# Chapter 1

# Why CUDA? Why Now?

There was a time in the not-so-distant past when parallel computing was looked upon as an "exotic" pursuit and typically got compartmentalized as a specialty within the field of computer science. This perception has changed in profound ways in recent years. The computing world has shifted to the point where, far from being an esoteric pursuit, nearly every aspiring programmer *needs* training in parallel programming to be fully effective in computer science. Perhaps you've picked this book up unconvinced about the importance of parallel programming in the computing world today and the increasingly large role it will play in the years to come. This introductory chapter will examine recent trends in the hardware that does the heavy lifting for the software that we as programmers write. In doing so, we hope to convince you that the parallel computing revolution has *already* happened and that, by learning CUDA C, you'll be well positioned to write high-performance applications for heterogeneous platforms that contain both central and graphics processing units.

## 1.1 Chapter Objectives

Through the course of this chapter, you will accomplish the following:

- You will learn about the increasingly important role of parallel computing.
- You will learn a brief history of GPU computing and CUDA.
- You will learn about some successful applications that use CUDA C.

## 1.2 The Age of Parallel Processing

In recent years, much has been made of the computing industry's widespread shift to parallel computing. Nearly all consumer computers in the year 2010 will ship with multicore central processors. From the introduction of dual-core, low-end netbook machines to 8- and 16-core workstation computers, no longer will parallel computing be relegated to exotic supercomputers or mainframes. Moreover, electronic devices such as mobile phones and portable music players have begun to incorporate parallel computing capabilities in an effort to provide functionality well beyond those of their predecessors.

More and more, software developers will need to cope with a variety of parallel computing platforms and technologies in order to provide novel and rich experiences for an increasingly sophisticated base of users. Command prompts are out; multithreaded graphical interfaces are in. Cellular phones that only make calls are out; phones that can simultaneously play music, browse the Web, and provide GPS services are in.

### 1.2.1 CENTRAL PROCESSING UNITS

For 30 years, one of the important methods for the improving the performance of consumer computing devices has been to increase the speed at which the processor's clock operated. Starting with the first personal computers of the early 1980s, consumer central processing units (CPUs) ran with internal clocks operating around 1MHz. About 30 years later, most desktop processors have clock speeds between 1GHz and 4GHz, nearly 1,000 times faster than the clock on the original personal computer. Although increasing the CPU clock speed is certainly not the only method by which computing performance has been improved, it has always been a reliable source for improved performance.

In recent years, however, manufacturers have been forced to look for alternatives to this traditional source of increased computational power. Because of various fundamental limitations in the fabrication of integrated circuits, it is no longer feasible to rely on upward-spiraling processor clock speeds as a means for extracting additional power from existing architectures. Because of power and heat restrictions as well as a rapidly approaching physical limit to transistor size, researchers and manufacturers have begun to look elsewhere.

Outside the world of consumer computing, supercomputers have for decades extracted massive performance gains in similar ways. The performance of a processor used in a supercomputer has climbed astronomically, similar to the improvements in the personal computer CPU. However, in addition to dramatic improvements in the performance of a single processor, supercomputer manufacturers have also extracted massive leaps in performance by steadily increasing the *number* of processors. It is not uncommon for the fastest supercomputers to have tens or hundreds of thousands of processor cores working in tandem.

In the search for additional processing power for personal computers, the improvement in supercomputers raises a very good question: Rather than solely looking to increase the performance of a single processing core, why not put more than one in a personal computer? In this way, personal computers could continue to improve in performance without the need for continuing increases in processor clock speed.

In 2005, faced with an increasingly competitive marketplace and few alternatives, leading CPU manufacturers began offering processors with two computing cores instead of one. Over the following years, they followed this development with the release of three-, four-, six-, and eight-core central processor units. Sometimes referred to as the *multicore revolution*, this trend has marked a huge shift in the evolution of the consumer computing market.

