Professional CUDA® C Programming

Foreword by Dr. Barbara Chapman, Center for Advanced Computing & Data Systems, University of Houston

John Cheng, Max Grossman, Ty McKercher

PROFESSIONAL

CUDA® C Programming

John Cheng
Max Grossman
Ty McKercher

Professional CUDA® C Programming

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-1-118-73932-7

ISBN: 978-1-118-73927-3 (ebk)

ISBN: 978-1-118-73931-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2014937184

CREDITS

ACQUISITIONS EDITOR
Mary James

PROJECT EDITOR
Martin V. Minner

TECHNICAL EDITORS
Wei Zhang
Chao Zhao

PRODUCTION MANAGER
Kathleen Wisor

COPY EDITOR
Katherine Burt

MANAGER OF CONTENT DEVELOPMENT AND ASSEMBLY
Mary Beth Wakefield

DIRECTOR OF COMMUNITY MARKETING
David Mayhew

MARKETING MANAGER
Carrie Sherrill

BUSINESS MANAGER
Amy Knies

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Patrick Redmond

PROOFREADER
Nancy Carrasco

INDEXER
Johnna VanHoose Dinse

COVER DESIGNER
Wiley

COVER IMAGE
© iStock.com/fatido

ABOUT THE AUTHORS

JOHN (RUNWEI) CHENG is a research scientist with extensive industry experience in high-performance computing on heterogeneous computing platforms. Before joining the oil and gas industry, John worked in the finance industry for more than ten years as an expert in computational intelligence, providing advanced solutions based on genetic algorithms hybridized with data mining and statistical learning to solve real world business challenges. As an internationally recognized researcher in the field of genetic algorithms and their application to industrial engineering, John has co-authored three books. John's first book, *Genetic Algorithms and Engineering Design*, published by John Wiley and Sons in 1997, is still used as a textbook in universities worldwide. John has a wide range of experience in both academic research and industry development, and is gifted in making complex subjects accessible to readers with a concise, illustrative, and edifying approach. John earned his doctoral degree in computational intelligence from the Tokyo Institute of Technology.

MAX GROSSMAN has been working as a developer with various GPU programming models for nearly a decade. His experience is focused in developing novel GPU programming models and implementing scientific algorithms on GPU hardware. Max has applied GPUs to a wide range of domains, including geoscience, plasma physics, medical imaging, and machine learning, and enjoys understanding the computational patterns of new domains and finding new and unusual ways to apply GPUs to them. Lessons learned from these domains help to guide Max's work in programming models and frameworks. Max earned his degree in computer science from Rice University with a focus in parallel computing.

TY MCKERCHER is a Principal Solution Architect with NVIDIA, leading a team that specializes in visual computing systems architecture across multiple industries. He often serves as a liaison between customer and product engineering teams during emerging technology evaluations. He has been engaged in CUDA-based projects since he participated in the first CUDA kitchen training session held at NVIDIA headquarters in 2006. Since then, Ty has helped architect GPU-based supercomputing environments at some of the largest and most demanding production datacenters in the world. Ty earned his mathematics degree with emphasis in geophysics and computer science from the Colorado School of Mines.

ABOUT THE TECHNICAL EDITORS

WEI ZHANG is a scientific programmer and has been working in the high-performance computing area for 15 years. He has developed or co-developed many scientific software packages for molecular simulation, computer-aided drug design, EM structure reconstruction, and seismic depth imaging. He is now focusing his effort on improving the performance of seismic data processing using new technologies such as CUDA.

CHAO ZHAO joined Chevron in 2008 and currently serves as Geophysical Application Software Development Specialist. In this role, Chao is responsible for designing and developing software products for geoscientists. Prior to joining Chevron, Chao was a software developer for Knowledge Systems Inc. and Seismic Micro Technology Inc. With more than 13 years of software developing experience in the exploration and production industry, Chao has gained rich knowledge in the fields of geology and geophysics. Having a broad education in science, Chao likes to see CUDA programming used widely in scientific research and enjoys contributing to it as much as he can. He holds a Bachelor of Science degree in chemistry from Peking University and a Master of Science in computer science from the University of Rhode Island.

