as máquinas de hoje. Queremos ajudá-lo a dominar a programação paralela de modo que seus programas possam se expandir até o nível de desempenho das novas gerações de máquinas.

Muito conhecimento técnico será exigido para alcançar esses objetivos, de modo que abordaremos muitos princípios e padrões de programação paralela neste livro. Contudo, não podemos garantir que todos eles serão cobertos, e por isso selecionamos várias das técnicas mais úteis e comprovadas para explicar com detalhes. Para complementar o seu conhecimento e habilidade, incluímos uma lista de literatura recomendada. Agora, estamos prontos para lhe oferecer uma visão geral rápida do restante do livro.

1.6 Organização do livro

O Capítulo 2 revê a história da computação em GPU. Ele começa com um breve resumo da evolução do hardware gráfico em direção à maior facilidade de programação e depois discute o movimento histórico GPGPU. Muitas das características e limitações atuais das GPUs CUDA têm suas raízes nesses desenvolvimentos históricos. Uma boa compreensão desses desenvolvimentos históricos ajudará o leitor a entender melhor o estado atual e as tendências futuras da evolução do hardware que continuará a ter impacto nos tipos de aplicações que se beneficiarão com CUDA.

O Capítulo 3 introduz a programação CUDA. Esse capítulo conta com o fato de que os alunos já tiveram experiência anterior com a programação C. Primeiro, ele introduz CUDA como uma extensão simples e pequena à linguagem C, dando suporte à computação heterogênea conjunta, entre CPU/GPU, bem como ao modelo bastante utilizado de programação paralela *Single-Program, Multiple-Data* (SPMD). Depois ele aborda os processos de pensamento envolvidos em: (1) identificar a parte dos programas de aplicação a serem paralelizados, (2) isolar os dados a serem usados pelo código paralelizado usando uma função da API para alocar memória no device de computação paralela, (3) usar uma função da API para transferir dados para o device de computador paralelo, (4) desenvolver uma função do kernel que será executada por *threads* individuais na parte paralelizada, (5) disparar uma função do kernel para execução pelas *threads* paralelas e, (6) por fim, transferir os dados de volta ao processador (*host*) com uma chamada de função da API. Embora o objetivo do Capítulo 3 seja ensinar conceitos suficientes do modelo de programação CUDA para que os leitores possam escrever um programa CUDA paralelo simples, ele, na realidade, explica várias habilidades básicas necessárias para desenvolver uma aplicação paralela com base em qualquer modelo de programação paralela. Usamos um exemplo contínuo de multiplicação de matriz-matriz para concretizar esse capítulo.

Os Capítulos de 4 a 7 foram elaborados para dar aos leitores um conhecimento mais profundo do modelo de programação CUDA. O Capítulo 4 explica o modelo de organização e execução de *threads* exigido para entender totalmente o comportamento da execução das *threads*, bem como conceitos básicos de desempenho. O Capítulo 5 é dedicado às memórias especiais que podem ser usadas para manter variáveis CUDA a fim de melhorar a velocidade de execução do programa. O Capítulo 6 introduz os principais fatores que contribuem para o desempenho de uma função do kernel CUDA. O Capí-

tulo 7 introduz a representação de ponto flutuante e conceitos como precisão e exatidão. Embora esses capítulos sejam baseados em CUDA, eles ajudam os leitores a acumularem uma base para a programação paralela em geral. Acreditamos que os humanos entendem melhor quando aprendemos de baixo para cima; ou seja, primeiro precisamos aprender os conceitos no contexto de um modelo de programação em particular, o que nos dá uma base sólida para generalizar nosso conhecimento para outros modelos de programação. Ao fazermos isso, podemos nos basear em nossa experiência concreta a partir do modelo CUDA. Uma experiência profunda com esse modelo também nos permite obter maturidade, que nos ajudará a aprender conceitos que podem nem sequer ser pertinentes a ele.

Os Capítulos 8 e 9 são estudos de caso de duas aplicações reais, que fazem os leitores refletirem sobre os processos de pensamento para paralelizar e otimizar suas aplicações a fim de obter ganhos de velocidade significativos. Para cada aplicação, começamos identificando modos alternativos de formular a estrutura básica da execução paralela e seguimos raciocinando a respeito das vantagens e desvantagens de cada alternativa. Depois, percorremos os passos da transformação de código necessária para alcançar um alto desempenho. Esses dois capítulos ajudam os leitores a consolidarem todo o material dos capítulos anteriores e se prepararem para os seus próprios projetos de desenvolvimento de aplicação.

