International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

Green Computing using GPU in Image Processing

Chahat Monga
Assistant Professor, Guru Nanak college, Department of Computer Science, Ferozepur, Punjab, India

ABSTRACT

Green computing is the process of reducing the power consumed by a computer and thereby reducing carbon emissions. The total power consumed by the computer excluding the monitor at its fully computative load is equal to the sum of the power consumed by the GPU in its idle state and the CPU at its full state. Recently, there have been tremendous interests in the acceleration of general computing applications using a Graphics Processing Unit (GPU). Now the GPU provides the computing powers not only for fast processing of graphics applications, but also for general computationally complex data intensive applications. On the other hand, power and energy consumptions are also becoming important design criteria. Consequently, software designs have to consider the power/energy consumptions together with performance when they are developing software.The GPU therefore does the 100% of the CPU work in its idle state .Hence the power consumed by the GPU will be low. Also when the GPU is doing all the work the CPU will remain at a load less than its idle load. Hence the power consumed will be equal to the power consumed by the CPU at a load less than its idle load plus the power consumed by a GPU.

Keywords : GPU (Graphical Processing Unit), CUDA, OpenCV, nVidia, Image Processing.

I. INTRODUCTION

Green computing refers to those practices adopted in the IT industry so as to reduce carbon emissions [1]. The practices may be both in hardware or software terms. The hardware obtained these days are usually very efficient if used properly . So there is more research in the coding part than in hardware be it with making the webpage of websites black so as to reduce power consumption to making shorter codes ,it all accounts for green computing [2-4]. In our paper we have used Graphical processing units for reducing the carbon emissions. Graphical processing units are the processing units which are used to calculate which pixel is to be lit up at what instant of time. Since these calculations have to be very fast so that the user doesn't feel any time delay in getting the required graphical output on the screen the processing speed and thereby the processing capacity of the graphical processing unit is very high. It is 30 -50 times higher than the processing speed of the CPU. The reason why the GPU has higher processing speed than the CPU is that the CPU has other functions other than processing like storing of data, retrieving of data from the cache memory, primary memory, secondary memory, monitoring the other parts of the system along with many other functions. Whereas the GPU has only one job that is to perform calculations. Thereby due to a singularity function the processing speed of the GPU is very high. Presently GPU's are used for game physics calculations so as to provide a very interactive environment to the gaming user. Green computing, also called green technology, is the environmentally sustainable to use of computers and related resources like - monitors, printer, storage devices, networking and communication systems - efficiently and effectively with minimal or no impact on the environment. Green computing whose goals are to reduce the use of hazardous materials, maximize energy efficiency during the product's lifetime, and promote the recyclability or biodegradability of defunct products and factory waste. Conserving resources means less energy is required to produce, use, and dispose of products ,Saving energy and resources saves money .Green computing even includes changing government policy to encourage recycling and lowering energy use by individuals and businesses.

WHAT IS GPU?

Digital Image processing is part of daily computations. The graphics processing unit (GPU) has become an essential part of today's conventional computing systems. In few years, there has been a marked raise in the performance and capabilities of GPUs. Graphics Processing Units is powerful, programmable, and highly parallel computing, which are increasingly targeting for general-purpose computing applications. The modern GPU is not only a powerful graphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth that substantial as CPU. The GPU's has capability computing broad range of computation, complex problems solve & especially for high range of graphical data. This effort in general purpose computing on the GPU, also known as GPU computing, which proposed GPU as an alternative to traditional microprocessors in high-performance computer systems of the future. The Graphics processing units (GPUs) are being increasingly used to accelerate a wide variety of applications. Recent advances in GPU architectures have not only improved the baseline performance but have also provided programmers with new options to optimize their applications for better performance. Despite these advances, writing efficient GPU applications remains a challenging task. Chief among these challenges is the problem of memory performance. GPGPU refers to the use of graphics processing units (GPUs) to perform general processing tasks, and it involves explicitly copying large amounts of data over the PCIe bus, between the CPU and GPU. This process can be relatively time consuming. Further complications and slowdowns can occur if the data does not fit on the GPU's memory, multiplying the problem by necessitating frequent CPU-GPU data transfers. Unlike a CPU, a GPU lacks the benefit of automatic memory paging. As a result, GPU memory management is explicitly handled by the programmer. This makes memory optimization a delicate process. The large number of threads running on a GPU will multiply any mistakes and inefficiencies in the code. This can seriously hamper an application's performance. Differences in algorithms and GPU architectures serve to complicate this matter even further. Data that arrives on a GPU's memory must then be accessed by a huge number of threads via the memory hierarchy. The threads need to be able to access this data efficiently, and through limited means. Inefficient GPU memory access is a common occurrence, and serves to hamper performance even further, particularly in workloads that are data intensive. Optimizing GPU memory performance is a complex and challenging problem. There are a wide assortment of memory optimization techniques and configurations. The suitability and efficiency of these optimization techniques may vary between different applications and data sizes.

