia is well known as the first benchmark suite targeted on heterogeneous computing. It was built by Che et al. (2008) for studying emerging platforms. Rodinia consists of applications and kernels which are mainly for multi-core CPU and GPU platforms. Rodinia benchmarks consist of three versions, OpenMP, CUDA and OpenCL, covering various parallel communication patterns, power consumption and synchronization techniques.

3 Modeling methodology

3.1 Experimental setup

In this work, A10-7850K is chosen as the experimental platform. Details are given in Table 1. Particularly, we rely on an Ubuntu 12.04 LTS machine to perform all the experiments. We choose the AMD APP SDK v2.9 for the OpenCL implementation. We run the Rodinia 3.2 Benchmark Suite to evaluate our proposed model.

Table 1 System configuration

Operating system	Ubuntu 12.04 LTS
Memory	8GB DDR3
OpenCL	AMD APP SDK v2.9
Profiling tool	AMD CodeXL v1.9
Benchmark Suite	Rodinia v3.2
APU version	AMD A10-7850K
APU core number	4CPU + 8 GPU
APU L1 Cache	192KB instruction, 64KB data
APU L2 Cache	4MB
APU TDP	95W

3.2 Regression factors determining

In this work we rely on the linear regression to build the power prediction model for iGPU. Noting that the selection of regression factors (i.e., explanatory variables in the regression model) would significantly affect the accuracy of the built prediction model, we carefully analyze the architecture of iGPU to figure out those on-chip resources that are closely related to its power consumption.

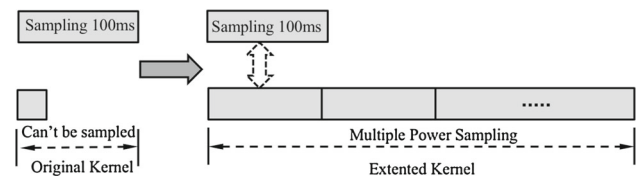
Particularly, these on-chip resources mainly consist of *Arithmetic Logic Unit (ALU)*, *Local Memory* and *Global Memory*. The main reason is that arithmetic operations in iGPU incur frequent data access with the memory and hence usually consume much power. That is, hardware events occurring in these operations account for the majority of power consumption. In our work, we utilize *performance counters* to record these events. Performance counters are used to calculate the hardware events number and for the application profiling. To profile the performance counters, we choose the profiler of AMD CodeXL v1.9. Table 2 describes the details of the performance counters collected by the profiler.

Note that the power consumption of an iGPU consists of two parts. The first part (also the main one) is the dynamic power consumption while the other part is the static one. More precisely, the dynamic part occurs due to the computation task, whose volume is accurately reflected by the number of hardware events captured by the performance counters listed in Table 2. That is, the number of activated hardware events would be significant when many computation tasks occur, which of course results in high power consumption. Regarding the static one, it is also known as basic power consumption as it is mainly caused by the bare operating system without any workload running. It is normally related to the processor architecture and also the outside environment such as the temperature.

Based on the aforementioned analysis, we formally set the model equation as follows:

Table 2 Performance counters description

Counters	Descriptions
VALUInsts	Vector ALU instructions number
SALUInsts	Scalar ALU instructions number
VFetchInsts	Vector fetch instructions number
SFetchInsts	Scalar fetch instructions number
FlatVMemInsts	Flat instructions number
VALUUtilization	Percentage of active vector ALU threads in a wave
VALUBusy	Percentage of time vector ALU instruction is processed
SALUBusy	Percentage of time scalar ALU instruction is processed
LDSInsts	LDS read or LDS write instructions number
LDSBankConflict	Percentage of time LDS is stalled by bank conflicts
CacheHit	L2 cache hit rate
MemUnitBusy	Percentage of time memory unit is active
MemUnitStalled	Percentage of time memory unit is stalled
WriteUnitStalled	Percentage of time write unit is stalled
FetchSize	Total kilobytes fetched from the memory
WriteSize	Total kilobytes written to the memory

**Fig. 3** Kernel extension mechanism

$$P_{\text{iGPU}} = \sum_{i=1}^{i=15} A_i \cdot E_i + C \quad (1)$$

In Eq. (1), P_{iGPU} denotes the total power consumption of the targeted iGPU. E_i represents the i -th performance counter listed in Table 2 and A_i is its corresponding coefficient. C is a constant value that denotes the static power consumption.

