Imperial Journal of Interdisciplinary Research (IJIR)
Vol-2, Issue-5, 2016
ISSN: 2454-1362, http://www.onlinejournal.in

GPU Computing with CUDA

Priyanka More[1], Prasad C. Shinde[2] & Prof. Ajit Patil
Department of Computer Engineering, BVCOEL, Pune, India.

Abstract*: The GPU (Graphics Processing Unit) has become a more popular processor for today's digital lifestyle. The GPU is good at data-parallel processing. The same computation executed on many data elements in parallel with high arithmetic intensity. The GPU computing is to use a CPU and GPU together in a heterogeneous computing model. The sequential part is run on CPU and the highly parallel part on GPU.*

1. Introduction

The GPU's performance and potential offer a great opportunity for future computing systems, yet the architecture and programming model of the GPU are markedly different than most other commodity single chip processors. Computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU. To enable this new computing paradigm, NVIDIA invented the CUDA (Compute Unified Device Architecture) parallel computing architecture. The modern GPUs are not only a powerful graphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth that substantially outpaces its CPU counterpart.

Officially introduced in 2006, CUDA is steadily winning in scientific and engineering fields. A few years ago, pioneering programmers discovered that GPUs could be harnessed for tasks other than graphics. However, their improvised programming model was clumsy, and the programmable pixel shaders on the chips weren't the ideal engines for general purpose computing.

2. History

The first graphics cards, introduced in August of 1981 by IBM, were monochrome cards designated as Monochrome Display Adapters (MDAs). The displays that used these cards were typically text-only, with green or white text on a black background. Color for IBM-compatible computers appeared on the scene with the 4-color Hercules Graphics Card (HGC), followed by the 8-color Graphics Adapter (CGA) and 16-color Enhanced Graphics Adapter (EGA).

GPUs were designed as graphics accelerators, supporting only a specific fixed-function pipelines. Starting in the late 1990s, the hardware became increasingly programmable, in 1999 NVIDIA has launched a first GPU.

But GPGPU was far from easy back then, even for those who knew graphics programming languages such as OpenGL. Developers had to map scientific calculations onto problems that could be represented by triangles and polygons. GPGPU was practically an off-limits for those who can't memorized the latest graphics APIs until a group of Stanford University researchers set out to reimagine the GPU as a "Streaming Processor".

3. GPU

3.1 GPU Architecture

NVIDIAs CUDA is a general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems. The programmable GPU is a highly parallel, multi-threaded; many core coprocessors specialized for compute intensive highly parallel computation. All the GPU threads run the same code and are very light weighted and have a low creation overhead.

A thread block is a three, two or one-dimensional group of threads. Threads within a block can co-operate among themselves by sharing data through some shared memory accesses. Threads in different blocks cannot co-operate and each block can execute in any order relative to other blocks. The number of threads per block is therefore restricted by the limited memory resources of processor core. A graphics programmer writes a single-thread program that draws one pixel, and the GPU runs multiple instances of this thread in parallel drawing multiple pixels in parallel. Graphics programs, written in shading languages such as Cg or High Level Shading Language (HLSL), thus scale transparently over a wide range of thread and processor parallelism.

Parallelism is the future of computing. Future microprocessor development efforts will continue to concentrate on adding cores rather than increasing single-thread performance. One example of this trend, the heterogeneous nine-core Cell broadband engine, is the main processor in the Sony Playstation 3 and has also attracted substantial interest from the scientific computing community.

The GPU is designed for a particular class of applications with the following characteristics. Over

the past few years, a growing community has identified other applications with similar characteristics and successfully mapped these applications onto the GPU.

3.1.1. Real time computational rendering requires billions of pixels per second, and this each pixel requires hundreds or more operations. GPU have to gives a high amount of compute performance to satisfy the demand of complex real time applications.

3.1.2. Parallelism is more powerful. The graphics pipeline is well suited for parallelism. Operations run in parallelism delivers a fine grained quality of applications

3.1.3. Throughput is having a more importance than latency. GPU implementations of graphics pipeline prioritize throughput over latency. The human visual system operates on millisecond time scales, while operations within a modern processor take nanoseconds, the pipeline is allowing feed-forward, removing the penalty of control hazards, further allowing optimal throughput of primitives through the pipeline.

In the development of GPU as a general-purpose computing engine has been the advancement of the programming model and programming tools. Now a day the challenges to GPU vendors and researchers has been to balance between the low-level access to the hardware to enable performance and high level programming languages and tools that allow programmer efficiency and effectiveness, all in the rapid development of hardware.

3.2 CUDA Parallel Architecture

Computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU. To enable this new computing paradigm, NVIDIA invented the CUDA parallel computing architecture that is now shipping in GeForce, ION Quadro, and Tesla GPUs, representing a significant installed base for application developers

CUDA is hardware and software co-processing architecture for parallel computing that uses NVIDIA GPUs to execute program written with C, C++, FORTRAN, OpenCL, DirectCompute, and other Languages.