O Capítulo 10 generaliza as técnicas de programação paralela nos princípios de decomposição de problema, estratégias de algoritmo e pensamento computacional. Ele faz isso abordando o conceito de organização das tarefas de computação de um programa de modo que possam ser feitas em paralelo. Começamos discutindo sobre o processo tradutório da organização dos conceitos científicos abstratos para tarefas computacionais, um primeiro passo importante na produção de software de aplicação de qualidade, serial ou paralelo. O capítulo, então, aborda as estruturas do algoritmo paralelo e seus efeitos sobre o desempenho da aplicação, que é baseado na experiência de ajuste de desempenho com CUDA. O capítulo termina com um tratamento dos estilos e modelos de programação paralela, permitindo que os leitores coloquem seu conhecimento em um contexto mais amplo. Com esse capítulo, os leitores podem começar a generalizar a partir do estilo de programação SPMD para outros estilos de programação paralela, como paralelismo de loop em OpenMP e "*fork-join*" na programação *p-thread*. Embora não nos aprofundemos nesses estilos alternativos de programação paralela, esperamos que os leitores sejam capazes de aprender a programar em qualquer um deles com a base obtida neste livro.

O Capítulo 11 introduz o modelo de programação OpenCL do ponto de vista de um programador CUDA. O leitor verá que OpenCL é extremamente semelhante a CUDA. A diferença mais importante está no uso que a OpenCL faz das funções de API para implementar funcionalidades como disparo do kernel e identificação de *thread*. O uso das funções de API torna OpenCL mais complexa de usar; apesar disso, um programador CUDA tem todo o conhecimento e habilidades necessárias para entender e escrever programas OpenCL. De fato, acreditamos que a melhor maneira de ensinar programação OpenCL é ensinar CUDA primeiro. Demonstramos isso com um capítulo que relaciona todos os principais recursos da OpenCL aos seus recursos CUDA correspondentes. Também ilustramos o uso desses recursos adaptando nossos exemplos CUDA simples para OpenCL.

O Capítulo 12 oferece alguns comentários finais e uma visão geral do futuro da programação maciçamente paralela. Revisamos nossos objetivos e resumimos como os capítulos se encaixam para ajudar a atingir a meta. Depois, apresentamos um breve estudo das principais tendências na arquitetura de processadores maciçamente paralelos e como essas tendências provavelmente afetarão a programação paralela no futuro. Concluímos com uma previsão de que esses avanços rápidos na computação maciçamente paralela a tornarão uma das áreas mais empolgantes na próxima década.

Referências e leitura adicional

HWU, W. W., Keutzer, K. & Mattson, T. (2008). The Concurrency Challenge. *IEEE Design and Test of Computers*, julho/agosto, 312–320.

Khronos Group. (2009). *The OpenCL Specification Version 1.0*. Beaverton, OR: Khronos Group. (http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pd).MATTSON, T. G., Sanders, B. A. & Massingill, B. L. (2004). *Patterns of parallel programming*. Upper Saddle River, NJ: Addison-Wesley.

Message Passing Interface Forum. (2009). *MPI: A Message-Passing Interface Standard, Version 2.2*. Knoxville: University of Tennessee. (http://www.forum.org/docs/mpi-2.2/mpi22-report.pdf).

NVIDIA. (2007). *CUDA Programming Guide*. Santa Clara, CA: NVIDIA Corp.

OpenMP Architecture Review Board. (2005). *OpenMP Application Program Interface*. (http://www.openmp.org/mp-documents/spec25.pd). SUTTER, H. & Larus, J. (2005). Software and the concurrency revolution. *ACM Queue*, 3(7), 54–62.

NEUMANN, J von. (1945). *First Draft of a Report on the EDVAC*. Contract No. W-670-ORD-4926, U.S. Army Ordnance Department and University of Pennsylvania (reproduzido em Goldstine H. H. (Ed.), (1972). *The computer: From Pascal to Von Neumann*. Princeton, NJ: Princeton University Press).

WING, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33–35.