HZ BOOKS
华章IT

国内顶尖的并行计算领域知名专家风辰多年实践经验总结，兼具深度和高度

简洁明了，辅以大量示例，全面介绍主流硬件平台上的向量化库，并行编程语言的设计细节，OpenCL程序在硬件平台的映射与执行，并给出多个性能优化实战案例

高性能计算系列丛书

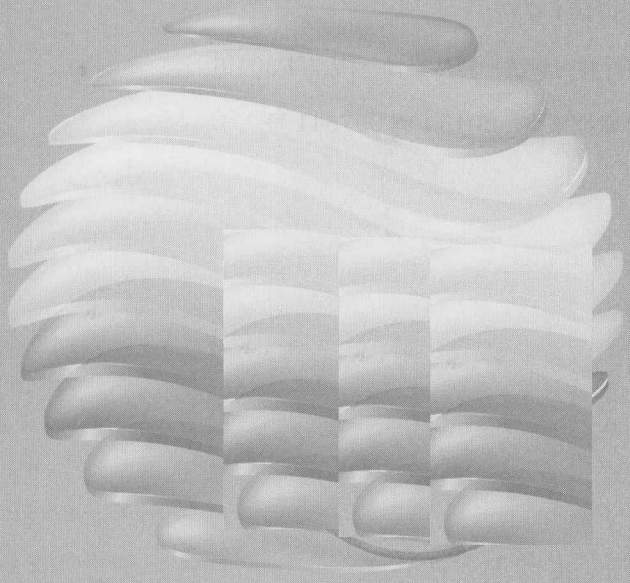
Parallel Programming Methodology and Optimization in Action

并行编程方法与优化实践

刘文志◎著

机械工业出版社
China Machine Press

系列丛书



Parallel Programming Methodology and Optimization in Action

并行编程方法与优化实践

刘文志◎著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

并行编程方法与优化实践 / 刘文志著 . —北京：机械工业出版社，2015.6
（高性能计算技术丛书）

ISBN 978-7-111-50194-7

I. 并… II. 刘… III. 并行算法－算法设计 IV. TP301.6

中国版本图书馆 CIP 数据核字（2015）第 097641 号

并行编程方法与优化实践

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：高婧雅 责任校对：殷 虹
印 刷：北京市荣盛彩色印刷有限公司 版 次：2015 年 6 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：14.25
书 号：ISBN 978-7-111-50194-7 定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

Preface 前　言

为什么要写这本书

本书主要是为软件工程师写的。在解释为什么笔者认为软件工程师需要这本书之前，先来介绍并行、并发和代码性能优化这 3 个概念，理解这 3 个概念是阅读本书的基础。

- 并行对应的英文单词是 parallelism，是指在具有多个处理单元的系统上，通过将计算或数据划分为多个部分，然后将各个部分分配到不同的处理单元上，各处理单元相互协作，同时运行，以达到加快求解速度或者扩大求解问题规模的目的。
- 并发对应的英文单词是 concurrency，是指在一个处理单元上运行多个应用，各应用分时占用处理单元。这是一种微观上串行、宏观上并行的模式，有时也称其为时间上串行、空间上并行。
- 代码性能优化是指通过调整源代码，使其生成的机器指令能够更高效地执行。通常高效是指执行时间更少、使用的存储器更少或能够计算更大规模的问题。

从大的方面来说，并行和并发都是代码性能优化的一种方式。但是今天并行和并发已经变得如此重要，以至于需要“开宗立派”。为了划清并行、并发和代码性能优化的界线，在本书中，代码性能优化特指除并行和并发以外的代码优化方法，比如向量化和提高指令流水线效率。在本书中，笔者将向量化独立出来解说。

