

## <<高性能CUDA应用设计与开发>>

### 图书基本信息

书名：<<高性能CUDA应用设计与开发>>

13位ISBN编号：9787111404460

10位ISBN编号：7111404467

出版时间：2013-1-1

出版时间：机械工业出版社华章公司

作者：Rob Farber

页数：271

译者：于玉龙,唐堃

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## <<高性能CUDA应用设计与开发>>

### 前言

译者序序言作为一种革新性的技术，GPU近年来在科学计算领域备受关注。

这项技术已有大量的科学应用，并在性能与能效方面取得了显著提升。

这些应用大多由那些致力于使用GPU的先驱者开发和使用。

最近，这项技术面临一个关键问题：它能否普遍适用于科学计算领域中各种各样的算法，以及能否为更广泛的人群而不仅是那些先驱者使用。

软件开发是阻碍这项技术得以广泛应用的关键，其中包括大规模并行CUDA代码的编写与优化、新的性能与正确性分析工具的使用、CUDA支持库的使用以及对GPU硬件架构的理解。

通过书籍、教程等方式，专家们可以将他们在此领域掌握的知识和方法与其他使用者分享，这在一定程度上解决了CUDA广泛应用的难题。

本书就是这样的一本书。

在这本书中，作者详尽地解释了重要算法的实现方法，如量子化学、机器学习以及计算机视觉等。

本书不仅讲述了GPU编程的基本方法，还介绍了如何改写算法以便从GPU架构中最大化获益。

此外，这本书提供了许多案例研究用以解释与充实GPU的重要概念，如CUDA线程、GPU存储层次结构以及多GPU的扩展性（书中使用一个MPI示例例证了其扩展性可以近似线性地增至使用500个GPU）

最后，任何一种编程语言都不能独立存在。

可以说，任何一个成功的编程语言，都伴随由强大的编译器、性能与正确性分析工具以及优化的支持库组成的整套系统环境。

这些软件开发中的实用工具是快速开发应用程序的关键。

本书的诸章节描述了如何使用CUDA编译器、调试器、性能分析工具、库以及与其他语言的互操作，本书在这些方面没有令人失望。

我享受从这本书中学到的一切，我确信您也会。

Jeffrey S.Vetter美国橡树岭国家实验室杰出研究员佐治亚理工学院教授

## <<高性能CUDA应用设计与开发>>

### 内容概要

本书是广受推崇的系统学习高性能CUDA应用开发与设计的经典著作，是美国国家安全实验室资深高性能编程专家多年工作经验结晶，橡树岭国家实验室资深专家鼎力推荐！本书不仅从硬件角度深入解读了CUDA的设计理念和GPGPU硬件的体系结构，而且从软件角度系统讲解了CUDA应用设计与开发的思想、方法、技巧、准则、注意事项和最佳实践。

第1章首先介绍了CUDA的核心概念和编程思想，以及构建与调试CUDA应用所需的工具和方法，然后讲解了有效提高程序性能的CPU编程准则；第2章讲解了CUDA在机器学习与优化中的核心概念与应用，并给出了完整的通用框架；第3章介绍了CUDA的性能分析工具套件以及性能分析的方法，同时讨论了PCA和NLPCA两种数据挖掘方法；第4章讲解了CUDA的执行模型，深刻揭示了GPU的工作方式和原理；第5章介绍了CUDA提供的多种GPU内存，以及各种内存的优缺点；第6章讲解了高效利用内存的技术；第7章介绍了GPU提供的多种并行方式及其应用；第8章首先讨论了多种CUDA后端设备，以及CUDA如何与Python、Java、R等高级语言交互；第9章讲解了CUDA与图形渲染混合编程；第10章讲解了在云计算和集群环境中使用CUDA的方法和技术细节；第11章介绍了CUDA在高维数据处理、力导向图、交互式 workflows、量子化学等现实问题中的应用；第12章为学习CUDA设计了一个综合性的针对实时视频流的应用案例。

## 作者简介

Rob

Farber, 资深高性能编程专家, 是Irish高端计算中心和美国国家实验室等权威机构的高性能编程技术顾问, 同时为多家《财富》世界500强企业提供咨询服务, 经验十分丰富, 在该领域颇具权威和影响力。