Today, it is relatively challenging to purchase a desktop computer with a CPU containing but a single computing core. Even low-end, low-power central processors ship with two or more cores per die. Leading CPU manufacturers have already announced plans for 12- and 16-core CPUs, further confirming that parallel computing has arrived for good.

## 1.3 The Rise of GPU Computing

In comparison to the central processor's traditional data processing pipeline, performing general-purpose computations on a graphics processing unit (GPU) is a new concept. In fact, the GPU itself is relatively new compared to the computing field at large. However, the idea of computing on graphics processors is not as new as you might believe.

### 1.3.1 A BRIEF HISTORY OF GPUS

We have already looked at how central processors evolved in both clock speeds and core count. In the meantime, the state of graphics processing underwent a dramatic revolution. In the late 1980s and early 1990s, the growth in popularity of graphically driven operating systems such as Microsoft Windows helped create a market for a new type of processor. In the early 1990s, users began purchasing 2D display accelerators for their personal computers. These display accelerators offered hardware-assisted bitmap operations to assist in the display and usability of graphical operating systems.

Around the same time, in the world of professional computing, a company by the name of Silicon Graphics spent the 1980s popularizing the use of three-dimensional graphics in a variety of markets, including government and defense applications and scientific and technical visualization, as well as providing the tools to create stunning cinematic effects. In 1992, Silicon Graphics opened the programming interface to its hardware by releasing the OpenGL library. Silicon Graphics intended OpenGL to be used as a standardized, platform-independent method for writing 3D graphics applications. As with parallel processing and CPUs, it would only be a matter of time before the technologies found their way into consumer applications.

By the mid-1990s, the demand for consumer applications employing 3D graphics had escalated rapidly, setting the stage for two fairly significant developments. First, the release of immersive, first-person games such as Doom, Duke Nukem 3D, and Quake helped ignite a quest to create progressively more realistic 3D environments for PC gaming. Although 3D graphics would eventually work their way into nearly all computer games, the popularity of the nascent first-person shooter genre would significantly accelerate the adoption of 3D graphics in consumer computing. At the same time, companies such as NVIDIA, ATI Technologies, and 3dfx Interactive began releasing graphics accelerators that were affordable enough to attract widespread attention. These developments cemented 3D graphics as a technology that would figure prominently for years to come.

The release of NVIDIA's GeForce 256 further pushed the capabilities of consumer graphics hardware. For the first time, transform and lighting computations could be performed directly on the graphics processor, thereby enhancing the potential for even more visually interesting applications. Since transform and lighting were already integral parts of the OpenGL graphics pipeline, the GeForce 256 marked the beginning of a natural progression where increasingly more of the graphics pipeline would be implemented directly on the graphics processor.

From a parallel-computing standpoint, NVIDIA's release of the GeForce 3 series in 2001 represents arguably the most important breakthrough in GPU technology. The GeForce 3 series was the computing industry's first chip to implement Microsoft's then-new DirectX 8.0 standard. This standard required that compliant hardware contain both programmable vertex and programmable pixel shading stages. For the first time, developers had some control over the exact computations that would be performed on their GPUs.

### 1.3.2 EARLY GPU COMPUTING

The release of GPUs that possessed programmable pipelines attracted many researchers to the possibility of using graphics hardware for more than simply OpenGL- or DirectX-based rendering. The general approach in the early days of GPU computing was extraordinarily convoluted. Because standard graphics APIs such as OpenGL and DirectX were still the only way to interact with a GPU, any attempt to perform arbitrary computations on a GPU would still be subject to the constraints of programming within a graphics API. Because of this, researchers explored general-purpose computation through graphics APIs by trying to make their problems appear to the GPU to be traditional rendering.

Essentially, the GPUs of the early 2000s were designed to produce a color for every pixel on the screen using programmable arithmetic units known as *pixel shaders*. In general, a pixel shader uses its `(x,y)` position on the screen as well as some additional information to combine various inputs in computing a final color. The additional information could be input colors, texture coordinates, or other attributes that would be passed to the shader when it ran. But because the arithmetic being performed on the input colors and textures was completely controlled by the programmer, researchers observed that these input "colors" could actually be *any* data.

