Book Review

Computación de Alto Desempeño en GPU (High Performance Computing in GPU)

María Fabiana Piccoli

EDULP, 2011
ISBN 978-950-34-0759-2

GPU is a dedicated graphic card for personal computers, workstations or video game consoles. It is an interesting architecture to high performance computing. GPU was developed with a highly parallel structure, high memory bandwidth and more chip surface dedicated to data processing than to data caching and flow control. It offers high speedup to any standard graphic application.

Mapping general-purpose computation onto GPU implies to use the graphics hardware to solve any application not necessarily of graphic nature. This is called GPGPU (General-Purpose GPU), GPU computational power is used to solve general-purpose problems. The parallel programming over GPUs has many differences from parallel programming in typical parallel computer, the most relevant are:

- The number of processing units: In massively parallel computer, generally the number is chosen to be the same as the number of cores in the system. In contrast, GPGPU allows the execution of hundreds or thousands processing units on a large number of processing elements.
- CPU-GPU memory structure: The CPU-GPU memory system considers two different memory spaces: the host memory (it is shared by every thread in CPU during the application) and GPU memory. Instead, GPU threads run in a separate memory space from the host threads in the application. In consequence, the code and data transfers are necessary between host and device in different moments. These transfers have to be done in controlled form, because they can affect directly the whole application performance.
- Number of parallel threads: GPUs programming has the ability to start a large number of threads with little overhead. GPU offers transparent mechanisms for creating and managing of threads with low cost.

These characteristics imply that GPU programming involves more overhead than CPU programming. Every GPGPU program has many basic steps, first the input data transfers to the graphics card. Once the data are in place on the card, many threads can be started (with little overhead). Each thread works over its data and, at the end of the computation, the results should be copied back to the host main memory.

Not all kind of problem can be solved in the GPU architecture, the most suitable problems are those that can be implemented with stream processing and using limited memory, i.e. applications with abundant data parallelism.

Through this textbook (written in Spanish), the author introduces the GPU as a parallel computer that is able to solve general-propose problems. The book has a didactic approach, it includes the first steps to CUDA programming, some tips to take account to develop good applications performance, many examples and exercises.

The book is organized in five chapters and three appendixes. Each chapter has an introduction, summary, and practical exercises. The topics are incrementally developed in incremental form, starting by the simplest cases. The next paragraphs summary the topics treated in each chapter.

Chapter 1 – High Performance Computing – This chapter introduces the basic concepts of high performance computing, such as parallel systems: Parallel hardware and software, main paradigms: Data, Task or Control and Nested parallelisms, Shared Memory and Messages Passing (Distributed Memory).

Chapter 2 – GPGPU: Introduction. In this chapter, the GPU is introduced. It includes: a comparison with CPU, their design philosophies and motivations of their evolution, describing the architectures of first GPU until Modern GPUs with unified graphics and computing processors (fixed processors vs. unified processors). Finally, GPGPU (General-Propose computing on GPU) is introduced, showing the general-propose programming characteristics since the first GPUs to current ones.

Chapter 3 – GPU programming: CUDA Programming Model. The Compute Unified Device Architecture (CUDA), supported from the NVIDIA Geforce 8 Series, enables to use GPU as a highly parallel computer for non-graphics applications. CUDA provides an essential high-level development environment with standard C/C++ language. It defines the GPU architecture as a programmable graphic unit that acts as a coprocessor for CPU. It has multiple streaming multiprocessors (SMs), each of them contains several (eight, thirty-two or forty-eight, depending GPU architecture) scalar processors (SPs). The first steps of programming are detailed in this chapter, its generalities, data transfers types, organization of CUDA computing: threads, blocks and grids, synchronization mechanisms and CUDA execution model: SIMT. Finally an example of CUDA programming is shown.

Chapter 4 – CUDA memory hierarchy. CUDA threads may access data from multiple memory spaces during their execution. The CUDA Memory Hierarchy is formed by Global Memory, Local Memory, Register, Shared Memory, Constant Memory and Texture memory. Each kind of memory has its own access cost, scope and lifetime. The proprieties of each memory are explained and two examples of their use are included at the chapter end. Furthermore, it is made an analysis of memory role as limiting factor to achieve high parallelism levels.

Chapter 5 - Performance analysis and optimizations. When a programmer develops a CUDA program, he/she makes decisions that can affect the application performance and can constrain parallel level of execution on the GPU. In order to achieve the best performance of a CUDA application, programmers have to take account some issues related with threads execution, global memory: access organization and data pre-fetching; instruction performance: instruction mix and thread granularity; and SM resources allocation. This chapter explains the above issues, and includes a guide for take advantages of them and achieve the best application performance.

Finally, three appendixes are included. They refer to basic aspect to CUDA programming such as basic extensions of C language (built-in types and variables, qualifiers of functions and variable, synchronization functions, memory management functions and atomic functions), additional functions for CUDA programs (to compute time or report some errors), and GPU models and their CUDA computing capabilities.

María F. Piccoli
mfpiccoli@gmail.com