2003 年以前，在摩尔定律的作用下，单核标量处理器的性能持续提升，软件开发人员只需要写好软件，而性能的提升则等待下次硬件的更新。2003 年之前的几十年里，这种“免费午餐”的模式一直在持续。2003 年后，主要由于功耗的原因，这种“免费午餐”已经不复存在了。为了生存，各硬件生产商不得不采用各种方式提高硬件的计算能力。目前最流行的 3 种方式如下：

1）让处理器在一个周期处理多条指令，多条指令可相同可不同。如 Intel Haswell 处理器

一个周期可执行 4 条整数加法指令、2 条浮点乘加指令，访存和运算指令也可同时执行。

2）使用向量指令，主要是 SIMD 和 VLIW 技术。SIMD 技术将处理器一次能够处理的数据位数从字长扩大到 128 位或 256 位，从而提升了计算能力。

3）在同一个芯片中集成多个处理单元，根据集成方式的不同，相应地称为多核处理器或多路处理器。多核处理器是如此的重要，以至于现在即使是手机上的嵌入式 ARM 处理器都已经是四核或八核的了。

目前绝大部分应用软件都是串行的，串行执行过程符合人类的思维习惯，易于理解、分析和验证。由于串行软件只能在多核处理器中的一个核上运行，和 2003 年以前的 CPU 没有多少区别，这意味着花多核处理器的价钱买到了单核的性能。通过多核技术，硬件生产商成功地将提高实际计算能力的任务转嫁给了软件开发人员，而软件开发人员则没有选择，只有直面挑战。

标量单核的计算能力没有办法持续大幅度提升，而应用对硬件计算能力的需求依旧在提升，这是个实实在在的矛盾。在可见的将来，要解决这个矛盾，软件开发人员只有选择代码优化和并行。代码优化并不能利用多核 CPU 的全部计算能力，它也不要求软件开发人员掌握并行开发技术，另外通常也无须对软件架构做改动，而且串行代码优化有时能够获得非常好的性能（如果原来的代码写得很差的话）。因此相比采用并行技术，应当优先选择串行代码优化。一般来说，采用并行技术获得的性能加速不超过核数，这是一个非常大的限制，因为目前 CPU 硬件生产商最多只能集成几十个核。

从 2006 年开始，可编程的 GPU 越来越为大众所认可。GPU 是图形处理单元（Graphics Processing Unit）的简称，最初主要用于图形渲染。自 20 世纪 90 年代开始，NVIDIA、AMD（ATI）等 GPU 生产商对硬件和软件加以改进，GPU 的可编程能力不断提高，GPGPU（General-Purpose computing on Graphics Processing Units）比以前容易许多。另外，由于 GPU 具有比 CPU 更强大的峰值计算能力，引起了许多科研人员和企业的兴趣。

近两三年来，在互联网企业中，GPU 和并行计算越来越受到重视。无论是国外的 Google、Facebook，还是国内的百度、腾讯、阿里和 360，都在使用代码优化、并行计算和 GPU 来完成以前不能完成的任务。

10 年前，并行计算还是大实验室里教授们的研究对象，而今天，多核处理器和 GPU 的普及已经使得普通人就可以研究它们。对于软件开发人员来说，如果不掌握并行计算和代码性能优化技术，在不久的将来就会被淘汰。

为了学习如何在 X86 和 ARM 平台上向量化代码，软件开发人员需要了解 Intel 和 ARM 提供的参考资料，它们通常细节丰富，开发人员需要从多如牛毛的细节上（而且通常没有示例）了解其概要，可谓难上加难。而本书通过丰富的简单示例，让读者能够从全局上把握这些扩展的特性。

NVIDIA 的 CUDA 和开放的 OpenCL 标准越来越得到大家的重视，OpenCL 和 CUDA 在概念上有太多相似的地方，在本书中，笔者尝试将它们放在一起描述，以便于读者理解。

为了帮助读者理解，本书使用了大量的示例。开发人员通常比较忙，因此本书力求简洁明了，点到为止。

读者对象

由于多核处理器和 GPU 已经非常便宜，而代码优化、向量化和并行已经深入 IT 行业的骨髓，所有 IT 行业的从业者都应当阅读本书。如果非要列一个清单，笔者认为下列人员应当阅读：