ACKNOWLEDGMENTS

IT WOULD BE HARD TO IMAGINE this project making it to the finish line without the suggestions, constructive criticisms, help, and resources of our colleagues and friends.

We would like to express our thanks to NVIDIA for granting access to many GTC conference presentations and CUDA technical documents that add both great value and authority to this book.

In particular, we owe much gratitude to Dr. Paulius Micikevicius and Dr. Peng Wang, Developer Technology Engineers at NVIDIA, for their kind advice and help during the writing of this book. Special thanks to Mark Ebersole, NVIDIA Chief CUDA Educator, for his guidance and feedback during the review process.

We would like to thank Mr. Will Ramey, Sr. Product Manager at NVIDIA, and Mr. Nadeem Mohammad, Product Marketing at NVIDIA, for their support and encouragement during the entire project.

We would like to thank Mr. Paul Holzhauer, Director of Oil & Gas at NVIDIA, for his support during the initial phase of this project.

Especially, we owe an enormous debt of gratitude to many presenters and speakers in past GTC conferences for their inspiring and creative work on GPU computing technologies. We have recorded all your credits in our suggested reading lists.

After years of work using GPUs in real production projects, John is very grateful to the people who helped him become a GPU computing enthusiast. Especially, John would like to thank Dr. Nanxun Dai and Dr. Bao Zhao for their encouragement, support, and guidance on seismic imaging projects at BGP. John also would like to thank his colleagues Dr. Zhengzhen Zhou, Dr. Wei Zhang, Mrs. Grace Zhang, and Mr. Kai Yang. They are truly brilliant and very pleasant to work with. John loves the team and feels very privileged to be one of them. John would like to extend a special thanks to Dr. Mitsuo Gen, an internationally well-known professor, the supervisor of John's doctoral program, for giving John the opportunity to teach at universities in Japan and co-author academic books, especially for his fully supporting John during the years when John was running a startup based on evolutionary computation technologies in Tokyo. John is very happy working on this project with Ty and Max as a team and learned a lot from them during the process of book writing. John owes a debt of gratitude to his wife, Joly, and his son, Rick, for their love, support, and considerable patience during evenings and weekends over the past year while Dad was yet again "doing his own book work."

For over 25 years, Ty has been helping software developers solve HPC grand challenges. Ty is delighted to work at NVIDIA to help clients extend their current knowledge to unlock the potential from massively parallel GPUs. There are so many NVIDIANs to thank, but Ty would like to specifically recognize Dr. Paulius Micikevicius for his gifted insights and strong desire to always improve while doing the heavy lifting for numerous projects. When John asked Ty to help share

CUDA knowledge in a book project, he welcomed the challenge. Dave Jones, NVIDIA, senior director approved Ty's participation in this project, and sadly last year Dave lost his courageous battle against cancer. Our hearts go out to Dave and his family — his memory serves to inspire, to press on, and to pursue your passions. The encouragements from Shanker Trivedi and Marc Hamilton have been especially helpful. Yearning to maintain his life/work balance, Ty recruited Max to join this project. It was truly a pleasure to learn from John and Max as they developed the book content that Ty helped review. Finally, Ty's wife, Judy, and his four children deserve recognition for their unconditional support and love — it is a blessing to receive encouragement and motivation while pursuing those things that bring joy to your life.

