PARALLEL COMPUTING USING CUDA

Author: Victor ARMAŞ
Scientific advisor: lect. sup. Dumitru CIORBĂ
Technical University of Moldova
Email: vict.armash@gmail.com, dumitru.ciorba@ati.utm.md

Abstract: *The direction of computing evolves from a "centralized processing" on the CPU to "co-processing" on the CPU and GPU. To implement the new paradigm of computing, NVIDIA has invented a parallel computing architecture CUDA, currently represented in the graphic processors GeForce, ION, Quadro, and Tesla, and provides the necessary framework for software developers.*

Index Terms: *CUDA, GPGPU, parallel computing.*

1. Introduction

The possibility to use the CPU graphics card (GPU) for non-graphics computations has been talked for a long time. For the first time CUDA (Compute Unified Device Architecture) architecture, appeared in February 2007, giving developers the ability to use technology for GPGPU (General-Purpose computing on Graphics Processing Units), through which in the familiar high-level languages (primarily - C) can be implemented algorithms that perform computations on the graphic accelerator GeForce of eighth generation and higher. Video card that supports CUDA is a powerful programmable architecture, like nowadays CPUs.

Until recently, the main component of computer has been considered the central processor. However, in some types of calculation, the theoretical computing power of graphics cards is several times higher than even the most powerful CPU. Previously, graphics processors (GPU) were used only for processing the output image to the screen. In full power they were working only in the games with advanced graphics, and the rest of their enormous computing power has been wasted for idle, even when the CPU was loaded at 100%. [1]

The breakthrough in this direction was the technology of CUDA, designed by NVIDIA. This development environment is much easier to use for writing software that uses algorithms applied to the video cards). Around the same time, the Open Source community has created a competitive package of OpenCL, which allows to work with both graphics cards from AMD, and those released by NVIDIA. The result was the emergence of various kinds of software that uses the processing power of graphics cards to accelerate the work. Typical examples of such software are Adobe Photoshop CS4, distributed computing clients, drivers and software for video encoding.

For developing applications using CUDA architecture can be used a wide range of languages and API, including C, C++, Fortran, OpenCL and DirectCompute. The CUDA architecture offers hundreds of cores that can handle parallel threads, while the CUDA programming model allows developers to focus on the implementation of parallel computing algorithms in them, rather than the structure of language. [2]

The CUDA architecture of the last generation, code-named «Fermi» - is the most advanced GPU computing architecture in history. Thanks to a three billion transistors Fermi architecture allows to make a joint processing on the GPU and CPU, providing a comprehensive and incredible performance in a range of computing applications. Support for C ++ allows to simplify parallel computing using GPU based on Fermi architecture and get a better performance in an even wider application list. Among the applications that provide significant performance gains are: ray tracing, finite element analysis, high-precision scientific computing, sparse matrices in linear algebra, sorting and searching algorithms.

2. CUDA architecture

A significant bonus for the use of GPU, is providing for the developers an unprecedented degree of parallelism: the ability to run a process in hundreds of thousands of threads!

It is fair to say that such threads are quite different from the threads of CPU, but CUDA is struggling to hide the complexity from the developer in order to use them. Despite the relative ease of implementation,

just to take and increase the productivity of any application by N times will not be possible. Thanks to CUDA, it can be optimized only that part of the application, which can be parallelized, and this unfortunately is not always an obvious task**.**

Let's see the differences between the main system processor (CPU) and graphics processor. It is important to understand that the CPU has been originally adapted for solving general order tasks and works with random addressable memory. Programs on the CPU can address directly to all cells in a linear and homogeneous memory. Comparing this with the GPU, that uses five different types of memory. But here, CUDA does everything to help the programmer, allowing the processes within the same block to work with shared memory.

The perennial problem for most computer systems is that the memory is slower than the processor. To neutralize this disadvantage, the producers use the CPU cache memory running at the processor frequency. Thus, it is possible to save time when accessing frequently used data. Modern graphics processors also have a system cache, but it is not as powerful as the CPU's one. Therefore, on the GPU the slow memory access is being hidden by using parallel computing. While some tasks are waiting for data, others work, which are ready for computation. This is one of the basic principles of CUDA, enabling greatly improve performance.

