8.1 Introduction

(Original section[1])

COD Appendix B (Graphics and Computing GPUs) authored by *John Nickolls*, Director of Architecture, NVIDIA, and *David Kirk*, Chief Scientist, NVIDIA.

> "Imagination is more important than knowledge.
> *Albert Einstein, On Science, 1930s*

This appendix focuses on the *GPU*—the ubiquitous *graphics processing unit* in every PC, laptop, desktop computer, and workstation. In its most basic form, the GPU generates 2D and 3D graphics, images, and video that enable Window-based operating systems, graphical user interfaces, video games, visual imaging applications, and video. The modern GPU that we describe here is a highly parallel, highly multithreaded multiprocessor optimized for *visual computing*. To provide real-time visual interaction with computed objects via graphics, images, and video, the GPU has a unified graphics and computing architecture that serves as both a programmable graphics processor and a scalable parallel computing platform. PCs and game consoles combine a GPU with a CPU to form *heterogeneous systems*.

Graphics processing unit (***GPU***): A processor optimized for 2D and 3D graphics, video, visual computing, and display.

Visual computing: A mix of graphics processing and computing that lets you visually interact with computed objects via graphics, images, and video.

Heterogeneous system: A system combining different processor types. A PC is a heterogeneous CPU-GPU system.

A brief history of GPU evolution

Fifteen years ago, there was no such thing as a GPU. Graphics on a PC were performed by a *video graphics array* (VGA) controller. A VGA controller was simply a memory controller and display generator connected to some DRAM. In the 1990s, semiconductor technology advanced sufficiently that more functions could be added to the VGA controller. By 1997, VGA controllers were beginning to incorporate some *three-dimensional* (3D) acceleration functions, including hardware for triangle setup and rasterization (dicing triangles into individual pixels) and texture mapping and shading (applying "decals" or patterns to pixels and blending colors).

In 2000, the single chip graphics processor incorporated almost every detail of the traditional high-end workstation graphics pipeline and, therefore, deserved a new name beyond VGA controller. The term GPU was coined to denote that the graphics device had become a processor.

Over time, GPUs became more programmable, as programmable processors replaced fixed-function dedicated logic while maintaining the basic 3D graphics pipeline organization. In addition, computations became more precise over time, progressing from indexed arithmetic, to integer and fixed point, to single-precision floating-point, and recently to double-precision floating-point. GPUs have become massively parallel programmable processors with hundreds of cores and thousands of threads.

Recently, processor instructions and memory hardware were added to support general purpose programming languages, and a programming environment was created to allow GPUs to be programmed using familiar languages, including C and C++. This innovation makes a GPU a fully general-purpose, programmable, manycore processor, albeit still with some special benefits and limitations.

GPU graphics trends

GPUs and their associated drivers implement the OpenGL and DirectX models of graphics processing. OpenGL is an open standard for 3D graphics programming available for most computers. DirectX is a series of Microsoft multimedia programming interfaces, including Direct3D for 3D graphics. Since these *application programming interfaces (APIs)* have well-defined behavior, it is possible to build effective hardware acceleration of the graphics processing functions defined by the APIs. This is one of the reasons (in addition to increasing device density) why new GPUs are being developed every 12 to 18 months that double the performance of the previous generation on existing applications.

Application programming interface (***API***): A set of function and data structure definitions providing an interface to a library of functions.

Frequent doubling of GPU performance enables new applications that were not previously possible. The intersection of graphics processing and parallel computing invites a new paradigm for graphics, known as visual computing. It replaces large sections of the traditional sequential hardware graphics pipeline model with programmable elements for geometry, vertex, and pixel programs. Visual computing in a modern GPU combines graphics processing and parallel computing in novel ways that permit new graphics algorithms to be implemented, and opens the door to entirely new parallel processing applications on pervasive high-performance GPUs.

Heterogeneous system

Although the GPU is arguably the most parallel and most powerful processor in a typical PC, it is certainly not the only processor. The CPU, now multicore and soon to be manycore, is a complementary, primarily serial processor companion to the massively parallel manycore GPU. Together, these two types of processors comprise a heterogeneous multiprocessor system.

The best performance for many applications comes from using both the CPU and the GPU. This appendix will help you understand how and when to best split the work between these two increasingly parallel processors.

GPU evolves into scalable parallel processor

GPUs have evolved functionally from hardwired, limited capability VGA controllers to programmable parallel processors. This evolution has proceeded by changing the logical (API-based) graphics pipeline to incorporate programmable elements and also by making the underlying hardware pipeline stages less specialized and more programmable. Eventually, it made sense to merge disparate programmable pipeline elements into one unified array of many programmable processors.

