**GPGPU-sim environment building and parameter fine-tuning**

Jiacheng Shi 201704565

Project

Man Lin

StFX University  
Spring 2021

# Abstract

As the graphics card configuration becomes higher and higher, the price becomes more and more expensive. Through the research of the Linux system under the virtual machine, This project tried to find a method that can tempo-rarily replace the real graphics card to test the energy consumption of the system and complete the energy consumption test when the equipment is insufficient. After reading some papers, this project found a platform that combines two modules of gpgpu-sim and gpuwatch.

By changing the underlying file method, the interface called by the graphics card file is changed, and it will be directly from the preprocessed fake cuda file Obtain graphics card parameters and calculate graphics card energy consumption.

At the same time, this project debugged various parameters in the second stage and show the different results of different parameters. This method's discovery has greatly reduced the hardware cost of small-scale experiments and can also provide convenience for various energy consumption test experiments in future work.

# Introduction

* **GPU**

GPU is a graphics processing unit. GPU can complete all data operations related to computer graphics. GPU's most significant role is to perform various calculations and graphics rendering operations, including vector setting, Ray tracing, Pixel operations. GPU is a set of graphics functions, and these functions are implemented by hardware. GPU graphics processing generally includes vertex processing, rasterization calculation, texture mapping, pixel processing, and final output. Simply put, the GPU is responsible for drawing and coloring polygons. The traditional GPU is based on the SIMD architecture, namely Single Instruction Multiple Data, single instruction multiple data. The ALU in the traditional VS and PS, that is, the Arithmetic and logic unit, can simultaneously complete the calculation of the vector's four channels in one cycle. But although the 4D SIMD architecture is very suitable for processing four-channel vector instructions, each channel's reduction will reduce the operating efficiency by a quarter. Therefore, the traditional GPU is not flexible in processing non-4D instructions. With the rapid update of GPUs, GPU computing speeds are getting faster and faster. Nowadays, GPUs mainly have two characteristics: ultra-long pipeline and parallel computing. To further improve the degree of parallelism, the number of pipelines is increased in the GPU. There are 16 sets of pixel shader pipelines and six sets of vertex shader pipelines in the GPU. In SIMD, a single control component dispatches instructions to each pipeline, and the exact instructions are executed simultaneously by all processing components. For example, Nvidia GPU contains 14 groups of Multiprocessors. Each group of processors has eight processors, but each group of multiprocessors only includes one Instruction Unit. Another structure is MIMD, multiple instructions, multiple Data. Each pipeline can independently execute programs different from other pipelines. MIMD can run branch programs more efficiently, but SIMD requires less hardware.

* GPGPU

General-puerpose graphics processing unit, GPGPU. It is a general-puerpose graphics processing unit that uses GPU to process graphics tasks to calculate general-purpose computing tasks initially processed by the CPU. These general calculations often have nothing to do with graphics processing. GPGPU is much faster than traditional CPU applications in performance.

* **Cuda**

Compute Unified Device Architecture, CUDA. is a general parallel computing architecture launched by NVIDIA. This architecture enables the GPU to solve complex computing problems. It includes the CUDA instruction set architecture (ISA) and the parallel computing engine inside the GPU. CUDA is NVIDIA's GPGPU model, which is written based on the C language. CUDA's advantages compared with CPU are larger memory bandwidth, more execution units, and lower price. The disadvantages are: there are many running units, which are only suitable for highly parallelized work. Inefficient multi-branch programs.

* **DVFS**

DVFS, Dynamic Voltage Frequency Scaling, this dynamic technology dynamically adjusts the chip's operating frequency and voltage according to the computing power requirements of the applications running on the chip, thereby saving energy. It is worth noting that reducing the frequency does not save energy because for a given task, only by reducing the frequency and reducing the voltage can the energy consumption be reduced. DVFS process: 1) The acquisition system is attached to the signal, and the current system load is calculated. 2) According to the system's current load, predict the system's required performance in the following period. 3) Find the frequency required for the predicted performance and modify the clock of the chip. 4) Calculate the voltage according to the new frequency. When modifying the voltage and frequency, you must first change the voltage and then the frequency. Otherwise, the motherboard will not be able to boot due to an insufficient power supply.