3.3 Kernel extension

As mentioned above, we choose the AMD CodeXL v1.9 to sample the performance counter information and also the power during the program execution. Unfortunately, we found that for CodeXL, the sampling period could be over 100 ms but for most kernels in Rodinia 3.2, the execution time is less than 10 ms. That means that the power information cannot be profiled within one kernel execution. Intuitively, we illustrate the conflict between the power sampling period and the kernel execution time in Fig. 3.

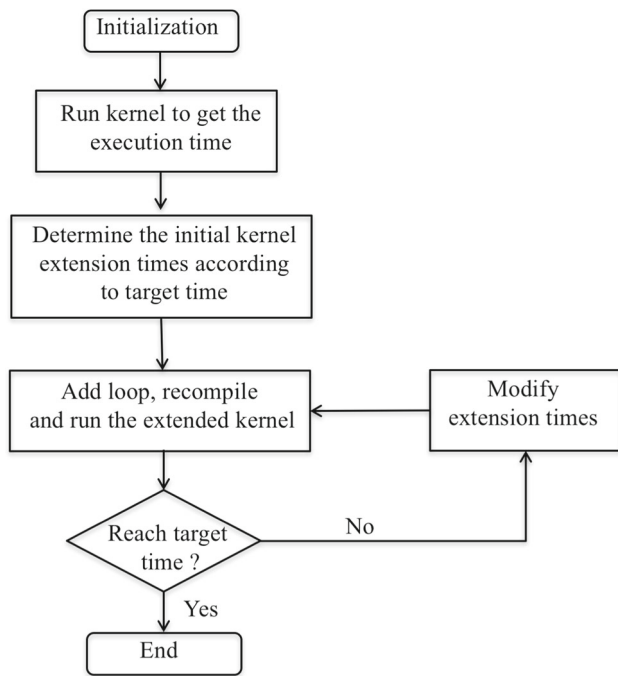


Fig. 4 Procedure of kernel extension

In order to overcome the above problem, we propose an approach called *kernel extension* to lengthen their execution time so that the power could be sampled within one kernel execution. The kernel extension procedure is depicted in Fig. 4. Particularly, kernels are rewritten and the original functions repeat for thousands of times. In this way, each kernel produces lots of hardware events within one kernel execution time so that the CodeXL could profile enough data information for the power model. Precisely, for high model accuracy, we modify the benchmark to make the kernel execution time longer than 1 s so that the CodeXL could collect at least 10 samples within one kernel execution.

To give a clearer picture, we take an application called BackPropagation as an example. The details of the kernel extension is as below.

1. Pick a kernel called `bpnn_layerforward_ocl_k1_Spectre1`, which is used to calculate the power array. Its original running time is only 0.4 ms.
2. The target running time is 1.7 s. Therefore, it needs to be extended for 4000 times.
3. The main function of the kernel is the Addition, Subtraction, Multiplication, Division. Therefore, we make this loop for 4000 times, and then re-compile and execute the kernel.
4. The real running time of the kernel increases to 0.8 s. However, it is still short compared to 1.7 s, and hence we increase the loop times.

Table 3 Extended kernel execution in Rodinia

App.	Kernel	iT (ms)	eT (ms)
Backprop	<i>bpnn_layerforward_ocl_k1</i>	2.01	1699.45
	<i>bpnn_adjust_weights_ocl_k2</i>	3.10	1988.48
BFS	<i>BFS_1_k1</i>	0.10	2188.54
	<i>BFS_2_k2</i>	0.09	1674.26
B+tree	<i>findRangeK_k1</i>	2.15	1819.42
	<i>findK_k2</i>	8.19	2114.41
Gaussian	<i>Fan1_k1</i>	0.01	2049.41
	<i>Fan2_k2</i>	0.20	1429.43
Heartwall	<i>kernel_gpu_opencl_k1</i>	131.71	1530.07
Hotspot	<i>hotspot_k1</i>	0.38	3526.27
Kmeans	<i>kmeans_swap_k1</i>	88.92	2213.84
	<i>kmeans_kernel_c_k2</i>	31.12	2182.16
Leukocyte	<i>GICOV_kernel_k1</i>	14.29	1440.90
	<i>dilate_kernel_k2</i>	7.55	1641.12
	<i>IMGVF_kernel_k3</i>	42.17	2432.67
LavaMD	<i>kernel_gpu_opencl_k1</i>	155.93	1556.52
NN	<i>NearestNeighbor_k1</i>	0.10	1553.09
PF-naive	<i>particle_kernel_k1</i>	5.53	1007.59

5. When the loop is repeated for 9000 times, the running time of the kernel becomes 1699 ms which almost equals to the target running time. We hence finish the extension of `bpnn_layerforward_ocl_k1_Spectre1`.