So if the inputs were actually numerical data signifying something other than color, programmers could then program the pixel shaders to perform arbitrary computations on this data. The results would be handed back to the GPU as the final pixel "color," although the colors would simply be the result of whatever computations the programmer had instructed the GPU to perform on their inputs. This data could be read back by the researchers, and the GPU would never be the wiser. In essence, the GPU was being tricked into performing nonrendering tasks by making those tasks appear as if they were a standard rendering. This trickery was very clever but also very convoluted.

Because of the high arithmetic throughput of GPUs, initial results from these experiments promised a bright future for GPU computing. However, the programming model was still far too restrictive for any critical mass of developers to form. There were tight resource constraints, since programs could receive input data only from a handful of input colors and a handful of texture units. There were serious limitations on how and where the programmer could write results to memory, so algorithms requiring the ability to write to arbitrary locations in memory (scatter) could not run on a GPU. Moreover, it was nearly impossible to predict how your particular GPU would deal with floating-point data, if it handled floating-point data at all, so most scientific computations would be unable to use a GPU. Finally, when the program inevitably computed the incorrect results, failed to terminate, or simply hung the machine, there existed no reasonably good method to debug any code that was being executed on the GPU.

As if the limitations weren't severe enough, anyone who *still* wanted to use a GPU to perform general-purpose computations would need to learn OpenGL or DirectX since these remained the only means by which one could interact with a GPU. Not only did this mean storing data in graphics textures and executing computations by calling OpenGL or DirectX functions, but it meant writing the computations themselves in special graphics-only programming languages known as *shading languages*. Asking researchers to both cope with severe resource and programming restrictions as well as to learn computer graphics and shading languages before attempting to harness the computing power of their GPU proved too large a hurdle for wide acceptance.

## 1.4 CUDA

It would not be until five years after the release of the GeForce 3 series that GPU computing would be ready for prime time. In November 2006, NVIDIA unveiled the industry's first DirectX 10 GPU, the GeForce 8800 GTX. The GeForce 8800 GTX was also the first GPU to be built with NVIDIA's CUDA Architecture. This architecture included several new components designed strictly for GPU computing and aimed to alleviate many of the limitations that prevented previous graphics processors from being legitimately useful for general-purpose computation.

### 1.4.1 WHAT IS THE CUDA ARCHITECTURE?

Unlike previous generations that partitioned computing resources into vertex and pixel shaders, the CUDA Architecture included a unified shader pipeline, allowing each and every arithmetic logic unit (ALU) on the chip to be marshaled by a program intending to perform general-purpose computations. Because NVIDIA intended this new family of graphics processors to be used for general-purpose computing, these ALUs were built to comply with IEEE requirements for single-precision floating-point arithmetic and were designed to use an instruction set tailored for general computation rather than specifically for graphics. Furthermore, the execution units on the GPU were allowed arbitrary read and write access to memory as well as access to a software-managed cache known as *shared memory*. All of these features of the CUDA Architecture were added in order to create a GPU that would excel at computation in addition to performing well at traditional graphics tasks.

### 1.4.2 USING THE CUDA ARCHITECTURE

The effort by NVIDIA to provide consumers with a product for both computation and graphics could not stop at producing hardware incorporating the CUDA Architecture, though. Regardless of how many features NVIDIA added to its chips to facilitate computing, there continued to be no way to access these features without using OpenGL or DirectX. Not only would this have required users to continue to disguise their computations as graphics problems, but they would have needed to continue writing their computations in a graphics-oriented shading language such as OpenGL's GLSL or Microsoft's HLSL.

To reach the maximum number of developers possible, NVIDIA took industry-standard C and added a relatively small number of keywords in order to harness some of the special features of the CUDA Architecture. A few months after the launch of the GeForce 8800 GTX, NVIDIA made public a compiler for this language, CUDA C. And with that, CUDA C became the first language specifically designed by a GPU company to facilitate general-purpose computing on GPUs.