WHY GPU?

Today's GPUs rapidly solve large problems having substantial inherent parallelism by using hundreds of parallel processor cores executing tens of thousands of parallel threads. They're now the most pervasive massively parallel processing platform ever available, as well as the most cost effective. GPU is a multi-core multithreaded multiprocessor that excels at both graphics and computing applications. Exposing high-definition graphics scenes is a problem with fabulous inherent parallelism. A graphics programmer writes a single-thread program that draws one pixel. The GPU runs multiple instances of this thread in parallel are known as multi-thread parallel computing. It draws multiple pixels in parallel at a time. As Software scalability rapidly increase GPU parallelism, simultaneously in hardware increasing transistor density with performance. GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications. Pioneered in 2007 by NVIDIA, GPU accelerators now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. GPUs are accelerating applications in platforms ranging from cars, to mobile phones and tablets, to drones and robots.

II. METHODS AND MATERIAL

ARCHITECTURE OF GPU

In GPU computing, code is typically executed both on a conventional CPU and on a GPU. Sequential code with many random memory accesses, branches and complicated logic is most efficiently executed on the CPU. Parallel, arithmetic code may see a significant speed-up when executed on the GPU [12]. Efficient development of parallel applications for GPUs is dependent upon a sufficiently high-level programming language. Examples of such languages are CUDA (for NVIDIA GPUs) and OpenCL (general-purpose). These languages helps abstract the complicated nature of the GPU hardware, hiding low-level features. By focusing on exploiting the inherent parallelism of the problem, significant speed-ups can be gained with reasonable effort. The parallel part of a CUDA program consists of kernels that are executed on the GPU. Many copies of the kernel are executed in parallel as threads, grouped together in thread blocks. Threads within the same block may communicate through shared memory. GPUs are optimized for Floating Point Operations, as well as being optimized for parallelism. On typical Multiprocessor consists of 4 cores (as with most of the Intel i-Series multiprocessors), while a GPU is composed of tens of processors. This is because CPUs could be considered memory based processors while GPUs could be called ALU based, which allows the GPU to perform more operations in parallel resulting in the high GFLOPS (FLOPS = Floating Point Operation Per Second) compared to the CPU.

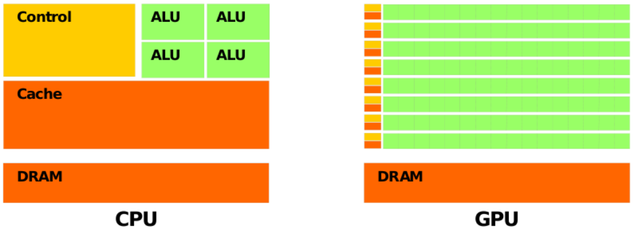


Figure 1. CPU v/s GPU Architecture.

Why do HPC customers care about Perf/watt when upgrading their systems?