The extended kernel execution time for each application in Rodinia is depicted in Table 3. It is worth mentioning that for some applications, simply adding the loop for the computation task may meet some problems. This is due to the complication of the OpenCL program framework. Particularly, adding too many loops could incur memory error. Therefore, some special treatments are required for the kernel extension.

3.4 Power profiling

We run the Rodinia 3.2 Benchmark Suite and use CodeXL to collect the profiling data, as depicted in Fig. 5. It is worth noting that the power sampled by the CodeXL is the global consumption and not necessarily always related to the kernel execution. For example, each time before a program starts to execute, there always exists a setup phase which usually does not involve the kernel. If we straightforwardly use the sample power from the CodeXL for model evaluation, it may occur errors and thus reduce the model accuracy. Therefore, we need to pick the kernel-related power from the recorded data to achieve high model accuracy.

To provide a clear picture, we choose the B+tree to illustrate this point. As depicted by Table 3, there are two kernels

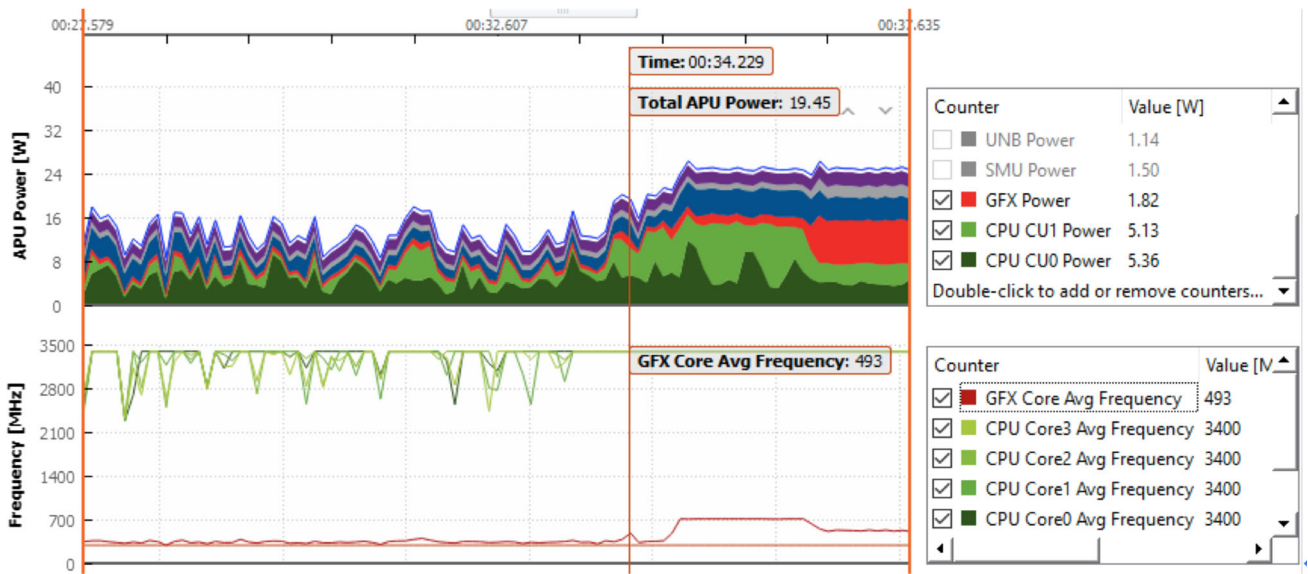


Fig. 5 CodeXL for APU power profiling

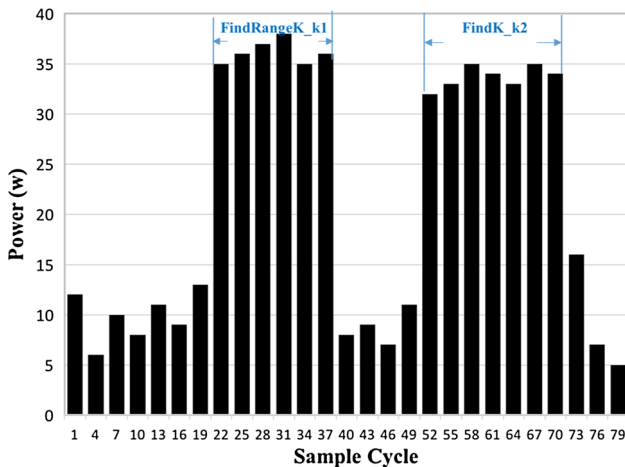


Fig. 6 Power for kernels in B+Tree

for B+tree: *findRangeK_k1* and *findK_k2*. We show the details in Fig. 6. One can note that the first 19 sample cycles is mainly for execution environment setup, and hence the power consumption is quite small, e.g., 5–15 W. After that, the kernel of *findRangeK_k1* begins to execute and meanwhile the power rapidly increases to around 35 W correspondingly. When it comes to the 38 cycle, the power recorded decreases sharply, which means the end of *findRangeK_k1* execution. Therefore, the power sampled from 19 cycle to 38 cycle is particularly the total power consumption for *findRangeK_k1*. As shown in Table 3, the running time of *findRangeK_k1* is about 1800 ms, and hence is consistent with the sample period. One can also note that the follow-up power recorded appears similarly and is mainly for the *findK_k2* execution.