In the GeForce 8-series generation of GPUs, the geometry, vertex, and pixel processing all run on the same type of processor. This unification allows for dramatic scalability. More programmable processor cores increase the total system throughput. Unifying the processors also delivers very effective load balancing, since any processing function can use the whole processor array. At the other end of the spectrum, a processor array can now be built with very few processors, since all of the functions can be run on the same processors.

Why CUDA and GPU computing?

This uniform and scalable array of processors invites a new model of programming for the GPU. The large amount of floating-point processing power in the GPU processor array is very attractive for solving nongraphics problems. Given the large degree of parallelism and the range of scalability of the processor array for graphics applications, the programming model for more general computing must express the massive parallelism directly, but allow for scalable execution.

GPU computing is the term coined for using the GPU for computing via a parallel programming language and API, without using the traditional graphics API and graphics pipeline model. This is in contrast to the earlier *General Purpose computation on GPU (GPGPU)* approach, which involves programming the GPU using a graphics API and graphics pipeline to perform nongraphics tasks.

GPU computing: Using a GPU for computing via a parallel programming language and API.

GPGPU: Using a GPU for general-purpose computation via a traditional graphics API and graphics pipeline.

Compute Unified Device Architecture (CUDA) is a scalable parallel programming model and software platform for the GPU and other parallel processors that allows the programmer to bypass the graphics API and graphics interfaces of the GPU and simply program in C or C++. The CUDA programming model has an SPMD (single-program multiple data) software style, in which a programmer writes a program for one thread that is instanced and executed by many threads in parallel on the multiple processors of the GPU. In fact, CUDA also provides a facility for programming multiple CPU cores as well, so CUDA is an environment for writing parallel programs for the entire heterogeneous computer system.

CUDA: A scalable parallel programming model and language based on C/C++. It is a parallel programming platform for GPUs and multicore CPUs.

GPU unifies graphics and computing

With the addition of CUDA and GPU computing to the capabilities of the GPU, it is now possible to use the GPU as both a graphics processor and a computing processor at the same time, and to combine these uses in visual computing applications. The underlying processor architecture of the GPU is exposed in two ways: first, as implementing the programmable graphics APIs, and second, as a massively parallel processor array programmable in C/C++ with CUDA.

Although the underlying processors of the GPU are unified, it is not necessary that all of the SPMD thread programs are the same. The GPU can run graphics shader programs for the graphics aspect of the GPU, processing geometry, vertices, and pixels, and also run thread programs in CUDA.

The GPU is truly a versatile multiprocessor architecture, supporting a variety of processing tasks. GPUs are excellent at graphics and visual computing as they were specifically designed for these applications. GPUs are also excellent at many general-purpose throughput applications that are "first cousins" of graphics, in that they perform a lot of parallel work, as well as having a lot of regular problem structure. In general, they are a good match to data-parallel problems (see COD Chapter 6 (Parallel Processor from Client to Cloud)), particularly large problems, but less so for less regular, smaller problems.

GPU visual computing applications

Visual computing includes the traditional types of graphics applications plus many new applications. The original purview of a GPU was "anything with pixels," but it now includes many problems without pixels but with regular computation and/or data structure. GPUs are effective at 2D and 3D graphics, since that is the purpose for which they are designed. Failure to deliver this application performance would be fatal. 2D and 3D graphics use the GPU in its "graphics mode," accessing the processing power of the GPU through the graphics APIs, OpenGL™, and DirectX™. Games are built on the 3D graphics processing capability.

Beyond 2D and 3D graphics, image processing and video are important applications for GPUs. These can be implemented using the graphics APIs or as computational programs, using CUDA to program the GPU in computing mode. Using CUDA, image processing is simply another data-parallel array program. To the extent that the data access is regular and there is good locality, the program will be efficient. In practice, image processing is a very good application for GPUs. Video processing, especially encode and decode (compression and decompression according to some standard algorithms), is quite efficient.

The greatest opportunity for visual computing applications on GPUs is to "break the graphics pipeline." Early GPUs implemented only specific graphics APIs, albeit at very high performance. This was wonderful if the API supported the operations that you wanted to do. If not, the GPU could not accelerate your task, because early GPU functionality was immutable. Now, with the advent of GPU computing and CUDA, these GPUs can be programmed to implement a different virtual pipeline by simply writing a CUDA program to describe the computation and data flow that is desired. So, all applications are now possible, which will stimulate new visual computing approaches.

(*1) This section is in original form.