Computing architecture CUDA is based on the concept of Single Instruction Multiple Data (SIMD) and the concept of a multiprocessor [3]. SIMD concept implies that an instruction can simultaneously handle a set of data. Multiprocessor - is a multi-core SIMD-processor that allows you at any given point in time to perform on all cores only one instruction. The below diagram describes the scheme of interaction between CPU and GPU (Figure 1).

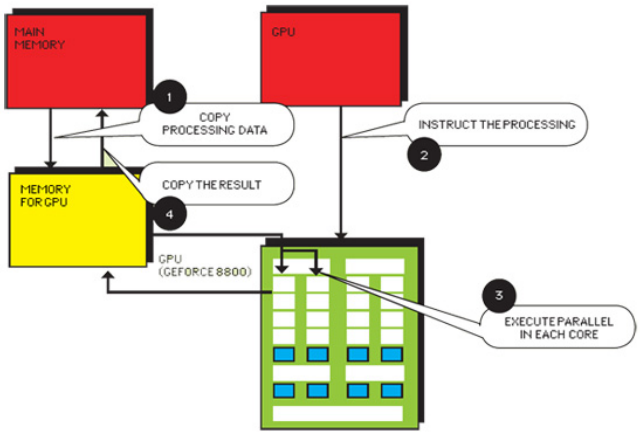

Figure 1 – The scheme of interaction between the CPU and GPU [4]

General steps of interaction between the CPU and GPU are described as follows:

1) The data from main memory is being copyed to the video card memory.
2) Transfering control to GPU.
3) GPU executes a command in parallel on each core.
4) The result from the memory of video card is being copyed in the main memory

It is important to understand that using the power of the GPU is not always appropriate. GPU is designed for computing with high parallelism and intensive arithmetic. This is explained by the fact that it has a much larger number of transistors designed for data processing rather than for flow control. So GPU shows good results in the parallel processing of data when using the same workflow for processing a large set of data. For this reason, the power of using CUDA can felt when performing the same actions over huge amounts of data (example: a local brute force of something) [5].

3. CUDA terminology

NVIDIA operates with very peculiar definitions for the CUDA API. They differ from the definitions used for the CPU. The general terminology leads to the following terms:

- ***Thread*.** Is a set of data to be processed (does not require large resources in processing).
- ***Warp***. Represents a group of 32 threads. The data is processed only by warps, so warp - is the minimum amount of data.
- ***Block***. Is the set of threads (from 64 to 512) or a set of warps (from 2 to 16).

- ***Grid***. Represents a set of blocks. This separation of data is used solely to improve performance. Thus, if the number of multiprocessors is large, the blocks will be executed in parallel. If the card is not good enough, the data block is processed sequentially. Developers recommend for complex calculations using the adapter not lower than the GeForce 8800 GTS 320 MB.

NVIDIA also introduces concepts such as kernel, host and device (Figure 2).

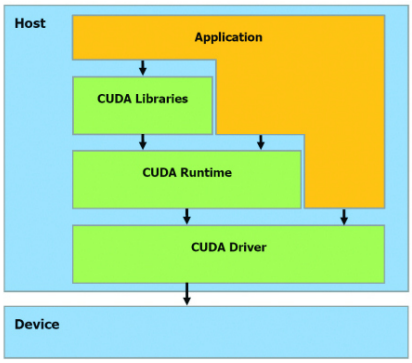

Figure 2 – CUDA integration scheme

The possibilities of the CUDA technology are:

- Standard C language for parallel application development on the GPU
- Standard libraries of numerical analysis for the fast Fourier transform and basic linear algebra software package
- Special CUDA driver for computing with fast data transfer between the GPU and CPU
- CUDA driver communicates with the graphics drivers OpenGL and DirectX
- Operating System Support Linux 32/64-bit, Windows XP/Vista/7 32/64-bit, and Mac [6].

4. Practical Results

To find out how big speed increase provides CUDA, were taken measurements of time of encoding video with technology being enabled, and without it [1].

Using the Super LoiLoScope Mars program, HD videos with a resolution of 1920x1080 pixels, encoded with H.264 and VC1 codecs were encoded in MP4 format. A test system has been assembled, using a motherboard with chipset X58, a processor Core i7 920, three gigabyte memory modules DDR3 (1333 MHz) and video card NVIDIA GeForce GTX285. As a result, when enabled, with NVIDIA CUDA technology the machine handled the encoding of the video clips on average five times faster, which gives a huge payoff in time, when it comes to large amounts of video.