Table 4 Statistical result

Multiple	0.997615
R square	0.995235
Adjusted R square	0.959498
Standard error	4.898391

4 Results and analysis

4.1 Model evaluation

We choose the Statistical Product and Service Solutions (SPSS) to evaluate our regression model.

The statistical result of our model is shown in Table 4 and the regression coefficient of each performance counter is listed in Table 5. Precisely, the model achieves an adjusted R-square of 95.9%. This indicates that the linear relationship between variables (both the independent and dependent ones) is reliable. Moreover, one the median absolute error is 2.12%, showing a good predicting precision.

We also test the predication error of each kernel in the Rodinia benchmark. The result is given in Table 6. Overall speaking, our model is of high accuracy as each kernel error is small. e.g., only 0.25% for *kernel_gpu_opencl_k1*.

4.2 Error analysis

One may notice that some kernels are of higher predication error compared to others. For example, the error of *findRangeK_k1* is 4.88% while that of *particle_kernel_k1* is 6.02%. The main reasons resulting in this are as follows.

Table 5 Regression coefficients of the model

PC	Coefficient	PC	Coefficient
VALUInsts	-1.5E-05	WriteSize	1.54E-05
SALUInsts	3.03E-05	CacheHit	-0.76
VFetchInsts	0.000187	MemUnitBusy	0.75
SFetchInsts	-1.9E-06	MemUnitStalled	-1.30
VALUUtilization	-1.03153	WriteUnitStalled	226.03
VALUBusy	1.500774	LDSInsts	0.00022
SALUBusy	-1.43736	LDSBankConflict	37.90
FetchSize	2.55E-06		

Table 6 Predication errors for all kernels

Kernel	Error (%)	Kernel	Error (%)
<i>bpnn_layerforward_ocl_k1</i>	2.63	<i>hotspot_k1</i>	0.48
<i>bpnn_adjust_weights_ocl_k2</i>	0.64	<i>kmeans_swap_k1</i>	0.31
<i>BFS_1_k1</i>	2.36	<i>kmeans_kernel_c_k2</i>	0.62
<i>BFS_2_k2</i>	3.60	<i>GICOV_kernel_k1</i>	0.80
<i>findRangeK_k1</i>	4.88	<i>dilate_kernel_k2</i>	3.91
<i>findK_k2</i>	1.06	<i>IMGVF_kernel_k3</i>	4.27
<i>Fan1_k1</i>	1.52	<i>kernel_gpu_openc1_k1</i>	0.25
<i>Fan2_k2</i>	3.86	<i>NearestNeighbor_k1</i>	2.21
<i>kernel_gpu_openc1_k1</i>	0.36	<i>particle_kernel_k1</i>	6.02