- 互联网及传统行业的 IT 从业者，尤其是希望将应用移植到多核向量处理器或 GPU 上的开发人员
- 对向量化和并行化感兴趣的专业工作者
- 高等院校、研究所的学生及教师

如何阅读本书

本系列包括 3 本书，本书是此系列的第二本[⊖]。本书的重点在于介绍如何利用目前主流的 C 语言的各种特定硬件或平台的向量化扩展、并行化库，来编写和优化串行代码的性能。而本系列的第一本《并行算法设计与性能优化》则关注并行优化和并行计算相关的理论、算法设计及高层次的实践经验。本书同时关注 C 程序设计语言的向量化和并行化扩展，以及算法到硬件的映射。

本书不但包括如何使用 SSE/AVX 向量化扩展、如何使用 OpenMP 编译制导语句优化运行在 X86 多核处理器上的代码的性能，还包括如何使用 NEON 向量化扩展、OpenMP 编译制导语句优化运行在移动处理器（ARM）上的代码性能优化，以及使用 CUDA 和 OpenCL 优化运行在图形处理器（GPU）上的代码性能优化及并行。不但有实际的各个扩展解析或语言的介绍说明，还有丰富的算法优化实例。作者希望通过这种方式让阅读本书的软件开发人员了解、掌握常见的向量化库、并行编程语言，学会如何使用这些语言或库，将常见算法映射到具体硬件上以获得高性能，以及这些库和语言的优缺点。

整体而言，本书分为如下几个部分：

- **代码向量化优化**，主要介绍常见的 C 语言的向量化库，主要是 X86 平台和 ARM 平台

⊖ 除本书之外，另两本书分别是《并行算法设计与性能优化》和《科学计算与企业级应用的并行优化》，均由机械工业出版社华章公司（www.hzbook.com）出版。——编者注

的向量化扩展。主要内容见第 1 章和第 2 章。

- **并行程序设计语言**，主要介绍目前主流的并行程序设计语言 OpenMP、CUDA、OpenCL 及 OpenACC。主要内容见第 3 ～ 5 章。
- **主流的向量化和并行化硬件平台，以及 OpenCL 程序如何映射到这些平台上**。主要介绍 Intel Haswell、ARM A15、AMD GCN GPU 和 NVIDIA Kepler/Maxwell GPU 的架构，及 OpenCL 程序如何在这些硬件上执行。主要内容见第 6 章。
- **常见简单应用的向量化和并行化**，主要介绍如何使用前面提到的向量化扩展和并行语言来优化图像处理、线性代数应用的性能。主要内容见第 7 章和第 8 章。

第 1 章 主要介绍 Intel SSE/AVX 的 C 语言向量化扩展（内置函数）的相关细节，如支持哪些操作，使用 AVX/SSE 向量化时需要注意哪些问题。另外还介绍了如何在 Intel X86 上测得峰值，及如何使用 SSE/AVX 优化一些实例代码（这些示例由高洋提供）。

第 2 章 介绍了 ARM A15 高性能处理器的特性，及 ARM 提供的向量化扩展（内置函数）NEON 的细节。另外介绍了如何在 ARM A15 处理器上优化彩色图转灰度图、矩阵转置和矩阵乘法等。

第 3 章 介绍了 OpenMP 程序设计的细节内容。主要包括 OpenMP 的环境变量、函数和编译制导语句。本章提供了许多 OpenMP 编译制导语句构造和子句的编程实例，以帮助读者理解。本章还简单介绍了 OpenMP 4.0 标准引入的异构并行计算的内容。并以如何使用 OpenMP 计算圆周率为例，介绍了 OpenMP 的互斥同步支持，并比较了这些不同同步方式的性能。