# Materials

* Vmware WorkStation Pro: This is a VMware workstation designed for professionals who need virtual machines at work. Vmware workstation pro supports most operating systems and has excellent user experience, complete functions, and excellent performance. The use of workstation pro requires a 64bits processor and a 64bits host operating system.
* Cuda toolkits 9.0 & Nvidia driver 384：The Nvidia driver of the system will determine the highest Cuda and Cuda toolkit that the system can support. It is worth noting that the Nvidia driver is backward compatible. Cuda and Cuda toolkit is different. The former refers to the Cuda installed by the system, which is officially provided by Nvidia and is usually installed in /usr/local/cuda. Cuda toolkit is just a toolkit used to install PyTorch, which is a subset of Cuda. The version relationship between Cuda and Nvidia Driver is shown in Figure.1

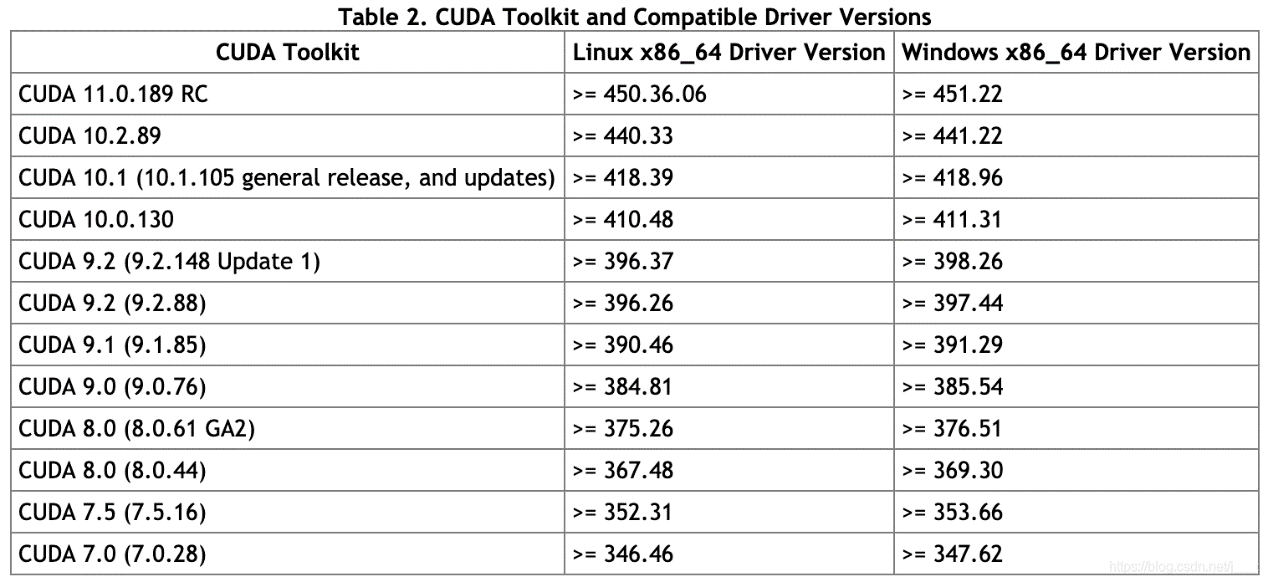


Figure 1

* Ubuntu 64bits 16.04: Ubuntu is a Linux operating system based on desktop applications. Its name comes from the word ubuntu in southern Africa's Zulu language, which means human nature. It is a traditional value in Africa. Ubuntu is based on the Debian release and the gnome desktop environment. After version 11.04, the ubuntu release has abandoned the Gnome desktop environment and changed it to UNITY. Ubuntu has changed the situation where LINUX is challenging to use. The 16.04 used by this project is the highest version of UBUNTU that supports GPGPU-SIM. The ubuntu version still uses GCC 5.0, but the interface and driver versions are relatively new. At the same time, UBUNTU still maintains the 16.04 update.

# Methods

* Cuda programming [1]: CUDA is a general-purpose parallel computing platform and programming model built on Nvidia's CPUs. Based on CUDA programming, the parallel computing engine of GPUS can be used to solve more complex computing problems more efficiently. In recent years, GPU One of the most successful applications is deep learning. GPU-based parallel computing has become the standard for deep learning. The GPU is not a computing platform that runs independently. The GPU needs to work with the CPU and can be regarded as a coprocessor of the CPU. Therefore, CUDA programming is a heterogeneous computing architecture based on CPU+GPU. In the heterogeneous computing architecture, the GPU and CPU are connected through the PCIe bus to work together. The location of the CPU is HOST, and the location of the GPU is DEVICE (Figure 2).
* 图示

  描述已自动生成

Figure 2

Cuda programming steps:

1). Allocate host memory and initialize data.