In addition to creating a language to write code for the GPU, NVIDIA also provides a specialized hardware driver to exploit the CUDA Architecture's massive computational power. Users are no longer required to have any knowledge of the OpenGL or DirectX graphics programming interfaces, nor are they required to force their problem to look like a computer graphics task.

## 1.5 Applications of CUDA

Since its debut in early 2007, a variety of industries and applications have enjoyed a great deal of success by choosing to build applications in CUDA C. These benefits often include orders-of-magnitude performance improvement over the previous state-of-the-art implementations. Furthermore, applications running on NVIDIA graphics processors enjoy superior performance per dollar and performance per watt than implementations built exclusively on traditional central processing technologies. The following represent just a few of the ways in which people have put CUDA C and the CUDA Architecture into successful use.

### 1.5.1 MEDICAL IMAGING

The number of people who have been affected by the tragedy of breast cancer has dramatically risen over the course of the past 20 years. Thanks in a large part to the tireless efforts of many, awareness and research into preventing and curing this terrible disease has similarly risen in recent years. Ultimately, every case of breast cancer should be caught early enough to prevent the ravaging side effects of radiation and chemotherapy, the permanent reminders left by surgery, and the deadly consequences in cases that fail to respond to treatment. As a result, researchers share a strong desire to find fast, accurate, and minimally invasive ways to identify the early signs of breast cancer.

The mammogram, one of the current best techniques for the early detection of breast cancer, has several significant limitations. Two or more images need to be taken, and the film needs to be developed and read by a skilled doctor to identify potential tumors. Additionally, this X-ray procedure carries with it all the risks of repeatedly radiating a patient's chest. After careful study, doctors often require further, more specific imaging—and even biopsy—in an attempt to eliminate the possibility of cancer. These false positives incur expensive follow-up work and cause undue stress to the patient until final conclusions can be drawn.

Ultrasound imaging is safer than X-ray imaging, so doctors often use it in conjunction with mammography to assist in breast cancer care and diagnosis. But conventional breast ultrasound has its limitations as well. As a result, TechniScan Medical Systems was born. TechniScan has developed a promising, three-dimensional, ultrasound imaging method, but its solution had not been put into practice for a very simple reason: computation limitations. Simply put, converting the gathered ultrasound data into the three-dimensional imagery required computation considered prohibitively time-consuming and expensive for practical use.

The introduction of NVIDIA's first GPU based on the CUDA Architecture along with its CUDA C programming language provided a platform on which TechniScan could convert the dreams of its founders into reality. As the name indicates, its Svara ultrasound imaging system uses ultrasonic waves to image the patient's chest. The TechniScan Svara system relies on two NVIDIA Tesla C1060 processors in order to process the 35GB of data generated by a 15-minute scan. Thanks to the computational horsepower of the Tesla C1060, within 20 minutes the doctor can manipulate a highly detailed, three-dimensional image of the woman's breast. TechniScan expects wide deployment of its Svara system starting in 2010.

### 1.5.2 COMPUTATIONAL FLUID DYNAMICS

For many years, the design of highly efficient rotors and blades remained a black art of sorts. The astonishingly complex movement of air and fluids around these devices cannot be effectively modeled by simple formulations, so accurate simulations prove far too computationally expensive to be realistic. Only the largest supercomputers in the world could hope to offer computational resources on par with the sophisticated numerical models required to develop and validate designs. Since few have access to such machines, innovation in the design of such machines continued to stagnate.

The University of Cambridge, in a great tradition started by Charles Babbage, is home to active research into advanced parallel computing. Dr. Graham Pullan and Dr. Tobias Brandvik of the "many-core group" correctly identified the potential in NVIDIA's CUDA Architecture to accelerate computational fluid dynamics unprecedented levels. Their initial investigations indicated that acceptable levels of performance could be delivered by GPU-powered, personal workstations. Later, the use of a small GPU cluster easily outperformed their much more costly supercomputers and further confirmed their suspicions that the capabilities of NVIDIA's GPU matched extremely well with the problems they wanted to solve.