第 4 章 介绍了目前用于异构并行计算平台（主要是 GPU）的 CUDA 和 OpenCL 的细节。首先介绍了基于 GPU 的异构并行计算的历史与现状，及 GPU 和 CPU 的优缺点。然后介绍了 CUDA 和 OpenCL 编程的细节，并展示了如何使用 CUDA 和 OpenCL 来计算圆周率。接着介绍了如何基于 GPU 优化 CUDA 和 OpenCL 程序性能。最后以矩阵转置和矩阵乘法为例介绍了如何在 GPU 上优化程序性能。

第 5 章 简单介绍了用于异构平台的 OpenACC 标准。介绍了常见的 OpenACC 编译制导语句，然后介绍了 CUDA 和 OpenACC 如何通过共享存储器指针实现通信。最后以一个简单的迭代算法为例，介绍了如何优化 OpenACC 程序。

第 6 章 介绍了常见的并行编程硬件平台的架构及 OpenCL 程序如何映射到这些平台上执行。简略介绍了 Intel Haswell、ARM A15、AMD GCN GPU 和 NVIDIA Kepler/Maxwell GPU 的架构。然后介绍了 OpenCL 程序如何映射到 Intel Haswell 处理器、AMD GCN GPU 和 NVIDIA Kepler/Maxwell GPU 上执行。

第 7 章 详细介绍了如何使用 SSE/AVX、ARM NEON、OpenMP 和 CUDA 来优化常见的图像处理应用，比如均值滤波和中值滤波、图像直方图和曼德勃罗集。

第 8 章 详细介绍了如何使用 SSE/AVX、ARM NEON、OpenMP 和 CUDA 来优化两向量距离和稠密矩阵向量乘法。

本书希望通过这种方式让读者渐进地、踏实地拥有并行思维，了解各个向量化和并行化编程库的优缺点，并且能够写出优良的向量化、并行代码。

对并行和代码优化不太了解的人员，笔者希望你们按章节顺序仔细阅读本书；对并行或代码优化非常了解的人员，可按照需求选择章节阅读。

勘误和支持

笔者的水平有限，工作繁忙，编写时间仓促，而向量化、并行和代码优化又是一个正在高速发展、与硬件及算法密切相关、影响因素非常多、博大精深、又具有个人特色的领域，许多问题还没有统一的解决方案。虽然笔者已经努力确认很多细节，但书中难免会出现一些不准确的地方，甚至是错误，恳请读者批评指正。你可以将书中的错误或写得不好的地方通过邮件发送到 ly152832912@163.com，或微信联系“风辰”，以便再版时修正，笔者会尽快回复邮件。如果你有更多的宝贵意见，也欢迎发送邮件，期待能够得到你们的真挚反馈。

致谢

首先要感谢我的老婆，她改变了我的人生轨迹，让我意识到人生有如此多的乐趣。

感谢中国地质大学（武汉）图书馆，那是使我对并行计算产生兴趣的地方。感谢中国科学院研究生院和中国科学院图书馆，那里为我奠定了从事并行计算事业的基础。

感谢我的朋友陈实富、赖俊杰、高洋、李修宇等，如果没有你们，我会需要更多时间来提升水平。感谢我的领导王鹏、吴韧和汤晓欧，在这些“技术大佬”和“人生赢家”的指导下，我才会成长得如此迅速。

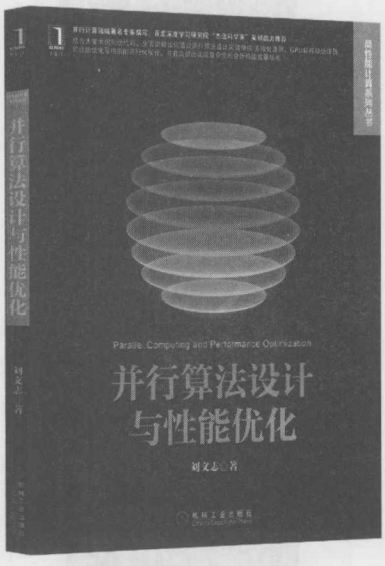
感谢机械工业出版社华章公司的高婧雅和杨福川，我本无意出版此书，是你们鼓励我将它付梓成书；是你们帮我修改书稿，让它变得可读可理解；是你们帮我修正错误，是你们的鼓励和帮助使得我顺利完成全部书稿。