Max has been fortunate to collaborate with and be guided by a number of brilliant and talented engineers, researchers, and mentors. First, thanks have to go to Professor Vivek Sarkar and the whole Habanero Research Group at Rice University. There, Max got his first taste of HPC and CUDA. The mentorship of Vivek and others in the group was invaluable in enabling him to explore the exciting world of research. Max would also like to thank Mauricio Araya-Polo and Gladys Gonzalez at Repsol. The experience gained under their mentorship was incredibly valuable in writing a book that would be truly useful to real-world work in science and engineering. Finally, Max would like to thank John and Ty for inviting him along on this writing adventure in CUDA and for the lessons this experience has provided in CUDA, writing, and life.

It would not be possible to make a quality professional book without input from technical editors, development editors, and reviewers. We would like to extend our sincere appreciation to Mary E. James, our acquisitions editor; Martin V. Minner, our project editor; Katherine Burt, our copy editor; and Wei Zhang and Chao Zhao, our technical editors. You are an insightful and professional editorial team and this book would not be what it is without you. It was a great pleasure to work with you on this project.

CONTENTS

FOREWORD

GPUs have come a long way. From their origins as specialized graphics processors that could rapidly produce images for output to a display unit, they have become a go-to technology when ultra-fast processing is needed. In the past few years, GPUs have increasingly been attached to CPUs to accelerate a broad array of computations in so-called *heterogeneous computing*. Today, GPUs are configured on many desktop systems, on compute clusters, and even on many of the largest supercomputers in the world. In their extended role as a provider of large amounts of compute power for technical computing, GPUs have enabled advances in science and engineering in a broad variety of disciplines. They have done so by making it possible for huge numbers of compute cores to work in parallel while keeping the power budgets very reasonable.

Fortunately, the interfaces for programming GPUs have kept up with this rapid change. In the past, a major effort was required to use them for anything outside the narrow range of applications they were intended for, and the GPU programmer needed to be familiar with many concepts that made good sense only to the graphics programmer. Today's systems provide a much more convenient means to create application software that will run on them. In short, we have CUDA.

CUDA is one of the most popular application programming interfaces for accelerating a range of compute kernels on the GPU. It can enable code written in C or C++ to run efficiently on a GPU with very reasonable programming effort. It strikes a balance between the need to know about the architecture in order to exploit it well, and the need to have a programming interface that is easy to use and results in readable programs.

This book will be a valuable resource for anyone who wants to use GPUs for scientific and technical programming. It provides a comprehensive introduction to the CUDA programming interface and its usage. For a start, it describes the basics of parallel computing on heterogeneous architectures and introduces the features of CUDA. It then explains how CUDA programs are executed. CUDA exposes the execution and memory model to the programmer; as a result, the CUDA programmer has direct control of the massively parallel environment. In addition to giving details of the CUDA memory model, the text provides a wealth of information on how it can be utilized. The following chapter discusses streams, as well as how to execute concurrent and overlapping kernels. Next comes information on tuning, on using CUDA libraries, and on using OpenACC directives to program GPUs. After a chapter on multi-GPU programming, the book concludes by discussing some implementation considerations. Moreover, a variety of examples are given to help the reader get started, many of which can be downloaded and executed.

CUDA provides a nice balance between expressivity and programmability that has proven itself in practice. However, those of us who have made it their mission to simplify application development know that this is an on-going story. For the past few years, CUDA researchers have worked to improve heterogeneous programming tools. CUDA 6 introduces many new features, including unified memory and plug-in libraries, to make GPU programming even easier. They have also provided a set of directives called OpenACC, which is introduced in this book. OpenACC promises to

complement CUDA by offering an even simpler means to exploit GPU programming power when less direct control over the execution is needed. Results so far are very promising. OpenACC, CUDA 6, and other topics covered in this book will allow CUDA developers to accelerate their applications for more performance than ever. This book will need to have a permanent place on your bookshelf.

Happy programming!

