# COMPUTE UNIFIED DEVICE ARCHITECTURE: POWERFUL PARALLEL SOLUTION FOR GENERAL PURPOSE COMPUTING

Miloš OČKAY*, Marcel HARAKAĽ*, Miroslav LÍŠKA*
*Department of Informatics, Armed Forces Academy of General M. R. Štefánik
Demänová 393, 031 01 Liptovský Mikuláš, Slovak Republic,
E-mail: milos.ockay@aoslm.sk, marcel.harakal@aoslm.sk, miroslav.liska@aoslm.sk,

**ABSTRACT**

*CUDA (Compute Unified Device Architecture) is a successful and promising implementation of unified architecture. CUDA simplified the development of parallel, general purpose applications on graphics accelerators and expanded possibilities of parallel processing. We are presenting a brief description of fundamental elements of this architecture and the possibility of its use in parallel environment.*

***Keywords:*** *Compute Unified Device Architecture, CUDA, Graphics Processing Unit, GPU, Parallel computing, Stream processor.*

## 1. INTRODUCTION

GPU (Graphics Processing Unit) is a parallel computing tool for general purpose computing. A programmer does not need to use graphics APIs. Graphics APIs make general purpose computing possible on GPUs, but it is a difficult way full of constraint rules. CUDA is a new architecture for general purpose computing that can achieve results with reasonable amount of effort [1]. CUDA has been developed by nVidia to support general purpose computing on the graphics hardware. CUDA can be used on graphics accelerators which include the G80, G92 and other future chips based on the unified architecture. Some of GeForce and Quadro products belong to this category. nVidia Tesla products have been developed especially for the general purpose computing [10]. This new approach has significantly simplified the development of general purpose applications on GPUs. Price of this solution makes graphics accelerators available on desktop PCs and we can assume a wide spread adaptation of CUDA architecture among the developers.

## 2. G80 ARCHITECTURE

G80 is a chip developed by nVidia and it is an implementation of the unified architecture. Unified architecture is an architecture standardized by DirectX 10 [6]. Unified architecture has integrated processing of the elements in computer graphics. In addition, it brings possibility to use graphics hardware for the wide variety of applications, not just for computer graphics. The chip can be divided in to two main sections. The first, the main part, is the core of the chip and the second part is called Lumex Engine. Lumex Engine is responsible for texture filtering, antialising, HDR and other processing.

Chip G80 is trying to fulfil the concept of unified architecture as close as possible [4]. The core of the chip (Fig. 1), consists of 128 independent stream processors (SP). These stream processors run at an astounding frequency 1.35GHz. Stream processors are grouped in 8 blocks and each block consists of 16 processors. Each block can use its own four texture addressing units (TA), eight filtering units (TF) and shared cache memory (L1). Each main block includes two shader processors (one shader processor = eight stream processors). Each block has an access to any shared cache memory (L2) and to the six general purpose registers (GPR) [5]. Shared memory model is useful for parallel processing of data with shader processors. Different versions of graphics accelerators have different number of stream processors. The number of processors described here belongs to the Tesla category. Stream processors are independent and they can be turned off (graphics accelerator will stay functional) in case of malfunction or manufacturing of the cheaper model.

Data are converted to the 32 bit floating point format in input assembler. Thread processor (TP) sorts processing threads and also optimizing the payload.

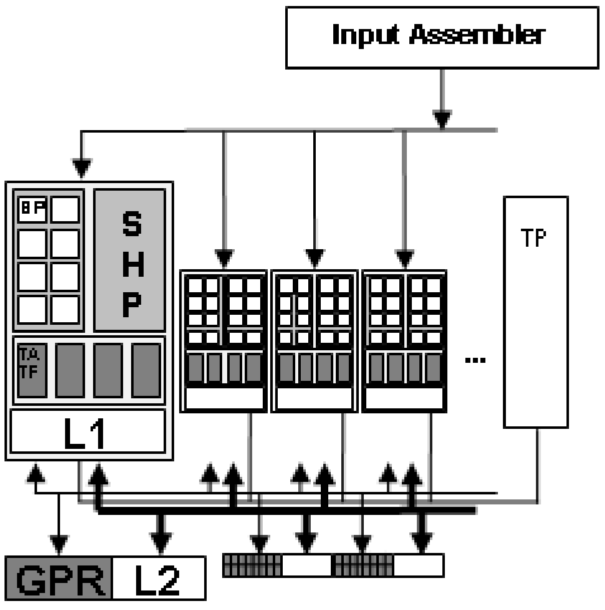

**Fig. 1** G80 Architecture

Each stream processor can process two scalar operations within every clock cycle. The overall performance is around 520 gigaflops according to nVidia. G80 is a fully scalar processor. Each stream processor is a classical ALU (Arithmetic logic unit) capable of processing floating point data. In fact, it means that processor is suitable for general purpose calculations. Processing of one processor is independent of the others. One part of G80 may process results and the other can take care of visualization. The output of one processor can be the input of the other.