最后感谢我的爸爸、妈妈、姥姥、姥爷、奶奶、爷爷，感谢你们将我培养成人，并时时刻刻为我提供精神力量！

谨以此书献给我最爱的家人，以及众多热爱代码优化、向量化、并行计算的朋友们！愿你们快乐地阅读本书！

风 辰

推荐阅读



并行算法设计与性能优化

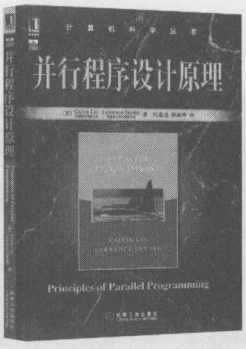
作者：刘文志 ISBN：978-7-111-50102-2 定价：59.00元

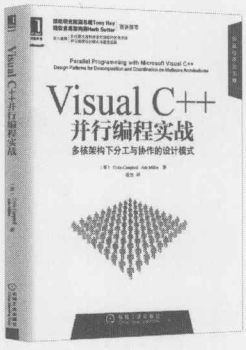
并行计算领域著名专家撰写，百度深度学习研究院“杰出科学家”吴韧鼎力推荐

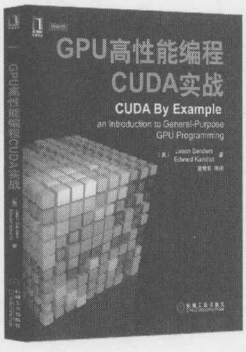
结合大量示例和伪代码，全面讲解如何通过并行算法设计实现单核/多核处理器、GPU和移动处理器的性能优化，以及相关的并行化秘技，并首次提出实现复杂度的全新性能度量标准。

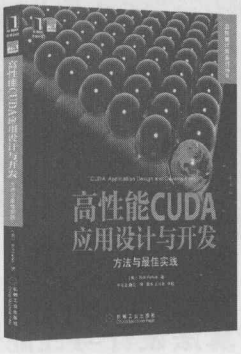
推荐阅读

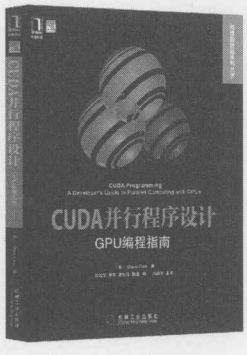

并行程序设计导论
An Introduction to Parallel Programming

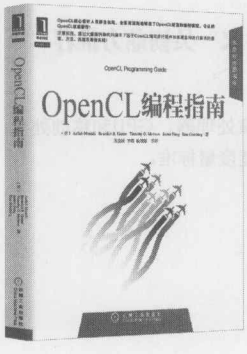

并行程序设计原理
Principles of Parallel Programming

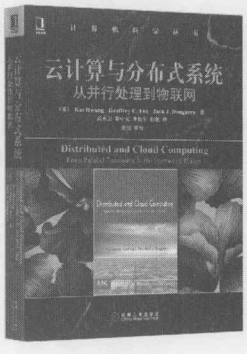

Visual C++
并行编程实战
多核架构下分工与协作的设计模式


GPU高性能编程
CUDA实战
CUDA By Example
an Introduction to General-Purpose
GPU Programming