Time of encoding, in seconds, of a video clip from MKS format (1920 × 1080) with the size of 32 Mbytes into MP4 (Figure 3)

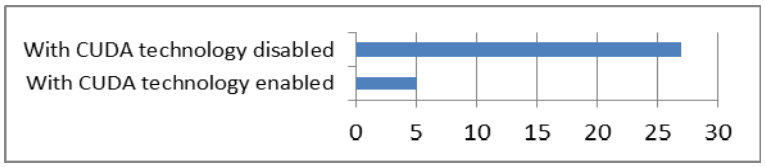

Figure 3 – Encoding a video clip from MKS to MP4

Time of encoding, in seconds, of a video clip from MOV format (1920 × 1080) with the size of 150 Mbytes into MP4 (Figure 4)

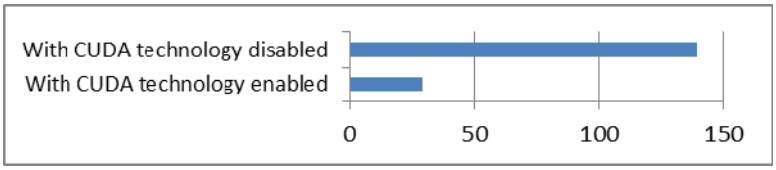

Figure 4 – Encoding a video clip from MOV to MP4

Time of encoding, in seconds, of a video clip from FLV format (1920 × 1080) with the size of 226 Mbytes into MP4 (Figure 5)

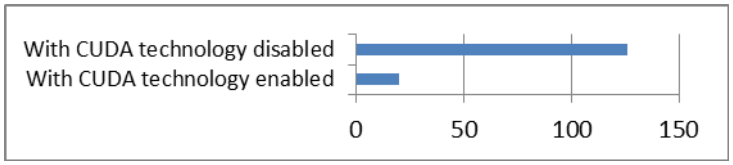

Figure 5 – Encoding a video clip from FLV to MP4

5. Conclusion

In the past five years, the growth of the clock speed of CPUs has slowed considerably, being more and more difficult to overcome the current level, and demanding big costs for developing and manufacturing thin transistors.

At the same time the demand of improving the performance of computer technology is growing each the day. The only possibility to satisfy this need nowadays is using parallel computing, which is why in today's processors instead of increasing the frequency, the manufacturer increases the number of cores, which gives a significant performance boost. The architecture of graphical processors has been originally designed for calculations with multi-threaded tasks, but modern GPU have in their disposition tens and even hundreds of independent computational blocks capable of producing calculations like CPU cores, that offers advantages for video card over traditional CPUs.

To take advantage of multi-core computing devices the software must be specially designed and optimized for parallel execution. CUDA technology makes it possible to effectively parallelize the problem by assuming itself most of the work and distributing the load between computational blocks of GPU. Thus, making it easier to develop applications rather than writing programs for multi-core CPU.

GPU is not able to completely replace CPU, because it is not adapted to carry out all types of calculations, which traditional processors are capable of, but even today graphics cards allow us to significantly increase the processing time of many tasks.

References

1. **Перекалин, Александр.** CUDA БЫСТРЕЕ. *Chip.* 07 2009, p. 148.

2. CUDA Parallel programming made easy. *Nvidia.* [Online] [Cited: december 12, 2011.] http://www.nvidia.com/object/cuda_home_new.html.

3. Flynn's taxonomy. *Wikipedia.* [Online] [Cited: 12 12, 2011.] http://en.wikipedia.org/wiki/Flynn's_taxonomy.

4. **Поликарпов, Глеб.** CUDA КАТИТСЯ МИР. *Хакер.* 07 2009, p. 127.

5. **FastVideo.** Производительность параллельных вычислений GPGPU. *FastVideo.* [Online] [Cited: december 12, 2011.] http://www.fastvideo.ru/info/cuda/cuda-benchmarks.htm.

6. **Хмельницкий Национальный Университет.** Параллельные вычисления. *Центр Параллельных Вычислений.* [Online] [Cited: december 12, 2012.] http://parallelcompute.sourceforge.net/parcom_ru.php.