他还是一位技术作家, 任职于Santa

Fe学院, 在《Dr. Dobb ' s Journal》、《Scientific

Computing》等媒体上发表了多篇关于高性能编程的经典技术文章, 深受读者喜爱。

此外, 他还是《财富》排名前100名的两家公司的联合创始人。

## &lt;&lt;高性能CUDA应用设计与开发&gt;&gt;

## 书籍目录

译者序

序言

前言

第1章 CUDA入门与编程思想

1.1 源代码与维基

1.2 一个用以区别CUDA与传统程序开发的示例

1.3 选择合适的CUDA API

1.4 CUDA的一些基本概念

1.5 理解首个Runtime Kernel

1.6 GPGPU编程的三条法则

1.6.1 法则1：将数据放入并始终存储于GPU

1.6.2 法则2：交给GPGPU足够多的任务

1.6.3 法则3：注重GPGPU上的数据重用，以避免带宽限制

1.7 大O记号的思想与数据传输

1.8 CUDA和Amdahl定律

1.9 数据并行与任务并行

1.10 混合执行：同时使用CPU和GPU资源

1.11 回归测试与正确性

1.12 静默错误

1.13 调试简介

1.14 UNIX调试方法

1.14.1 NVIDIA cuda-gdb调试器

1.14.2 CUDA内存检查器

1.14.3 通过UNIX ddd界面使用cuda-gdb

1.15 使用Parallel Nsight进行Windows调试

1.16 本章小结

第2章 CUDA在机器学习与优化中的应用

2.1 建模与模拟

2.1.1 拟合参数化模型

2.1.2 Nelder-Mead方法

2.1.3 Levenberg-Marquardt方法

2.1.4 算法加速

2.2 机器学习与神经网络

2.3 异或逻辑：一个重要的非线性机器学习问题

2.3.1 目标函数示例

2.3.2 针对多GPU设备、多CPU处理器的完整仿函数

2.3.3 完整Nelder-Mead优化代码的简要讨论

2.4 异或逻辑的性能结果

2.5 性能讨论

2.6 本章小结

2.7 C++ NELDER-MEAD代码模板

第3章 CUDA工具套件：对PCA、NLPCA进行性能分析

3.1 PCA和NLPCA

3.1.1 自编码网络

3.1.2 用于PCA分析的仿函数示例

## <<高性能CUDA应用设计与开发>>

- 3.1.3 用于NLPCA分析的示例仿函数
- 3.2 获得基础性能分析数据
- 3.3 gprof：通用UNIX性能分析器
- 3.4 NVIDIA可视化性能分析器：computeprof
- 3.5 Microsoft Visual Studio中的Parallel Nsight
  - 3.5.1 Nsight时间表分析
  - 3.5.2 NVTX跟踪支持库
  - 3.5.3 CUDA API的可扩展性表现
- 3.6 性能调节与分析实用工具（TAU）
- 3.7 本章小结
- 第4章 CUDA执行模型
  - 4.1 GPU架构综述
    - 4.1.1 线程调度：通过执行配置统筹性能与并行度
    - 4.1.2 computeprof中Warp相关值
    - 4.1.3 Warp分歧
    - 4.1.4 关于Warp分歧的若干准则
    - 4.1.5 computeprof中Warp分歧相关值
  - 4.2 Warp调度与TLP
  - 4.3 ILP：高性能低占用率
    - 4.3.1 ILP隐藏算术计算延迟
    - 4.3.2 ILP隐藏数据延迟
    - 4.3.3 ILP的未来
    - 4.3.4 computeprof中指令速率相关值
  - 4.4 Little法则
  - 4.5 检测限制因素的CUDA工具
    - 4.5.1 nvcc编译器
    - 4.5.2 启动约束
    - 4.5.3 反汇编器
    - 4.5.4 PTX Kernel函数
    - 4.5.5 GPU模拟器
  - 4.6 本章小结
- 第5章 CUDA存储器
  - 5.1 CUDA存储器层次结构
  - 5.2 GPU存储器
  - 5.3 L2缓存
  - 5.4 L1缓存
  - 5.5 CUDA内存类型
    - 5.5.1 寄存器
    - 5.5.2 局域内存
    - 5.5.3 和局域内存相关的computeprof性能分析参数
    - 5.5.4 共享内存
    - 5.5.5 和共享内存相关的computeprof性能分析参数
    - 5.5.6 常量内存
    - 5.5.7 纹理内存
    - 5.5.8 和纹理内存相关的computeprof性能分析参数
  - 5.6 全局内存
    - 5.6.1 常见的整合内存示例