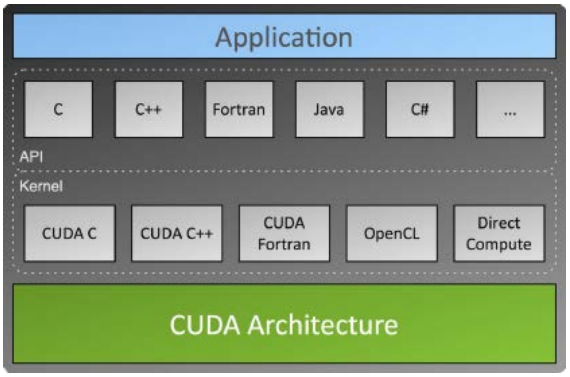

Fig. 1 CUDA architecture

3.3 Components of GPU

There are several components on a typical graphics card such as:

3.3.1. Graphics Processor: The graphics processor is the brains of the card, and is typically one of three configurations:

3.3.2. Graphics co-processor. A card with this type of processor can handle all of the graphics chores without any assistance from the computer's CPU. Graphics co-processors are typically found on high end video cards.

3.3.3. Graphics accelerator. In this configuration, the chip on the graphics card renders graphics based on commands from the computer's CPU. This is the most common configuration used today.

3.3.4. Frame buffer. This chip simply controls the memory on the card and sends information to the digital to analog converter (DAC). It does no processing of the image data and is rarely used anymore.

3.3.5. Memory. The type of RAM used on graphics cards varies widely, but the most popular types use a dual ported configuration. Dual ported cards can write to on section of memory while it is reading from another section, decreasing the time it takes to refresh an image.

3.3.6. Graphics BIOS. Graphics cards have a small ROM chip containing basic information that tells the other components of the card how to function in relations each other. The BIOS also performs diagnostic tests on the card's memory and input output(I/O) to ensure that everything is functioning correctly.

3.3.7. Digital to Analog Converter (DAC).
The DAC on a graphics card is a commonly

known as a RAMDAC because it takes the data it converts directly from the card's memory. RAMDAC speed greatly affects the image you see on the monitor. This is because the refresh rate of the image depends on how quickly the analog information gets to the monitor.

3.3.8. Display Connector. Graphics cards use a standard connector. Most cards use the 15pins connector that was introduced with Video Graphics Array (VGA).

3.3.9. Computer (BUS) connector. This is usually Accelerated Graphics Port (AGP). This port enables the video card to directly access system memory. Direct memory access helps to make the peak bandwidth four times higher than the peripheral Component Interconnect (PCI) bus adapter card slots. This allows the central processor to do other tasks while the graphics chip on the video card accesses system memory.

4. GPU Programming Model

Researchers and developers have adopted a CUDA and GPU computing for wide range of applications. Library and tools developers are trying to make GPU development more productive. GPU programming includes CUDA C, CUDA C++, Portland Group (PGI) CUDA Fortran, DirectCompare and OpenCL

The programmable units of the GPU follow a single program multiple data (SPMD) programming model. For the efficiency, the GPU accesses and processes the more elements in parallel using the same program. Each element is independent from the other elements. All GPU programs have to be structured prefer to allow different elements to take different paths through the same program.

CUDA programming model is designed to overcome the problem of scalability while maintaining a low learning curve for programmers familiar with standard programming languages such as C.

The benefit of the GPU is its large fraction of resources devoted to computation. Now a day's GPUs support an arbitrary control flow per thread but impose a penalty for incoherent branching. GPU vendors have largely adopted this approach. Elements are grouped together into a blocks, and this blocks are processed in a parallel manner.

The size of the block is known as the "branch granularity" and has been decreasing with recent GPU generations, now days it is on the order of 16 elements. While writing GPU programs, then, branches are permitted but not free. Programmers who structure their code such that blocks have coherent branches will make the best use of the hardware.

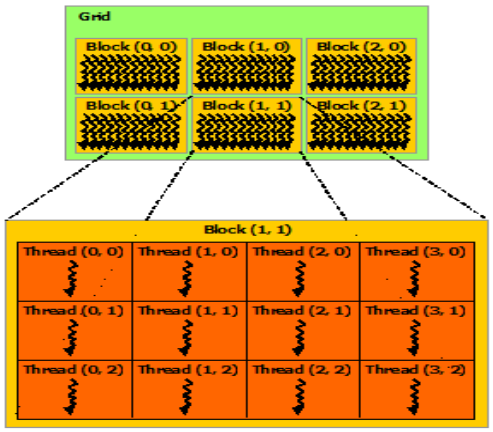

Figure 2. GPU Grid

Figure 2. shows that the one GPU Grid contains a more than one thread to process a large amount of data in parallel.