For the researchers at Cambridge, the massive performance gains offered by CUDA C represent more than a simple, incremental boost to their supercomputing resources. The availability of copious amounts of low-cost GPU computation empowered the Cambridge researchers to perform rapid experimentation. Receiving experimental results within seconds streamlined the feedback process on which researchers rely in order to arrive at breakthroughs. As a result, the use of GPU clusters has fundamentally transformed the way they approach their research. Nearly interactive simulation has unleashed new opportunities for innovation and creativity in a previously stifled field of research.

### 1.5.3 ENVIRONMENTAL SCIENCE

The increasing need for environmentally sound consumer goods has arisen as a natural consequence of the rapidly escalating industrialization of the global economy. Growing concerns over climate change, the spiraling prices of fuel, and the growing level of pollutants in our air and water have brought into sharp relief the collateral damage of such successful advances in industrial output. Detergents and cleaning agents have long been some of the most necessary yet potentially calamitous consumer products in regular use. As a result, many scientists have begun exploring methods for reducing the environmental impact of such detergents without reducing their efficacy. Gaining something for nothing can be a tricky proposition, however.

The key components to cleaning agents are known as *surfactants*. Surfactant molecules determine the cleaning capacity and texture of detergents and shampoos, but they are often implicated as the most environmentally devastating component of cleaning products. These molecules attach themselves to dirt and then mix with water such that the surfactants can be rinsed away along with the dirt. Traditionally, measuring the cleaning value of a new surfactant would require extensive laboratory testing involving numerous combinations of materials and impurities to be cleaned. This process, not surprisingly, can be very slow and expensive.

Temple University has been working with industry leader Procter & Gamble to use molecular simulation of surfactant interactions with dirt, water, and other materials. The introduction of computer simulations serves not just to accelerate a traditional lab approach, but it extends the breadth of testing to numerous variants of environmental conditions, far more than could be practically tested in the past. Temple researchers used the GPU-accelerated Highly Optimized Object-oriented Many-particle Dynamics (HOOMD) simulation software written by the Department of Energy's Ames Laboratory. By splitting their simulation across two

NVIDIA Tesla GPUs, they were able achieve equivalent performance to the 128 CPU cores of the Cray XT3 and to the 1024 CPUs of an IBM BlueGene/L machine. By increasing the number of Tesla GPUs in their solution, they are already simulating surfactant interactions at 16 times the performance of previous platforms. Since NVIDIA's CUDA has reduced the time to complete such comprehensive simulations from several weeks to a few hours, the years to come should offer a dramatic rise in products that have both increased effectiveness and reduced environmental impact.

## 1.6 Chapter Review

The computing industry is at the precipice of a parallel computing revolution, and NVIDIA's CUDA C has thus far been one of the most successful languages ever designed for parallel computing. Throughout the course of this book, we will help you learn how to write your own code in CUDA C. We will help you learn the special extensions to C and the application programming interfaces that NVIDIA has created in service of GPU computing. You are *not* expected to know OpenGL or DirectX, nor are you expected to have any background in computer graphics.

We will not be covering the basics of programming in C, so we do not recommend this book to people completely new to computer programming. Some familiarity with parallel programming might help, although we do not *expect* you to have done any parallel programming. Any terms or concepts related to parallel programming that you will need to understand will be explained in the text. In fact, there may be some occasions when you find that knowledge of traditional parallel programming will cause you to make assumptions about GPU computing that prove untrue. So in reality, a moderate amount of experience with C or C++ programming is the only prerequisite to making it through this book.

In the next chapter, we will help you set up your machine for GPU computing, ensuring that you have both the hardware and the software components necessary get started. After that, you'll be ready to get your hands dirty with CUDA C. If you already have some experience with CUDA C or you're sure that your system has been properly set up to do development in CUDA C, you can skip to Chapter 3.