- *Kernel extension* On the one hand, we found that the length of kernel execution time affects the prediction error significantly. Particularly, our built model produces high error for those kernels with short execution time. This is because that the shorter time the kernel executes for, the less power data the CodeXL could record, and hence the higher sample error occurs. For example, the execution time of *particle_kernel_k1* is the shortest (i.e., 1007.59 ms) among all the kernels while its corresponding predication error is the highest (i.e., 6.02%). On the other hand, as shown in Table 3, for each kernel, the extended execution time is usually not integral multiple of 100 ms while the CodeXL samples the power per 100 ms strictly. This would result in mismatching between the power sample and the kernel execution.
- *Implementation environment* Beside the aforementioned data measurement issue, the environment also plays an important role in affecting the model accuracy. Essentially, it would affect the static power consumption. The following equations represent the relationship between the static power consumption and the temperature and the voltage.

$$P_{\text{idle}} = W_1 \times T + W_0, \quad (2)$$

where $W_1 = a_3 \times V^3 + a_2 \times V^2 + a_1 \times V + a_0$, $W_0 = b_3 \times V^3 + b_2 \times V^2 + b_1 \times V + b_0$. Here P_{idle} denotes the static power and T denotes the temperature, V represents the voltage and a_i, b_i ($i \in [0, 3]$) are constant multivariable coefficients. One can note from the above equation that the static power of preprocessor is linear with the temperature when other variables remain unchanged. In our model equation, we use the constant value C to denote the static power. Ideally, this item should be stable during model evaluation for achieving high model accuracy. However, we are normally unable to guarantee that the temperature remains stable during the experiment. The heat caused by the executing program would always raise the environment temperature. Moreover, the temperature raise would also reduce the voltage, which will reduce both the W_0 and W_1 .

5 Simplifying model for lower latency

The power of the processor is tightly related to the frequency and voltage of the underlying processor. Normally, the power consumption will be smaller when the processor frequency and voltage are lower. Relying on this observation, dynamic voltage frequency scaling (DVFS) was proposed to reduce the processor frequency and voltage to lower the power consumption of the processor. The processor power in both the current and other states are considered for dynamic scaling. While applying DVFS to APU, the processor is quite sensitive to the instantaneity, which demands the power model must have low latency. Furthermore, due to the combination of CPU and GPU, the space for other online resources is quite limited on APU. If the power model is complicated and includes many performance counters, it will be difficult for the design and usage of APU hardware. Therefore, our goal here is to reduce the latency of the power model. Particularly, we ask whether it is possible to cut down the performance counters number involved in the power model while the model accuracy remain high.

In this work, we first evaluate the importance of each performance counter by exploring the relationship between the type of performance counters and the standard error of power models. We then further simplify the power model to tackle with the aforementioned dilemmas. We mainly explore the connection among the number of performance counters and the stability and prediction accuracy of power model.

5.1 Performance counter evaluation

We first evaluate the sensitivity of the model and analyze the importance of each performance counter so that we can figure out the possibilities to simplify our built power model.