The reality of today is that computing performance cannot be considered by itself. The amount of power required to drive these systems is an important part of the overall cost of ownership. For example, powering a 2.3 PFlops machine that is built with CPU requires upto ~7 megawatts. Assuming 1 megawatt can power 1000 homes for a year, it's equal to the powering up 7000 homes for a year. Clearly the CPU technology just cannot keep up with the explosive demand for computing. It's too expensive and power-consuming. This has led the HPC customers to look for options. CPUs are optimized for sequential processing, but not very efficient for parallel computing. GPUs on the other hand are tailor-made for parallel computation. Combining a GPU wth a CPU makes a winning combination with each doing what one does best. The result is 10x higher performance per socket and 5x the energy efficiency. CPUs alone cannot keep up with the demand for computing performance, the era of GPU accelerated computing is here because the benefit it offers is meaningful in terms of perf/watt. In computing, FLOPS is a measure of computer performance, useful in fields of scientific calculations that make heavy use of floating-point calculat ions.

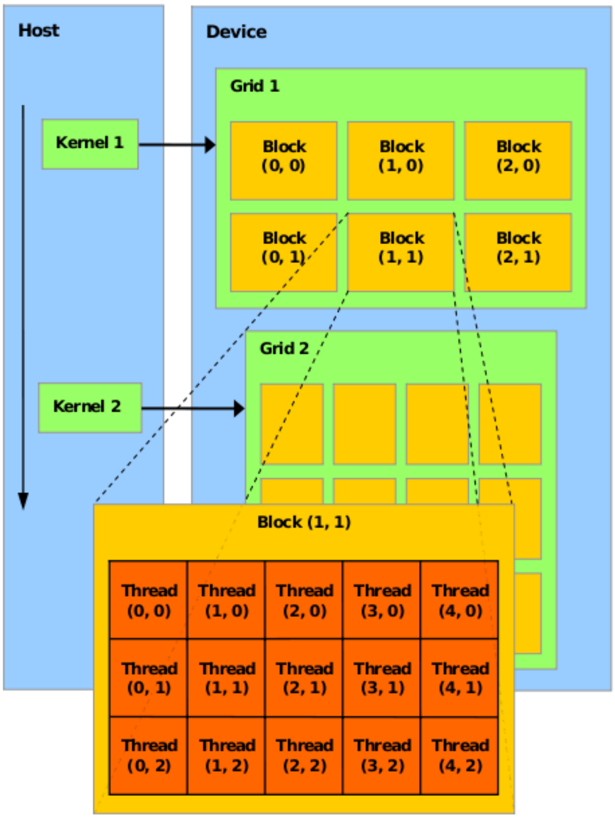
CUDA

The massive amount of digital data on the web has brought a number of challenges such as prohibitively high costs in terms of storing and delivering the data. Image compression therefore involves reducing the amount of memory it takes to store an image in order to reduce these large costs. Image processing generally, is a very compute intensive task. Taking into consideration the image representation and quality, systems or application/techniques for image processing must have special capabilities for unquestionable results in their image processing. The rapid growth of digital imaging applications, including desktop publishing, multimedia, teleconferencing, computer graphics and visualization and high definition television (HDTV) has increased the need for effective image compression techniques. Many image processing algorithms require dozens of floating point computations per pixel, which can result in slow runtime even for the fastest of CPUs. Because of the need for high performance computing in today's computing era, this paper presents a study based on Cordic based Loeffler DCT for image compression using CUDA. Image processing takes the advantage of CUDA processing because of the parallelism that pixels exhibit in an image and that can be offered by the CUDA architecture. In that way the CUDA architecture is found to be relevant for image processing [1], [2], [3] and [4]. This paper presents a performance based evaluation of the DCT image compression technique on both the CPU and GPU using CUDA. The focus of the paper is on the Cordic based Loeffler DCT. We begin by giving a brief review of literature (background), followed by a description of the research methodology employed, including an overview of image compression, GPUs, CUDA architecture, Cordic based Loeffler DCT and related theory. This is then followed by the results of the experiments and their interpretation and lastly, conclusions and future work. CUDA is the missing link between the developer and the GPU. It was developed by NVIDIA and is implemented in all NVIDIA GPUs starting the G80s. Before having programming architectures dedicated to programming the GPU, a programmer had to choose either between dealing with the complex APIs of the GPUs or "tricking" it by passing a texture, that contains the data or the instructions, to the GPU and then receiving a the data in the form of a texture, which typically creates a lot of overhead. CUDA processors are programmed in CUDA C, which is basically C/C++ with some CUDA extensions, which will be mentioned and explained later on. It is important to know that in early versions of CUDA the GPU had to be programmed in C, while the CPU could be programmed in either. This is important when writing code, since the developer must know at all times, whether the code is compiled for the CPU or the GPU. Starting from CUDA 3.0 more C++ features had been enabled for the code compiled for the GPU.