Barbara Chapman
CACDS and Department of Computer Science
University of Houston

PREFACE

Years ago when we were porting our production code from legacy C programs to CUDA C, we encountered many troubles as any beginner does, problems with solutions that were far beyond what you could dig out of a simple web search. At that time, we thought that it would be great if there were a book written *by* programmers, *for* programmers, that focused on what programmers need for production CUDA development. Fulfilling that need with lessons from our own experiences in CUDA is the motivation for this book. This book is specially designed to address the needs of the high-performance and scientific computing communities.

When learning a new framework or programming language, most programmers drag out a piece of code from anywhere, test it, and then build up their own code based on that trial. Learning by example with a trial-and-error approach is a quintessential learning technique for many software developers. This book is designed to fit these habits. Each chapter focuses on one topic, using concise explanations to provide foundational knowledge, and illustrating concepts with simple and fully workable code samples. Learning concepts and code side-by-side empowers you to quickly start experimenting with these topics. This book uses a profile-driven approach to guide you deeper and deeper into each topic.

The major difference between parallel programming in C and parallel programming in CUDA C is that CUDA architectural features, such as memory and execution models, are exposed directly to programmers. This enables you to have more control over the massively parallel GPU environment. Even though some still consider CUDA concepts to be low-level, having some knowledge of the underlying architecture is a necessity for harnessing the power of GPUs. Actually, the CUDA platform can perform well even if you have limited knowledge of the architecture.

Parallel programming is always motivated by performance and driven by profiling. CUDA programming is unique in that the exposed architectural features enable you, the programmer, to extract every iota of performance from this powerful hardware platform, if you so choose. After you have mastered the skills taught through the exercises provided in this book, you will find that programming in CUDA C is easy, enjoyable, and rewarding.

INTRODUCTION

WELCOME TO THE WONDERFUL WORLD of heterogeneous parallel programming with CUDA C!

Modern heterogeneous systems are evolving toward a future of intriguing computational possibilities. Heterogeneous computing is constantly being applied to new fields of computation — everything from science to databases to machine learning. The future of programming is heterogeneous parallel programming!

This book gets you started quickly with GPU (Graphical Processing Unit) computing using the CUDA platform, CUDA Toolkit, and CUDA C language. The examples and exercises in this book are designed to jump-start your CUDA expertise to a professional level!

WHO THIS BOOK IS FOR

This book is for anyone who wants to leverage the power of GPU computing to accelerate applications. It covers the most up-to-date technologies in CUDA C programming, with a focus on:

- Concise style
- Straightforward approach
- Illustrative description
- Extensive examples
- Deliberately designed exercises
- Comprehensive coverage
- Content well-focused for the needs of high-performance computing

If you are an experienced C programmer who wants to add high-performance computing to your repertoire by learning CUDA C, the examples and exercises in the book will build on your existing knowledge so as to simplify mastering CUDA C programming. Using just a handful of CUDA extensions to C, you can benefit from the power of massively parallel hardware. The CUDA platform, programming models, tools, and libraries make programming heterogeneous architectures straightforward and immediately rewarding.

If you are a professional with domain expertise outside of computer science who wants to quickly get up to speed with parallel programming on GPUs, maximize your productivity, and enhance the performance of your applications, you have picked the right book. The clear and concise explanations in this book, supported by well-designed examples and guided by a profile-driven approach, will help you gain insight into GPU programming and quickly become proficient with CUDA.

If you are a professor or a researcher in any discipline and wish to accelerate discovery and innovation through GPU computing, this book will improve your time-to-solution. With minimal past programming experience, parallel computing concepts, and knowledge of computer science, you can quickly dive into the exciting world of parallel programming with heterogeneous architectures.

If you are new to C but are interested in exploring heterogeneous programming, this book does not assume copious amounts of experience in C programming. While the CUDA C and C programming languages obviously share some syntax, the abstractions and underlying hardware for each are different enough that experience with one does not make the other significantly easier to learn. As long as you have an interest in heterogeneous programming, are excited about new topics and new ways of thinking, and have a passion for deep understanding of technical topics, this book is a great fit for you.