Memory subsystem has a new management circuit. Memory bandwidth is 348 bits. Memory is implemented as the GDDR3 modules.

New graphics chip produced by nVidia introduced a lot of innovations which significantly raised the chip's complexity. The chip consists of the 681 million transistors, in contrast to modern desktop CPU (154 million transistors in AMD Athlon 64 X2 or 589 million in Intel Kentsfield).

## 3. CUDA ARCHITECTURE

CUDA can be integrated in to the existing IT environments. CUDA comprises a software development kit and a C compiler. The C compiler is integrated in to the well known environment. This solution is compatible with x86 and x64 microprocessors produced by AMD/INTEL under Linux and Windows operating systems. Graphics accelerator (DEVICE) is interconnected with a host system (host) over PCI EXPRES x16. The speed of this interconnection is 8 GB per second (4GB/s upstream and 4GB/s downstream).

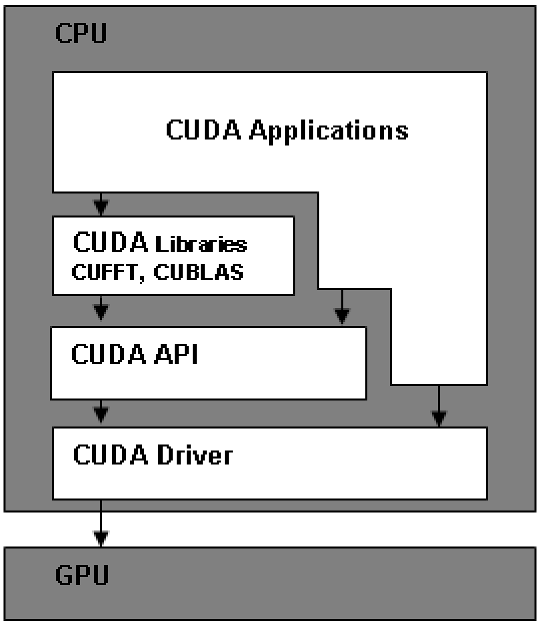

**Fig. 2** Layered model of CUDA

CUDA expands usage possibilities of general purpose computing on GPU. GPU is, in this case, a parallel coprocessor. GPU processes large sets of parallel data. CPU is taking care of management, presentation and organization of computational process. Complex datasets are divided into smaller independent parts and they are processed with the same instructions. In other words, data can be processed simultaneously. CUDA consist a few basic components. These components are taking care of parallel processing of data. These components are: CUDA driver, CUDA API and CUDA mathematical libraries (CUBLAS and CUFFT) (Fig. 2). C compiler significantly simplifies the process of development of parallel applications. A developer can be fully focused on the development of an application. He does not need to adapt a problem to be realized on graphics API. Developer can build CPU and GPU code as one consistent entity. Expansion directives show to the compiler which part will be processed on the GPU and witch part on the CPU. First, code is compiled with CUDA compiler for GPU and than with standard C compiler. This programming model allows us to use the code on different versions of graphics accelerator. Programmer does not need to know how many stream processors are engaged. Realization of the GPU program looks like this. Program is executed on the host system. CUDA driver automatically executes code on GPU. The CPU part of program communicates with the GPU part through high-speed interface. Communication between CPU and GPU is realized with special operations in a driver. The management of computational resources is not in care of a programmer. CUDA combined with massively parallel GPU brings the acceleration to applications which need parallel computational power.

### 3.1. Composition of parallel computation with CUDA

Thread is a basic element of parallel processing of data. Overall dataset is divided to threads. Threads are grouped in specific way and the parallel program (kernel) is applied to the groups and process results. We can obtain an efficient acceleration with thousands of threads in process. Block is a group of threads which can effectively cooperate and share data through the shared cache memory. Threads in the same block are processed with one kernel. Each thread in the block has its own ID. In other words, ID specifies location of thread in the block. In case of thread location the block is considered as 2D or 3D array. (x+Dx) is ID of the thread which is on (x,y) position in 2D array. (x+yDx+zDxDy) is ID of the thread which is on (x,y,z) position in 3D array. The count of threads in block is limited, but it is possible to group blocks which have the same dimensions and are processed by the same kernel. This group of blocks is called grid. Thanks to grid we are not limited with the block capacity. Each block in the grid has its own ID. The indexation of blocks within the grid is alike to thread indexation in block. In other words, independent data are divided to threads. Threads are grouped to blocks and if we reach the block maximum capacity, we can use multiple blocks and group them to the grid. All threads in grid are processed with the same kernel in SIMD (Single Instruction Multiple Data) parallel fashion.