CUDA Structure and Terminology Thread : The smallest unit executing an instruction. Block : Contains several threads. Warp : A group of threads physically executed in parallel (usually running the same application). Grid : Contains several thread blocks. Kernel : An application or program, that runs on the GPU. Device : The GPU. Host : The CPU.

Addressing Scheme in CUDA

Threads and Blocks need to have a unique ID in order to access them while writing code. This is important, since threads and blocks are the main components when it comes to writing efficient parallel code. Within each block a thread is uniquely accessible through the Thread ID, which is an integer between 0 and n, where n is the total number of threads within the same block. In a more complex approach – when dealing with 2D or 3D blocks – the Thread ID is calculated as a function, rather than having a fixed integer, which represents the number of the thread inside the block. Inside a 2D Block threads are positioned at (0,0), (0,1), … (n-1, n-1), as shown in the previous figure. Since the Thread ID must be of type uint3 – which will be considered a normal integer for now – something like (3,2) for the Thread ID is not applicable.



III. CONCLUSION

As a promising architecture for supercomputers, GPU-based computing is receiving significant attention for its performance and power efficiency. In order to help the HPC community better understand the power characteristics of different application kernels on the GPU, we have conducted this study to characterize the performance and power of various application kernels under varying frequency settings. Based on our findings, the GPU application kernels' performance and power consumption are largely determined by two characteristics: the rate of issuing instructions and the ratio of global memory transactions to computation instructions. For future work, we intend to conduct our study on other GPUs, including from AMD and Intel. Given the software abstraction used in this work is OpenCL which is supported by most accelerator platforms, a fair comparison running the same code across platforms is possible. In addition, we believe a complete and comprehensive modeling of GPU performance and power consumption would be very useful.

IV. REFERENCES

[1]. Jain, Anil. K. (2013) Fundamentals of digital image processing . Prentice Hall: Englewood Cliffs, NJ.

[2]. Jackson, David Jeff & Hannah, Sidney Joel, (2011) "Comparative Analysis of image Compression Techniques," System Theory 2013, Proceedings SSST '93, 25th Southeastern Symposium, pp 513517.

[3]. Zhang, Hong., Zhang, Xiaofei.& Cao, Shun (2000) "Analysis & Evaluation of Some Image Compression Techniques," High Performance Computing in Asia Pacific Region, 2000 Proceedings, 4th Int. Conference, Vol. 2, pp 799-803.

[4]. Gonzalez, Rafael & Woods, Richard E. (2002) Digital Image Processing, 2nd ed. n. Edition 2002: Prentice-Hall Inc.

[5]. Sonal, Dinesh Kumar (2007), "A Study of Various Image Compression Techniques", Guru Jhambheswar University of Science and Technology, Hisar.

[6]. Gleb V. Tcheslavski (2008), "Image compression fundamentals".Springer 2008, ELEN 4304/5365 DIP

[7]. NVIDIA Corporation (2012), "NVIDIA CUDA Programming Guide v. 4.2", http://www.nvidia.com.

[8]. Owens, John D., Houston, Mike, Luebke, David., Green, Simon., Stone, John E. and Phillips, James C. (2008), "GPU Computing", Proceedings of the IEEE, Vol. 96, no. 5, pp. 879-897, May / 2008.

[9]. Halfhil, Tom R., (2008), "Parallel Processing With CUDA". Microprocessor Report, Scottsdale, Arizona, 28 / January / 2008.

[10]. Gupta, Maneesha. & Garg, Amit Kumar. (2012) "Analysis of Image Compression Algorithm Using DCT" International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com, Vol. 2, Issue 1, pp.515-521.

[11]. Chi-Chia Sun, Benjamin Heyne, Shanq-Jang Ruan and Juergen Goetze.,(2006), "A Low-Power and High-Quality Cordic Based Loeffler DCT"., Information Processing Lab, Dortmund University of Technology, Germany.