Even if you have experience with CUDA C, this book can still be a useful tool to refresh your knowledge, discover new tools, and gain insight into the latest CUDA features. While this book is designed to create CUDA professionals from scratch, it also provides a comprehensive overview of many advanced CUDA concepts, tools, and frameworks that will benefit existing CUDA developers.

WHAT THIS BOOK COVERS

This book provides foundational concepts and techniques of CUDA C programming for people that need to drastically accelerate the performance of their applications. This book covers the newest features released with CUDA Toolkit 6.0 and NVIDIA Kepler GPUs. After briefly introducing the paradigm shift in parallel programming from homogeneous architectures to heterogeneous architectures, this book guides you through essential programming skills and best practices in CUDA, including but not limited to the CUDA programming model, GPU execution model, GPU memory model, CUDA streams and events, techniques for programming multiple GPUs, CUDA-aware MPI programming, and NVIDIA development tools.

This book takes a unique approach to teaching CUDA by mingling foundational descriptions of concepts with illustrative examples that use a profile-driven approach to guide you toward optimal performance. Each topic is thoroughly covered in a step-by-step process based heavily on code examples. This book will help you quickly master the CUDA development process by teaching you not only how to use CUDA-based tools, but also how to interpret results in each step of the development process based on insights and intuitions from the abstract programming model.

Each chapter handles one main topic with workable code examples to demonstrate the basic features and techniques of GPU programming, followed by well-designed exercises that facilitate your exploration of each topic to deepen your understanding.

All examples are developed using a Linux system with CUDA 5.0 or higher and a Kepler or Fermi GPU. Since CUDA C is a cross-platform language, examples in the book are also applicable to other platforms, such as embedded systems, tablets, notebooks, PCs, workstations, and high-performance computing servers. Many OEM suppliers support NVIDIA GPUs in a variety of form-factors.

HOW THIS BOOK IS STRUCTURED

This book consists of ten chapters, and covers the following topics:

Chapter 1: Heterogeneous Parallel Computing with CUDA begins with a brief introduction to the heterogeneous architecture that complements CPUs with GPUs, as well as the paradigm shift towards heterogeneous parallel programming.

Chapter 2: CUDA Programming Model introduces the CUDA programming model and the general structure of a CUDA program. It explains the logical view for massively parallel computing in CUDA: two levels of thread hierarchy exposed intuitively through the programming model. It also discusses thread configuration heuristics and their impact on performance.

Chapter 3: CUDA Execution Model inspects kernel execution from the hardware point of view by studying how thousands of threads are scheduled on a GPU. It explains how compute resources are partitioned among threads at multiple granularities. It also shows how the hardware view can be used to guide kernel design, and guides you in developing and optimizing a kernel using a profile-driven approach. Then, CUDA dynamic parallelism and nested execution are illustrated with examples.

Chapter 4: Global Memory introduces the CUDA memory model, probes the global memory data layout, and analyzes access patterns to global memory. This chapter explains the performance implications of various memory access patterns and demonstrates how a new feature in CUDA 6, Unified Memory, can simplify CUDA programming and improve your productivity.

Chapter 5: Shared Memory and Constant Memory explains how shared memory, a program-managed low-latency cache, can be used to improve kernel performance. It describes the optimal data layout for shared memory and illustrates how to avoid poor performance. Last, it illustrates how to perform low-latency communication between neighboring threads.

Chapter 6: Streams and Concurrency describes how multi-kernel concurrency can be implemented with CUDA streams, how to overlap communication and computation, and how different job dispatching strategies affect inter-kernel concurrency.

Chapter 7: Tuning Instruction-Level Primitives explains the nature of floating-point operations, standard and intrinsic mathematical functions, and CUDA atomic operations. It shows how to use relatively low-level CUDA primitives and compiler flags to tune the performance, accuracy, and correctness of an application.