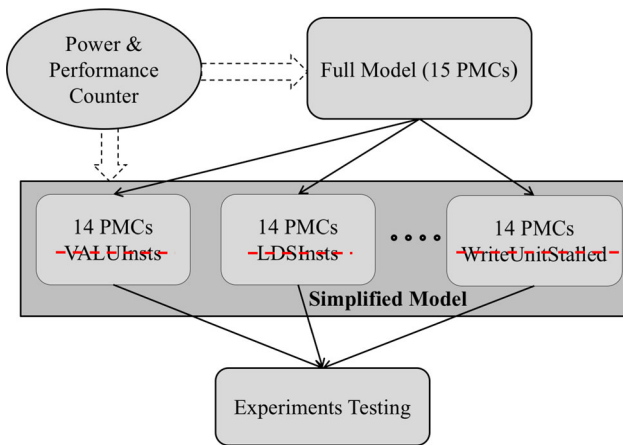


Fig. 7 Performance counter evaluation procedure

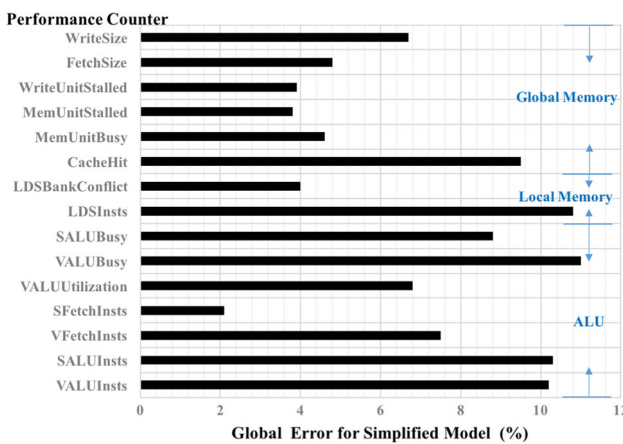


Fig. 8 Global error for performance counter

As depicted by Fig. 7, for each testing, one performance counter is removed from the full model which is of 15 performance counters. We then run the benchmark and sample the power and performance counter to reconstruct the regression model. Formally, for j -th ($j \in [1, 15]$) simplification, the model equation is as follows:

$$P_{iGPU} = \sum_{i=1}^{j-1} A_i \cdot E_i + \sum_{i=j+1}^{15} A_i \cdot E_i + C \quad (3)$$

The goal of each simplified model is to evaluate the importance of the removed performance counter, based on which we can simplify the model by removing those performance counters that are of less importance and hence the simplified one can still achieve high accuracy.

The accuracies, shown in Figs. 8 and 9, are compared to the full model which has 15 performance counters. The accuracy variation directly reflects the importance of performance counters. If the removed performance counter results in a sig-

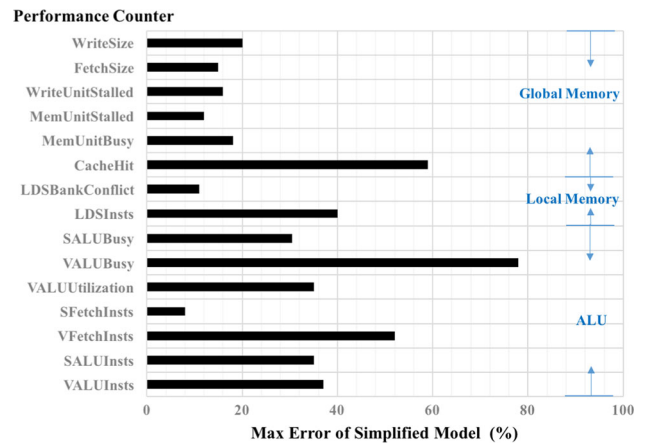


Fig. 9 Maximum error for performance counter

nificant loss in the model accuracy, we say that it plays an important role in the power consumption. Precisely, we have the following conclusions.

- Most ALU performance counters play important roles in the power consumption. In general, when an ALU performance counter is removed from the model, the resulting errors and accuracy variations would be higher than that of the full model. Particularly, the VALUBusy is the most crucial one to the proposed model among all ALU performance counters. It is worth mentioning that the SFetchInsts is an exception, without which, the median absolute error is 2.21%.
- The local memory performance counters are of various importance. The LDSInsts is quite a decisive component, as it reflects the state of executing LDS instructions on iGPU. Therefore, when the iGPU is processing more LDS instructions, the resulting power consumption would increase. The LDSBankConflict is very small, this is because that the related hardware event rarely happens and thus consumes little power.
- Simplified models without WriteSize and MemUnitBusy are still of high accuracy due to the small dataset in the benchmark. One could notice similar cases on MemUnitStalled, WriteUnitStalled. Essentially, in our chosen benchmark, the cache hit rate is high and thus the global memory access is small, leading to a low power consumption (Fig. 10).

5.2 Model simplifying

Based on the above performance counter importance analysis, we have the fact that some performance counters do not consume much power and thus affect the model accuracy slightly. Particularly, when the SFetchInsts is removed

