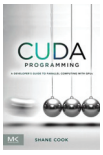
Editors: Larry Engelhardt, lengelhardt@fmarion.edu
Stephen P. Weppner, weppnesp@eckerd.edu

ALL THE WAY TO CUDA

By Thomas Vogel

Shane Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, Morgan Kaufmann, 2012, ISBN: 978-0124159334, 563 pp.



"Serial programming is dead." This is the peg and main theme of this Compute Unified Device Architecture (CUDA) textbook, and once you start reading, the author relentlessly guides you through the transition from serial (CPU) programming, which almost all of us grew up with, to massively parallel programming of GPUs. It's no secret that CPUs reached some physical limitations and haven't gotten much faster over the last several years. Consequently, if we want to make progress, we have no option but to go parallel in some way; and GPU programming using CUDA has turned out to be a promising option.

There aren't that many, but this is the most comprehensive textbook about CUDA programming I've seen so far. The 563-page book is highly detailed (sometimes maybe a little too much so), up-to-date, and provides both straightforward help for CUDA beginners and advanced topics for experienced GPU programmers concerning tuning or optimization. It isn't a pure "introduction to" or guide to playing around with GPUs, nor is it a reference work for finding quick information. Rather, it's a textbook for someone at the early stage of the transition to parallel GPU computing who seriously wants to dive into that world. In this regard, it quite nicely complements other publications like the *GPU: Computing Gems* series or *CUDA by Example* (both reviewed in earlier issues of *CiSE*[1,2]). In any case, as the subtitle suggests, the book addresses developers of some kind. That is, the author expects a solid background and knowledge in serial programming as well as basic understanding of hardware architectures.

Topics Covered

In the first chapters the author takes some time to set the stage, beginning with a nicely written review of the history of supercomputing. He then discusses parallelism with GPUs, why it might be useful or even far superior to CPU programming, and why you would use CUDA in the first place. (The author talks exclusively about CUDA in this book. Don't expect anything else.) At the end of the introductory section, he gives an overview about the different GPU hardwares and compute levels and guides the reader through the actual installation of CUDA under different operating systems (Windows, Linux, and Mac). Massive parallelization on GPUs isn't always as straightforward as problem-based parallelization using "only" a couple of cores on CPUs or CPU clusters, and there are a lot of concepts which might also not be completely straightforward to understand for "serial programmers." Hence, in the following the author introduces and clarifies basic GPU programming terminologies (such as threads, warps, and blocks) and parallelization approaches. He presents a first basic common example (filling a histogram), demonstrating this challenge and analyzing different programming strategies in detail.

When engineers or scientists use computers for their calculations, they naturally care about their specific problems rather than computer science. It is, of course, a dream that we can abstract from all hardware details and still write well performing code—that is, let compilers do all of the optimization and memory handling work. It's highly likely, though, that this dream will stay a dream for quite a while, and consequently the author devotes the entire next chapter to the problem of memory handling, which is even more complex on GPUs than on CPUs. This chapter is the second largest in the book, another indication of the author's priorities and intentions. In contrast to how code often evolves—in a "top to bottom" way, where a correctly running but poorly performing code will be successively optimized—he chooses the other way, where memory-optimized (and hence speed-optimized) code is developed from the beginning. That is, right from the start problems are laid out to efficiently use registers, shared memory, and global memory. Even though we might not always follow that approach in daily work for several reasons, it provides an insightful perspective. Everyday problems

like sorting are analyzed in detail in this context.

In chapter 7, a couple of examples of specialized uses of GPUs are discussed, for example how to implement AES encryption. Even though these specific topics might not be particularly relevant for engineers or scientists, they again provide insight into the differences of code design for GPUs versus CPUs. The next chapter, however, is of particular interest as it touches on a problem that any scientific GPU user will encounter sooner or later: If I have multiple GPUs available, how do I make good use of them? This chapter contains a lot of example source code and emphasizes inhomogeneous environments where GPUs of different generations work together.