Chapter 8: GPU-Accelerated CUDA Libraries and OpenACC introduces a new level of parallelism with CUDA domain-specific libraries, including specific examples in linear algebra, Fourier transforms, and random number generation. It explains how OpenACC, a compiler-directive-based GPU programming model, complements CUDA by offering a simpler means to exploit GPU computational power.

Chapter 9: Multi-GPU Programming introduces GPUDirect technology for peer-to-peer GPU memory access. It explains how to manage and execute computation across multiple GPUs. It also

illustrates how to scale applications across a GPU-accelerated compute cluster by using CUDA-aware MPI with GPUDirect RDMA to realize near linear performance scalability.

Chapter 10: Implementation Considerations discusses the CUDA development process and a variety of profile-driven optimization strategies. It demonstrates how to use CUDA debugging tools to debug kernel and memory errors. It also provides a case study in porting a legacy C application to CUDA C using step-by-step instructions to help solidify your understanding of the methodology, visualize the process, and demonstrate the tools.

WHAT YOU NEED TO USE THIS BOOK

This book does not require either GPU or parallel programming experience. Before you jump in, it would be best if you have basic experience working with Linux. To run all examples in the book, the ideal environment is:

- A Linux system
- A C/C++ compiler
- CUDA 6.0 Toolkit installed
- NVIDIA Kepler GPU

However, most examples will run on Fermi devices, though some examples using CUDA 6 features might require Kepler GPUs. Most of these examples can be compiled with CUDA 5.5.

CUDA TOOLKIT DOWNLOAD

You can download the CUDA 6.0 Toolkit from `https://developer.nvidia.com/cuda-toolkit`.

The CUDA Toolkit includes a compiler for NVIDIA GPUs, CUDA math libraries, and tools for debugging and optimizing the performance of your applications. You will also find programming guides, user manuals, an API reference, and other documentation to help you start accelerating your application with GPUs.

CONVENTIONS

To help you get the most from the text, we have used a number of conventions throughout the book.

We *highlight new terms* and important words when we they are introduced.

We show file names, URLs, and code within the text like so: `this_is_a_kernel_file.cu`.

We present code in following way:

```
// distributing jobs among devices
for (int i = 0; i < ngpus; i++)
{
   cudaSetDevice(i);
   cudaMemcpyAsync(d_A[i], h_A[i], iBytes, cudaMemcpyDefault,stream[i]);
   cudaMemcpyAsync(d_B[i], h_B[i], iBytes, cudaMemcpyDefault,stream[i]);
   iKernel<<<grid, block,0,stream[i]>>> (d_A[i], d_B[i], d_C[i],iSize);
   cudaMemcpyAsync(gpuRef[i], d_C[i], iBytes, cudaMemcpyDefault,stream[i]);
}
```

We introduce CUDA runtime functions in the following way:

```
cudaError_t cudaDeviceSynchronize (void);
```

We present the output of programs as follows:

```
./reduce starting reduction at device 0: Tesla M2070
     with array size 16777216  grid 32768 block 512
cpu reduce      elapsed 0.029138 sec cpu_sum: 2139353471
gpu Warmup      elapsed 0.011745 sec gpu_sum: 2139353471 <<<grid 32768 block 512>>>
gpu Neighbored  elapsed 0.011722 sec gpu_sum: 2139353471 <<<grid 32768 block 512>>>
```

We give command-line instructions as follows:

```
$ nvprof --devices 0 --metrics branch_efficiency ./reduce
```

SOURCE CODE

As you work through the examples in this book, you might choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `www.wrox.com/go/procudac`. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

When you work on the exercises at the end of each chapter, we highly encourage you to try to write them yourself by referencing the example codes. All the exercise code files are also downloadable from the Wrox website.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you might save another reader hours of frustration and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to www.wrox.com/go/procudac. Then, on the book's details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature where topics of interest of your choosing when new posts are made to the forums can be sent to you via e-mail. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At http://p2p.wrox.com you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join. Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum sent to your e-mail address, click the "Subscribe to this Forum" icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