## &lt;&lt;高性能CUDA应用设计与开发&gt;&gt;

- 5.6.2 全局内存的申请
- 5.6.3 全局内存设计中的限制因素
- 5.6.4 和全局内存相关的computeProf性能分析参数
- 5.7 本章小结
- 第6章 高效使用CUDA存储器
  - 6.1 归约
    - 6.1.1 归约模板
    - 6.1.2 functionReduce.h的测试程序
    - 6.1.3 测试结果
  - 6.2 使用非规则数据结构
  - 6.3 稀疏矩阵和CUSP支持库
  - 6.4 图论算法
  - 6.5 SoA、AoS以及其他数据结构
  - 6.6 分片和分块
  - 6.7 本章小结
- 第7章 提高并行度的技巧
  - 7.1 CUDA上下文环境对并行度的扩展
  - 7.2 流与上下文环境
    - 7.2.1 多GPU的使用
    - 7.2.2 显式同步
    - 7.2.3 隐式同步
    - 7.2.4 统一虚拟地址空间
    - 7.2.5 一个简单的示例
    - 7.2.6 分析结果
  - 7.3 使用多个流乱序执行
    - 7.3.1 在同一GPU内并发执行Kernel函数的建议
    - 7.3.2 隐式并行Kernel的原子操作
  - 7.4 将数据捆绑计算
    - 7.4.1 手动分割数据
    - 7.4.2 映射内存
    - 7.4.3 映射内存的工作机制
  - 7.5 本章小结
- 第8章 CUDA在所有GPU与CPU程序中的应用
  - 8.1 从CUDA到多种硬件后端的途径
    - 8.1.1 PGI CUDA x86编译器
    - 8.1.2 PGI CUDA x86编译器
    - 8.1.3 将x86处理器核心用作流多处理器
    - 8.1.4 NVIDIA NVCC编译器
    - 8.1.5 Ocelot
    - 8.1.6 Swan
    - 8.1.7 MCUDA
  - 8.2 从其他语言访问CUDA
    - 8.2.1 SWIG
    - 8.2.2 Copperhead
    - 8.2.3 EXCEL
    - 8.2.4 MATLAB
  - 8.3 支持库

## <<高性能CUDA应用设计与开发>>

- 8.3.1 CUBLAS
- 8.3.2 CUFFT
- 8.3.3 MAGMA
- 8.3.4 phiGEMM支持库
- 8.3.5 CURAND
- 8.4 本章小结
- 第9章 CUDA与图形渲染混合编程
  - 9.1 OpenGL
    - 9.1.1 GLUT
    - 9.1.2 通过OpenGL映射GPU内存
    - 9.1.3 使用基元重启提升3D处理性能
  - 9.2 框架内各文件的介绍
    - 9.2.1 Kernel与Perlin Kernel演示的示例代码
    - 9.2.2 simpleGLmain.cpp文件
    - 9.2.3 simpleVBO.cpp文件
    - 9.2.4 callbacksVBO.cpp文件
  - 9.3 本章小结
- 第10章 在云计算和集群环境中使用CUDA
  - 10.1 消息传递接口
    - 10.1.1 MPI编程模型
    - 10.1.2 MPI通信器
    - 10.1.3 MPI进程号
    - 10.1.4 主从模式
    - 10.1.5 点对点模式基础
  - 10.2 MPI通信机制
  - 10.3 带宽
  - 10.4 平衡率
  - 10.5 运行大型MPI程序需要考虑的因素
    - 10.5.1 初始数据加载的可扩展性
    - 10.5.2 使用MPI进行计算
    - 10.5.3 可扩展性检查
  - 10.6 云计算
  - 10.7 代码示例
    - 10.7.1 数据的产生
    - 10.7.2 主体代码部分
  - 10.8 本章小结
- 第11章 CUDA在现实问题中的应用
  - 11.1 高维数据的处理
    - 11.1.1 PCA/NLPCA
    - 11.1.2 多维尺度分析
    - 11.1.3 K均值聚类算法
    - 11.1.4 期望最大化
    - 11.1.5 支持向量机
    - 11.1.6 Bayesian网络
    - 11.1.7 互信息
  - 11.2 力导向图
  - 11.3 Monte Carlo方法