### 3.2. Memory model of CUDA architecture

CUDA architecture uses shared memory model. Expansion directives allow memory transfers between the

HOST and the DEVICE. Local shared memory and registers help to keep memory operations on the chip. The memory of the graphics accelerator is divided to different parts. The local memory and registers are used for the local thread operations. The shared memory is used for communication among the threads. The global memory is used for the communication among the blocks and grids.

Threads have an access only to the DRAM memory of DEVICE through the following mechanisms [2]:

- Read-write per-thread registers,
- Read-write per-thread local memory,
- Read-write per-block shared memory,
- Read-write per-grid global memory,
- Read-only per-grid constant memory,
- Read-only per-grid texture memory.

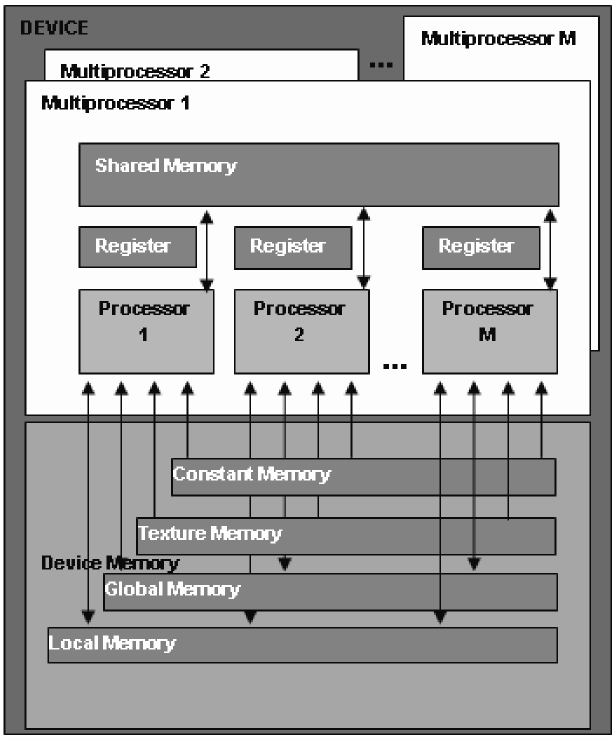

**Fig. 3** Hardware Implementation of CUDA architecture

Thread does not have an access to the HOST memory. If we need to access this part of the memory, data are copied to the global memory of DEVICE. The opposite transfer is also possible. CUDA offers the common memory model. Read and write memory operations on DEVICE are alike on the HOST. The shared memory model brings acceleration of memory operations and it makes thread communication more efficient. According to that, the application becomes less dependent on DRAM bandwidth.

## 3.3. Hardware implementation of CUDA architecture

DEVICE is implemented as the group of multiprocessors (Fig. 3). Each multiprocessor is SIMD (Single Instruction Multi Data) architecture. The multiprocessor processes the same instruction every clock cycle, but on the different data.

Each processor has few different kinds of memory spaces:

- Local 32bit registers for each processor.
- Shared parallel cache shared by all processors. It implements shared memory space.
- Constant memory shared by all processors. It implements read only memory space.
- Texture cache shared by all processors. It implements read only memory space.

## 3.4. Execution model of CUDA architecture

Grid is processed by the DEVICE. Each multiprocessor processes batches of blocks one batch after another. A block is processed by only one multiprocessor. Memory operations stay local and that fact accelerates whole process. The number of blocks which multiprocessor can process depends on the number of registers and shared memory required. If there is not enough memory, kernel will fail to launch. The blocks that are processed in one batch by one multiprocessor are called active. Each active block is split to two SIMD warps. Each warp contains the same number of threads. The order of warps is not set, but their execution can be synchronized. The order of blocks within a grid is undefined. There is no synchronization between the blocks. Threads from different blocks of the same grid cannot safely communicate with each other [3] [2].

## 4. GRAPHICS ACCELERATOR INTEGRATION TO THE PARALLEL ENVIRONMENT

Massively parallel power of graphics accelerators is not their one and only advantage. For example, integration to the parallel environment like computer cluster is also possible. Integration of these accelerators adds another level of parallelism to the cluster.

CUDA programming model simplifies the process of management and synchronization in parallel computation. Each cluster nod would be a HOST for graphics accelerator. Overall computation can be divided in to several parallel levels. We can possibly face several problems such as the platform unification for all included hardware, finding the proper algorithms for computation on this hardware platform and others. The bottleneck of this acceleration is memory operations (between GPU and CPU) latency. During the process of searching for proper algorithms we need to consider the time of realization and communication that the computation consumes and then decide which part will be realized on GPU instead of CPU (Fig. 4). In this environment we have to consider multi-paradigm approach [8]. A multi-paradigm language provides an opportunity to a user for exploiting more programming methodologies. It simplifies the language syntax, and extends the application areas by the extended semantics. That is why multi-paradigm languages can align a problem in wider application areas and more flexibly than that based on a single paradigm [9].