2). Allocate device memory and copy data from Host to device.

3). Call the core function of Cuda to complete the operation of the corresponding function on the device.

4). Copy the result of the operation on the device to the Host

5) Release the memory allocated on the device and Host.

Cuda programming needs to perform parallel calculations based on the kernel. The kernel is a function executed in parallel in threads on the device. The kernel uses the \_\_global\_\_ symbol declaration. When calling, you need to use <<<grid, block>>> to specify the number of threads to be executed by the kernel. Each thread will execute the kernel, and Each thread will be assigned a unique thread ID. This ID can be obtained through threadIdx.

# Experience

* **Environment build:**

1. **Install Library:**

文本

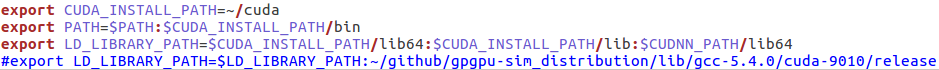
描述已自动生成

1. **Install Cuda:**

文本

中度可信度描述已自动生成

When the gedit file opened, added four lines code at the end of this file.



Then type:



To apply the address.

1. **Install github:**

图形用户界面, 文本, 应用程序

描述已自动生成



After type make, the environment will be built. And add two lines of codes.



at the end of config file. The address of this file is

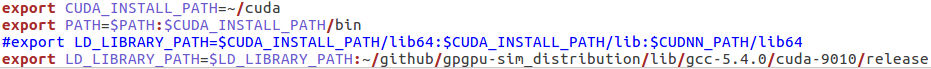
/home/sjcpc/github/gpgpu-sim\_distribution/configs/tested-cfgs/SM2\_GTX480/gpgpusim.config.

Copy three files in the folder “SM2\_GTX480” to Test folder that has the test.cu file.

1. **Run program:**



When gedit file opened, changed the comment of the last four lines codes.



And use “bash” to apply the change. If the folder named test.

文本

描述已自动生成

文本

描述已自动生成

The result of LDD showed the libcudart.so.9.1 already replaced to gpgpu-sim’s fake cuda 9.1.

* **Power consumption estimation:**

**-gpgpu\_clock\_domains <Core Clock>:<Interconnect Clock>:<L2 Clock>:<DRAM Clock>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Core** | **Interconnect** | **L2 Cache** | **DRAM** | **Average Power** |
| **Origin** | **700** | **700** | **700** | **924** | **38.6234** |
| **Core+100** | **800** | **700** | **700** | **924** | **38.5381** |
| **Core+200** | **900** | **700** | **700** | **924** | **38.1725** |
| **Core+300** | **1000** | **700** | **700** | **924** | **37.7127** |
| **Core-100** | **600** | **700** | **700** | **924** | **40.2915** |
| **Core-200** | **500** | **700** | **700** | **924** | **40.8856** |
| **Core-300** | **400** | **700** | **700** | **924** | **41.035** |

图表, 折线图

描述已自动生成

Table 1. Core Clock Performence

First, the project tested the impact of the core clock on energy consumption by adjusting the core clock. When I increase the rate of the core clock, the average power gets lower and lower.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Core** | **Interconnect** | **L2 Cache** | **DRAM** | **Average Power** |
| **Origin** | **700** | **700** | **700** | **924** | **38.6234** |
| **Interconnect +100** | **700** | **800** | **700** | **924** | **40.0083** |
| **Interconnect +200** | **700** | **900** | **700** | **924** | **40.317** |
| **Interconnect +300** | **700** | **1000** | **700** | **924** | **40.7078** |
| **Interconnect -100** | **700** | **600** | **700** | **924** | **38.6099** |
| **Interconnect -200** | **700** | **500** | **700** | **924** | **38.023** |
| **Interconnect -300** | **700** | **400** | **700** | **924** | **36.0437** |

图表, 折线图

描述已自动生成

Table 2. Interconnect Clock Performance

Secondly, the project tested the influence of the interconnect clock on energy consumption by adjusting the interconnect clock. When I increase the rate of Interconnect clock, the average power gets higher and higher.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Core** | **Interconnect** | **L2 Cache** | **DRAM** | **Average Power** |
| **Origin** | **700** | **700** | **700** | **924** | **38.6234** |
| **L2 +100** | **700** | **700** | **800** | **924** | **38.7251** |
| **L2 +200** | **700** | **700** | **900** | **924** | **38.6548** |
| **L2 +300** | **700** | **700** | **1000** | **924** | **38.7012** |
| **L2 -100** | **700** | **700** | **600** | **924** | **38.6304** |
| **L2 -200** | **700** | **700** | **500** | **924** | **38.6604** |
| **L2 -300** | **700** | **700** | **400** | **924** | **38.5697** |