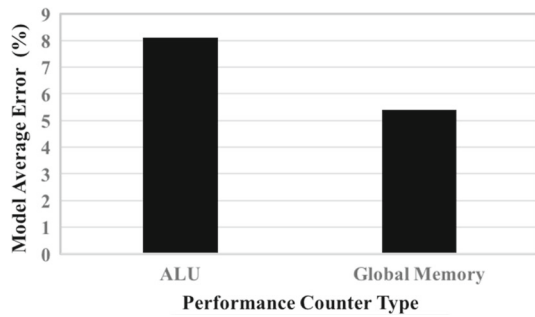


Fig. 10 Average error of performance counter

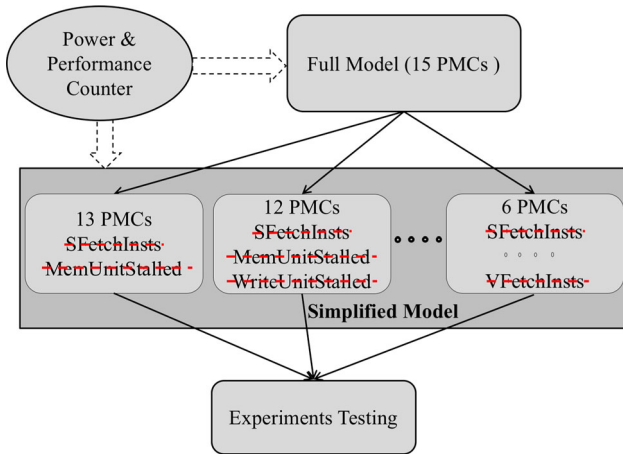


Fig. 11 Multi-simplification model building

from the model, the accuracy is almost the same as that of the full model. This shows the possibility for us to simplify the model by removing SFetchInsts. To simplify the model, as shown in Fig. 11, we start with the full model and gradually remove the performance counters that are of less importance. The simplification procedure stops when only 6 performance counters remain in the list, resulting in 9 simplified models for further analysis. These simplified models are equipped with 14, 12, . . . , 6 performance counters, respectively.

Since now we remove more than one performance counter from the power model, its stability would be possibly weakened significantly. Normally, a power model with weak stability is not suitable for DVFS even if it is still of high accuracy.

Therefore, in addition to the median absolute error, we choose the root mean square deviation (RMSE) to evaluate the accuracy and stability of the simplified models. The RMSE can well indicate the sample standard deviation between the predicted and observed values.

As depicted by Figs. 12 and 13, when the number of performance counters are 11 or fewer, the median absolute errors of model would be above 10%, which means that the simplified model would provide inaccurate predictions. Generally

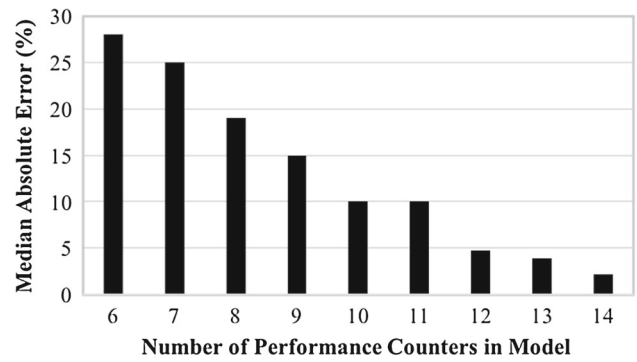


Fig. 12 Median absolute error of model with varied number of performance counters

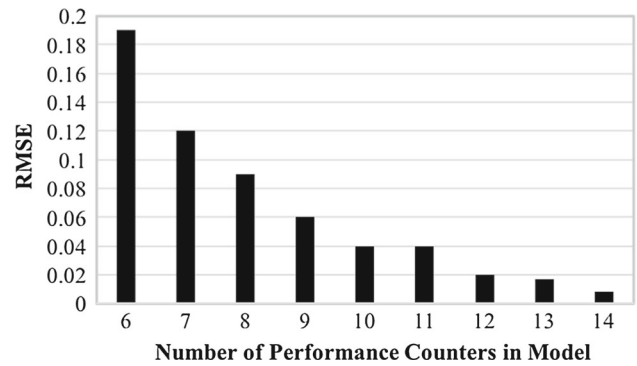


Fig. 13 Root mean square error of model with varied number of performance counters

speaking, the higher RMSE indicates that the underlying simplified model is of poor stability. Precisely, where there are 12 (i.e., SFetchInsts, WriteUnitStalled and WriteSize are removed) or more performance counters involved in the model, the median absolute errors are 4.75, 3.86 and 2.21%, respectively, and RMSEs keep below 0.02. This shows that the three simplified models are still of high precision and robustness.