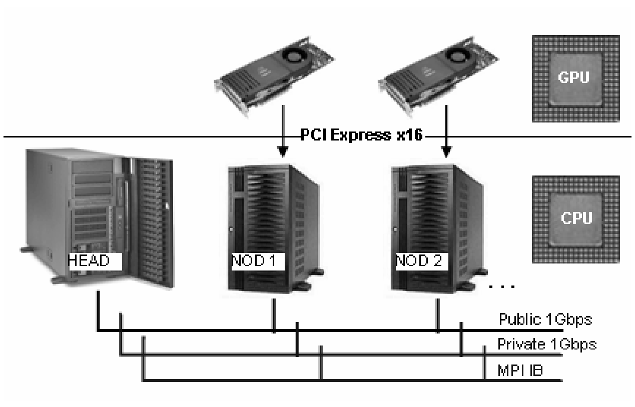

**Fig. 4** Possible integration of graphics accelerators to the cluster environment

## 5. CONCLUSIONS

Unified architecture brought significant innovations to the High Performance Computing world. Graphics accelerators are not just graphics processing tools anymore. G80 and CUDA are one of the promising implementations of unified architecture. CUDA simplified process of development of general purpose parallel applications. These applications have now enough computational power to get proper results in a short time. In the future we can expect progress in the field of standardization in this area and growth of power of graphics accelerators. In addition, we can expect progress in applications that use graphics accelerators as a computational tool [7].

## REFERENCES

[1] General-purpose computation using graphics hardware homepage, http://www.gpgpu.org

[2] CUDA Programming Guide. ver. 1.1, http://www.nvidia.com/object/cuda_develop.html

[3] TESLA GPU Computing Technical Brief, http://www.nvidia.com/object/tesla_product_literature.html

[4] G80 architecture reviews and specification, http://www.nvidia.com/page/8800_reviews.html, http://www.nvidia.com/page/8800_tech_specs.html

[5] Beyond3D G80: Architecture and GPU Analysis, http://www.beyond3d.com/content/reviews/1

[6] Microsoft DirectX resource center, http://msdn2.microsoft.com/en-us/xna/aa937781.aspx

[7] Owens D.J., Luebke D., Govindaraju N., Harris M., Krüger J., Lefohn E.A., Purcell T.: A Survey of General-Purpose Computation on Graphics Hardware. In: The Eurographics, (2005).

[8] Kollár Ján, Porubän Jaroslav, Václavík Peter, Tóth Marcel, Bandáková Jana, Forgáč Michal: Multi-paradigm Approaches to Systems Evolution, Computer Science and Technology Research Survey, Košice, Elfa s.r.o., (2007), 1, pp. 6-10, 978-80-8086-046-2

[9] Kollár Ján, Porubän Jaroslav, Václavík Peter: Separating Concerns in Programming: Data, Control and Actions, In: Computing and Informatics, 24, 5, (2005), pp. 441-462, ISSN 1335-9150

[10] Graphics adapters supporting CUDA, http://www.nvidia.com/object/cuda_learn_products.html

Received May 12, 2009, accepted August 18, 2009

## BIOGRAPHIES

**Miloš Očkay** was born on 28.9.1978. In 2003 he graduated (MSc) at Military Academy in Liptovský Mikuláš as civil student. Since 2006 he is working as a tutor with the department of Informatics at the Armed Forces Academy in Liptovský Mikuláš. He is currently (2009) PhD candidate at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. His scientific research is focusing on parallel computing and graphics adapters general purpose programming methods.

**Marcel Harakaľ** is associate professor and head of the department of Informatics at the Armed Forces Academy in Liptovský Mikuláš. He received the MSc degree in electrical engineering from the Faculty of Electrical Engineering at Slovak Technical University Bratislava in 1983. He defended PhD degree in the field of artificial intelligence at the Military Academy in Liptovský Mikuláš in 1997. From 1983 to 1989, he was a research engineer at the Military Research Institute in Liptovský Mikuláš. Since 1989 he has been working in various computer science teaching positions at the department of Informatics at the Armed Forces Academy (former Military Academy) in Liptovský Mikuláš.

**Miroslav Líška** is professor of Military Communication and Information Systems at the department of Informatics at the Armed Forces Academy in Liptovský Mikuláš. He is former rector (chancellor) of the Military Academy in Liptovský Mikuláš (1999-2002) and former head of the department of Informatics (1993-2002).