5. GPU with CPU Co-Processing

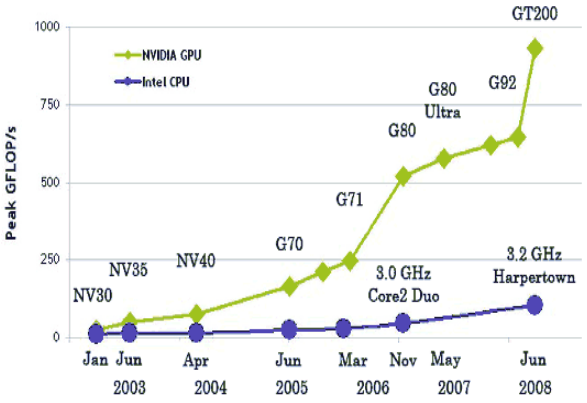

Fig. 3: CPU vs GPU

The Fig. 3 shows the difference between GPU and CPU performance NVIDIAs GPUs are always got a higher peak value than Intel CPU as per year. It shows us how GPU is faster than the CPU. But the GPU is not an independent processor for processing a data it requires a CPU to handle a processes and data.

The CPU+GPU co-processing systems has been evolved because the CPU and GPU have uses the attributes that allow to applications to perform best using both types of processors. CUDA programs are uses co-processing programs and the serial portions execute by the CPU, while parallel portions execute on the GPUL. The co-processing optimizes the total application performance.

By using a co-processing, we can use the right core for the right job. We use the CPU core for a code's serial portions, and we the GPU for the parallel portions of the code. This approach will give a more performance per unit area or power than either on CPU or GPU cores.

Use of CUDA shared memory on NVIDIA GPUs also helps, reducing over fetch and enabling strategies for "blocking" the computation in this fast on-chip memory. The experience will then have shown on when algorithms and applications can follow these design principles for GPU computing such as the PDE solvers, linear algebra packages, and database systems referenced above, and the game physics and molecular dynamics applications examined in detail next they can achieve 10 to 100x speedups over even mature, optimized CPU codes.

6. Application Performance

Now a day the complexity of current applications makes their execution time to be extremely high. applications consist of a mixture of fundamentally serial control logic and inherently parallel computations. This directly matches the CUDA co-processing programming model, which is a sequential control thread, it is capable of launching a series of parallel kernels. The use of kernels is launched from a sequential program also make it relatively easy to parallelize an application in individual components rather than rewrite the entire application.

Many real-world applications are composed of many different algorithms each with varying degrees of parallelism. CUDA architecture provides a mechanism to control and schedule a wide variety of tasks on both CPU and GPU. Some tasks are primarily serial and execute on the CPU, such as compilation, data structure management and co-ordination with the operating system and user interactions.

Many other tasks, such as building an acceleration structure or updating animations may run either on the CPU or the GPU depending on the choice of an algorithms and the performance required.

7. The Future work for GPU Computing with CUDA

In this digital world the importance of GPU computing is increasing, GPU hardware and software are changing at a remarkable pace. In the upcoming years, we expect to see several changes to allow more flexibility and performance from future GPU computing systems such as:

7.1. Delivering heterogeneous/hybrid, energy-efficient computing.

7.2. Allows developers to unlock the potential of complex applications for consumers.

7.3. Parallel CPU/GPU processing will become a norm in all program.

7.4. More tightly coupled CPU and GPU (AMD's fusion or NVIDIAs NForce).

7.5. Instruction overheads. Modern processors evolved in an environment where power was plentiful and absolute performance or performance per unit area was the important figure of merit. The resulting architectures were optimized for single-thread performance with features such as branch prediction, out-of-order execution, and large primary instruction and data caches. In such architectures, most of the energy is consumed in overheads of data supply, instruction supply, and control.

7.6. Memory bandwidth and energy. The memory bandwidth bottleneck is a well-known computer program challenge that can limit application performance. Although GPUs provide more bandwidth than CPUs the scaling trends of off-chip bandwidth relative to on-chip computing capability are not promising. Power is another limiter for off-die memory. Energy efficiency can be achieving by eliminating as many instruction overheads as possible.

8. Acknowledgement

We thank for our guide to inspire us for this article and to guide us for this article, and the special thanks for entire NVIDIA team that brings GPU computing to market.

9. References

[1] NVIDIA, *NVIDIA CUDA Programming Guide, 2009; http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/*NVIDIA_CUDA_PROGRAMMING_Guide_2.3.pdf.

[2] M.Garland et al., "Parallel Computing Experiences with CUDA," *IEEE Micro*, vol. 28, no. 4, 2008, pp. 13-27.

[3] J.Nickolis et al., "Scalable Parallel Programming with CUDA," ACM Queue, vol. 6, no. 2, 2008, pp. 40-53.

[4] Tom R. Halfhill .PARALLEL PROCESSING WITH CUDA- NVidia's high-Performance Computing Platform Uses Massive Multithreading.

[5] www.wikipedia.org/wiki/CUDA

[6] Punit Kishore. *CUDA Programming Basics.*