6 Conclusion

In this work, we built a power model for iGPU in APU. A mechanism, namely kernel extension, was proposed to enable data profiling. The median absolute error of our model is 2.12%. To reduce the latency of our power model, we also evaluated the role of each performance counter and further simplified the full model by removing some performance counters that consume less power. Our optimization result showed that the model accuracy is still desirable when only 12 performance counters are involved in the model.

Acknowledgements This work is supported by the National Natural Science Foundation of China (61472431, 61272143 and 61272144).

Compliance with ethical standards

Conflicts of interest All authors declare that they have no conflicts of interest regarding the publication of this manuscript.

References

- AMD (2016) Amd codexl. <http://developer.amd.com/tools-and-sdks/opengl-zone/codexl/>
- Baghsorkhi SS, Delahaye M, Patel SJ, Gropp WD, Hwu WMW (2010) An adaptive performance modeling tool for gpu architectures. In: ACM sigplan notices, vol 45, pp 105–114
- Branover A, Foley D, Steinman M (2012) Amd fusion apu: Llano. *IEEE Micro* 32(2):28–37
- Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Skadron K (2008) A performance study of general-purpose applications on graphics processors using cuda. *J Parallel Distrib Comput* 68(10):1370–1380
- Chitty DM (2016) Improving the performance of gpu-based genetic programming through exploitation of on-chip memory. *Soft Comput* 20(2):661–680
- Corporation I (2016a) Intel core i7-920 processor. <http://ark.intel.com/product.aspx?id=37147>
- Corporation N (2016b) Geforce gtx 280. http://www.nvidia.com/object/product_geforce_gtx280_us.html
- Corporation N (2016c) What is cuda. http://www.nvidia.com/object/what_is_cuda_new.html
- Corporation N (2017) Machine learning. <http://www.nvidia.com/object/machine-learning.html>
- Diop T, Jeger NE, Anderson J (2014) Power modeling for heterogeneous processors. In: Proceedings of workshop on general purpose processing using GPUs, p 90
- Hong S, Kim H (2009) An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In: ACM SIGARCH computer architecture news, vol 37, pp 152–163
- Karami A, Khunjush F, Mirsoleimani SA (2015) A statistical performance analyzer framework for opengl kernels on nvidia gpus. *J Supercomput* 71(8):2900–2921
- Karami A, Mirsoleimani SA, Khunjush F (2013) A statistical performance prediction model for opengl kernels on nvidia gpus. In: 2013 17th CSI international symposium on computer architecture and digital systems (CADSD), pp 15–22
- Leng J, Hetherington T, ElTantawy A, Gilani S, Kim NS, Aamodt TM, Reddi VJ (2013) Gpuwatch: enabling energy optimizations in gpgpus. In: ACM SIGARCH computer architecture news, vol 41, pp 487–498
- Li J, Du Q, Li Y (2016) An efficient radial basis function neural network for hyperspectral remote sensing image classification. *Soft Comput* 20(12):4753–4759
- Luo C, Suda R (2011) A performance and energy consumption analytical model for gpu. In: 2011 IEEE ninth international conference on dependable, autonomous and secure computing (DASC), pp 658–665
- Stone JE, Gohara D, Shi G (2010) Opencl: a parallel programming standard for heterogeneous computing systems. *Comput Sci Eng* 12(3):66–73
- Wang Y, Roy S, Ranganathan N (2012) Run-time power-gating in caches of gpus for leakage energy savings. In: Design, automation & test in Europe conference & exhibition (DATE), 2012, pp 300–303
- Wu G, Greathouse JL, Lyashevsky A, Jayasena N, Chiou D (2015) Gpgpu performance and power estimation using machine learning. In: 2015 IEEE 21st international symposium on high performance computer architecture (HPCA), pp 564–576
- Zhang Y, Owens JD (2011) A quantitative performance analysis model for gpu architectures. In: 2011 IEEE 17th international symposium on high performance computer architecture (HPCA), pp 382–393
- Zhang H, Xiao N (2016) Parallel implementation of multilayered neural networks based on map-reduce on cloud computing clusters. *Soft Comput* 20(4):1471–1483
- Zhang Y, Hu Y, Li B, Peng L (2011) Performance and power analysis of ati gpu: a statistical approach. In: 2011 6th IEEE international conference on networking, architecture and storage (NAS), pp 149–158