Chapter 9 is the heart of the book. For more than 130 pages it discusses the main paths to optimize CUDA performance. The author now assumes that all previous chapters have been read and uses a strategic, step-by-step approach rather than a collection of examples. It starts off with strategies to break down the problem, setting design goals and finding serial bottlenecks and sources of limits. Based on that, considerations for memory organization are made and data transfer is discussed. An extremely useful aspect here is timing on GPUs—that is, how to actually measure GPU performance. The next steps include compiler optimizations and the use of dedicated hardware to evaluate trigonometric and other costly functions, before talking about selecting the best algorithms for certain problems. The point is that the best CPU algorithms are generally not efficient on GPUs (quicksort is an example of this), and vice versa. Then, more technically, the author discusses external analysis tools to identify bottlenecks. In particular, he introduces Nvidia Visual Profiler and Parallel Nsight. Finally, self-tuning strategies are mentioned, where a code first analyzes the available hardware and then "adapts" to make best use of it.

Very useful for scientific applications is the next chapter, which starts with a discussion of existing CUDA libraries—such as Nvidia Performance Primitives (NPP), thrust, CuRand, and CuBLAS (BLAS stands for Basic Linear Algebra Subprograms). The chapter is more example-based and contains a lot of source code. A nice feature is the links to relevant examples in the software development kit (SDK). A few examples from the SDK are discussed in detail. Finally, directive-based programming is mentioned—not in great detail, more as a rough introduction of how to use GPUs in the "openMP style," but definitely something which is "good to know."

The book ends with considerations and tips regarding the actual building and configuration of a GPU-based system, including such fundamental issues as proper cooling or power supply and a chapter about common problems and their solutions, which mainly includes general thoughts on finding and avoiding errors. Both chapters feel somehow "attached" rather than "included," but still round up the volume nicely.

Key Strengths and Concerns

"If you just want to write code and don't care about performance, parallel programming is actually quite easy." The author notes this fact on p. 43, but he goes the other way in his treatment of the subject: throughout the book, he sets value on understanding the underlying hardware and performance, however, knowing that GPU hardware in particular is rapidly evolving and changing. Consequently, the chapter "Optimizing Your Application" is the largest single chapter in the book.

The chapters are well balanced and mostly build on top of each other. The author touches all relevant aspects of GPU programming, and everything is explained in sufficient detail, mainly without getting bogged down. A useful feature is the connections to CPU programming that are embedded throughout the book, facilitating the reading for "CPU programmers." The book is full of examples and source code, and it's quite up to date. Each chapter ends with brief-but-helpful conclusions wrapping up the information given in the chapter and providing a measure for the reader to check if he's still on the same page as the author.

Overall, this is a nice and enjoyable read. Besides some typesetting flaws, which might bother some readers at times, it is—compared to other programming books—quite refreshing and never dry, even while covering a technically ambitious topic. However, the book could have been useful as a reference as well, if the index were better organized.

I would recommend this book to anyone who is familiar with serial programming and seriously wants to make the transition to massively parallel programming on GPUs, with all its

consequences. The author is doubtlessly an acknowledged expert in the field and left me with the overall question: How can such a complete, detailed, and profound book even be written about such a rapidly changing subject?

However, this book might not be the best choice for everyone interested in GPU programming. If you just want to play around a little to get a feeling for GPU programming, this book might be too complex (but *CUDA by Example*[2] might be worth a look); and if you are already an expert GPU programmer looking for a reference, online resources are probably more useful by now.

References

1. M. Weigel, "The GPU Revolution at Work," *Computing in Science & Eng.*, vol. 13, no. 5, 2011, pp. 5–6.
2. M. Tiglio, "Teaching Yourself CUDA: Learn to Walk While You Run," *Computing in Science & Eng.*, vol. 13, no. 6, 2011, pp. 6–9.

Thomas Vogel is a postdoctoral fellow at Los Alamos National Laboratory. He uses Monte Carlo and molecular dynamics methods to study soft matter systems and material defects. Vogel has a doctorate degree in physics from the University of Leipzig, Germany. Contact him at tvogel@lanl.gov.