图表, 折线图

描述已自动生成

Table 3. L2 Clock Performance

Third, the project tested the impact of L2 clock on energy consumption by adjusting the L2 clock. When I increase the rate of L2 clock, the average power has a smaller fluctuation, but the overall trend is getting higher and higher.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Core** | **Interconnect** | **L2 Cache** | **DRAM** | **Average Power** |
| **Origin** | **700** | **700** | **700** | **924** | **38.6234** |
| **DRAM +100** | **700** | **700** | **700** | **1024** | **38.6652** |
| **DRAM +200** | **700** | **700** | **700** | **1124** | **38.8233** |
| **DRAM +300** | **700** | **700** | **700** | **1224** | **38.7603** |
| **DRAM -100** | **700** | **700** | **700** | **824** | **38.6905** |
| **DRAM -200** | **700** | **700** | **700** | **724** | **38.6895** |
| **DRAM -300** | **700** | **700** | **700** | **624** | **38.6922** |

图表, 折线图

描述已自动生成

Table 4. DRAM Clock PerformenceFinally, the project tested the impact of the DRAM clock on energy consumption by adjusting the DRAM clock. When I increase the rate of the DRAM clock, the average power drops first and then rises. At the same time, I found that for different graphics cards and process complexity, there is a more reasonable rate of DRAM clock. In this experiment, 947 is the most suitable DRAM clock.

# Conclusion

This study showed us the possibility of a new CUDA-programmed energy consumption monitoring. This paper starts with the environment construction. The existing gpgpu-sim construction process is outdated. This paper gives a common system now: gpgpu-sim construction process under ubuntu16.04. At the same time, it is aimed at the old version of gpgpu- The bugs brought by sim have been dealt with, and additional environment variables have been added.

The gpgpu-sim construction method provided in this paper is reproducible and modifiable. Although the Cuda code of gpgpu-sim currently uses gcc5.4, the system version is limited to ubuntu 16.04 LTS, but with the gpgpu-sim kernel code, The method given in this article can be directly applied to higher version systems.

Subsequently, this paper carried out Cuda programming and energy parameter testing and gave the influence of key parameters on energy consumption. There are four key parameters in total, namely core clock, interconnect clock, L2 clock, and DRAM clock. When debugging the parameters, this article found that as the Core clock rises, the efficiency of the execution of the program will increase, so the running time will be shorter, and the energy consumption will be reduced. With the rise of the interconnect clock, increasing the interconnect frequency will increase the power consumption. The most interesting performance is L2 clock and DRAM clock. These two parameters have corresponding minimum values according to hardware and programs. In the code used in this article, 947 is the value of best performance.

In general, the research in this article has effectively helped researchers with a shortage of hardware devices to detect the energy consumption of various GPUs. By updating and building GPGPU-SIM, they are adapted to newer operating systems and Linux libraries, and are The clock frequency was tuned to point out how different parameters affect energy consumption, allowing researchers to adjust the parameters as needed. In subsequent research, we will try to establish a visual interface to make Fine-tuning and result in output become easier.

# References

[1] Cheng, J., Grossman, M., & McKercher, T. (2014). *Professional CUDA c programming*. John Wiley & Sons.

[2] Bakhoda, A., Yuan, G. L., Fung, W. W., Wong, H., & Aamodt, T. M. (2009, April). Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software* (pp. 163-174). IEEE.

[3] Zhang, H., Putic, M., & Lach, J. (2014, June). Low power gpgpu computation with imprecise hardware. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)* (pp. 1-6). IEEE.

[4] Lew, J., Shah, D. A., Pati, S., Cattell, S., Zhang, M., Sandhupatla, A., ... & Aamodt, T. M. (2019, March). Analyzing machine learning workloads using a detailed GPU simulator. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 151-152). IEEE.

[5] Jiao, Q., Lu, M., Huynh, H. P., & Mitra, T. (2015, February). Improving GPGPU energy-efficiency through concurrent kernel execution and DVFS. In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (pp. 1-11). IEEE.

[6] Mei, X., Yung, L. S., Zhao, K., & Chu, X. (2013, November). A measurement study of GPU DVFS on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (pp. 1-5).