USEFUL LINKS

GTC On-Demand: http://on-demand-gtc.gputechconf.com/gtcnew/on-demand-gtc.php

GTC Express Webinar Program: https://developer.nvidia.com/gpu-computing-webinars

Developer Zone: www.gputechconf.com/resources/developer-zone

NVIDIA Parallel Programming Blog: http://devblogs.nvidia.com/parallelforall

NVIDIA Developer Zone Forums: devtalk.nvidia.com

NVIDIA support e-mail: devtools-support@nvidia.com

1

Heterogeneous Parallel Computing with CUDA

WHAT'S IN THIS CHAPTER?

- Understanding heterogeneous computing architectures
- Recognizing the paradigm shift of parallel programming
- Grasping the basic elements of GPU programming
- Knowing the differences between CPU and GPU programming

CODE DOWNLOAD *The wrox.com code downloads for this chapter are found at* `www.wrox.com/go/procudac` *on the Download Code tab. The code is in the Chapter 1 download and individually named according to the names throughout the chapter.*

The *high-performance computing* (HPC) landscape is always changing as new technologies and processes become commonplace, and the definition of HPC changes accordingly. In general, it pertains to the use of multiple processors or computers to accomplish a complex task concurrently with high throughput and efficiency. It is common to consider HPC as not only a computing architecture but also as a set of elements, including hardware systems, software tools, programming platforms, and parallel programming paradigms.

Over the last decade, high-performance computing has evolved significantly, particularly because of the emergence of GPU-CPU heterogeneous architectures, which have led to a fundamental paradigm shift in parallel programming. This chapter begins your understanding of heterogeneous parallel programming.

PARALLEL COMPUTING

During the past several decades, there has been ever-increasing interest in parallel computation. The primary goal of parallel computing is to improve the speed of computation.

From a pure calculation perspective, *parallel computing* can be defined as a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently.

From the programmer's perspective, a natural question is how to map the concurrent calculations onto computers. Suppose you have multiple computing resources. Parallel computing can then be defined as the simultaneous use of multiple computing resources (cores or computers) to perform the concurrent calculations. A large problem is broken down into smaller ones, and each smaller one is then solved concurrently on different computing resources. The software and hardware aspects of parallel computing are closely intertwined together. In fact, parallel computing usually involves two distinct areas of computing technologies:

- Computer architecture (hardware aspect)
- Parallel programming (software aspect)

Computer architecture focuses on supporting parallelism at an architectural level, while *parallel programming* focuses on solving a problem concurrently by fully using the computational power of the computer architecture. In order to achieve parallel execution in software, the hardware must provide a platform that supports concurrent execution of multiple processes or multiple threads.

Most modern processors implement the *Harvard architecture*, as shown in Figure 1-1, which is comprised of three main components:

- Memory (instruction memory and data memory)
- Central processing unit (control unit and arithmetic logic unit)
- Input/Output interfaces

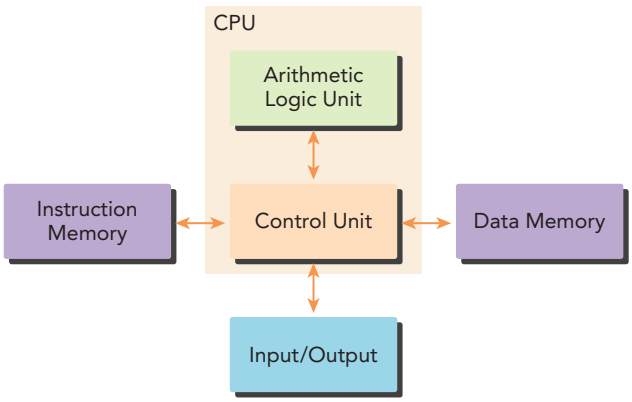

FIGURE 1-1