## <<高性能CUDA应用设计与开发>>

- 11.4 分子建模
- 11.5 量子化学
- 11.6 交互式 workflow
- 11.7 其他众多的项目
- 11.8 本章小结
- 第12章 针对现场实况视频流的应用程序
  - 12.1 机器视觉话题
    - 12.1.1 3D效果
    - 12.1.2 肤色区域分割
    - 12.1.3 边缘检测
  - 12.2 FFmpeg
  - 12.3 TCP服务器
  - 12.4 实况视频流应用程序
    - 12.4.1 kernelWave(): 动画Kernel函数
    - 12.4.2 kernelFlat(): 在平面渲染图像
    - 12.4.3 kernelSkin(): 仅保留肤色区域
    - 12.4.4 kernelSobel(): Sobel边缘检测过滤器
    - 12.4.5 launch\_kernel()方法
  - 12.5 simpleVBO.cpp文件
  - 12.6 callbacksVBO.cpp文件
  - 12.7 生成与执行代码
  - 12.8 展望
    - 12.8.1 机器学习
    - 12.8.2 Connectome
  - 12.9 本章小结
  - 12.10 simpleVBO.cpp文件
- 参考文献
- 术语表

## &lt;&lt;高性能CUDA应用设计与开发&gt;&gt;

## 章节摘录

版权页：插图：计算能力为2.0的设备为每个流多处理器增加了L1缓存，并且在流多处理器和全局内存之间增加了一套所有流多处理器都共享的L2缓存，如图5.2所示。

5.3 L2缓存 GPU上所有流多处理器都可访问的L2缓存（二级缓存）是一种高速数据存储设备，可极大地降低设备内存子系统的工作压力。

由于L2缓存的存在，许多程序在计算能力为2.0的设备上的运行速度会变得很快。

这可归结为两个原因：在不需要CUDA编程者做任何调整的情况下，L2缓存可通过LRU方式缓存数据，这使得很多CUDA Kernel避免了全局内存的带宽瓶颈。

非规则内存访问会表现出极差的性能，而L2缓存对非规则内存访问具有明显的加速效果。

对于很多算法，程序自身的特性决定了其能否有效地部署在不具备L2缓存的计算能力为1.x的设备上。

Fermi架构的GPU提供了768KB共享的L2缓存，并保证了在各流多处理器间的一致性。

也就是说，任何线程都可以对L2缓存中的内容进行修改。

然后，任何其他GPU线程都可以读取该地址，得到更新后的正确数据。

当然，必须使用原子操作才能保证当前数据写操作完成后再允许其他线程的读取操作。

因为早期GPGPU架构采用了不同的处理路径来处理数据的读操作和写操作，所以即使最普通的读写操作也存在执行上的风险。

特别是只读的纹理加载路径和只写的像素数据路径。

为了保证数据正确性，在早期GPU架构中，如果任何线程修改了一个被缓存的内存区域的数据，都将导致读操作路径上该内存区域的缓存数据无效，并清空已存在的缓存数据。

在Fermi架构中，流多处理器间共享的L2缓存消除了早期GPU架构中纹理和光栅缓存中存在的这一性能瓶颈。

所有的数据加载与存储操作都经过L2缓存（包括CPU与GPU之间的数据拷贝）。

这就解释了为什么GPU和主机间的数据传输居然会影响缓存命中率和程序性能。

显然，异步的Kernel执行也会造成缓存污染。

## <<高性能CUDA应用设计与开发>>

### 媒体关注与评论

本书是所有希望系统学习和使用CUDA编写用于科学计算或可视化程序等并行应用的程序员的必读书之一。

本书通俗易懂，并且提供了便于读者动手实践的详细操作案例。

—— Jack Dongarra 田纳西大学计算机科学分布式处理器研究院

## <<高性能CUDA应用设计与开发>>

### 编辑推荐

《高性能CUDA应用设计与开发:方法与最佳实践》编辑推荐：美国国家安全实验室资深专家撰写，硬件角度解读CUDA设计理念与体系结构，软件角度讲解CUDA应用开发思想、方法、技巧与准则。

## <<高性能CUDA应用设计与开发>>

### 名人推荐

本书是所有希望系统学习和使用CUDA编写用于科学计算或可视化程序等并行应用的程序员的必读书之一。

本书通俗易懂，并且提供了便于读者动手实践的详细操作案例。

——Jack Dongra 田纳西大学计算机科学分布式处理器研究院

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>