高性能CUDA
应用设计与开发
方法与最佳实践


CUDA并行程序设计
GPU编程指南


OpenCL编程指南


云计算与分布式系统
从并行处理到物联网
Distributed and Cloud Computing


大规模分布式存储系统
原理解析与架构实战

目　录 *Contents*

前　言

第 1 章　X86 SSE/AVX 指令集 ····· 1

1.1　SSE 内置函数 ····· 2

1.1.1　算术运算 ····· 2

1.1.2　逻辑运算 ····· 5

1.1.3　比较 ····· 5

1.1.4　加载和存储 ····· 6

1.2　AVX 内置函数 ····· 8

1.2.1　算术运算 ····· 8

1.2.2　逻辑运算 ····· 10

1.2.3　比较 ····· 10

1.2.4　加载和存储 ····· 10

1.3　优化实例及分析 ····· 11

1.3.1　如何测得 CPU 的浮点峰值性能 ····· 11

1.3.2　积分计算圆周率 π ····· 14

1.3.3　稀疏矩阵向量乘法 ····· 16

1.3.4　二维单通道图像离散卷积 ····· 19

1.4　本章小结 ····· 22

第 2 章　ARM NEON SIMD 指令优化 ····· 23

2.1　NEON 指令集综述 ····· 23

2.2　ARM A15 处理器性能 ····· 25

2.3　NEON 支持的操作 ····· 25

2.3.1　基本算术运算 ····· 26

2.3.2　基本比较运算 ····· 28

2.3.3　基本数据类型转换及舍入运算 ····· 29

2.3.4　基本位运算 ····· 30

2.3.5　基本逻辑运算 ····· 30

2.3.6　基本设置加载存储操作 ····· 31

2.3.7　特殊操作 ····· 32

2.4　应用实例 ····· 33

2.4.1　彩色图像转灰度图像 ····· 33

2.4.2　矩阵转置 ····· 37

2.4.3　矩阵乘 ····· 39

2.5　本章小结 ····· 42

第 3 章　OpenMP 程序设计 ····· 43

3.1　OpenMP 编程模型 ····· 44

3.1.1　OpenMP 执行模型 ····· 44

3.1.2　OpenMP 存储器模型 ····· 45

3.2　环境变量 ····· 46

3.3　函数 ····· 46

3.3.1　普通函数 ····· 47

3.3.2　锁函数 ····· 48

3.4　OpenMP 编译制导语句 ····· 49

3.4.1 常用的 OpenMP 构造 …… 49
3.4.2 常用的 OpenMP 子句 …… 59
3.5 OpenMP 异构并行计算 …… 65
3.6 OpenMP 程序优化 …… 66
3.6.1 OpenMP 程序优化准则 …… 66
3.6.2 OpenMP 并行优化实例 …… 67
3.7 本章小结 …… 71

第 4 章 基于 GPU 的异构并行计算环境：CUDA 与 OpenCL …… 72

4.1 GPU 计算概述 …… 73
4.1.1 GPU 计算历史 …… 75
4.1.2 CUDA 概述 …… 76
4.1.3 OpenCL 概述 …… 77
4.2 异构并行计算模型 …… 78
4.2.1 平台模型 …… 79
4.2.2 执行模型 …… 80
4.2.3 存储器模型 …… 83
4.2.4 编程模型 …… 85
4.3 C 语言接口 …… 86
4.3.1 OpenCL C 语言 …… 86
4.3.2 CUDA C 语言 …… 108
4.4 基于 GPU 的异构并行计算性能优化 …… 122
4.4.1 总体优化准则 …… 123
4.4.2 全局存储器优化 …… 125
4.4.3 合并访问 …… 125
4.4.4 局部存储器 …… 127
4.4.5 存储体冲突 …… 127
4.4.6 常量存储器优化 …… 128
4.4.7 CUDA 纹理存储器优化 …… 129
4.4.8 寄存器及私有存储器优化 …… 130
4.4.9 工作组数目及大小 …… 131
4.4.10 占用率 …… 132
4.4.11 指令优化 …… 133
4.4.12 分支优化 …… 133
4.4.13 数据传输优化 …… 134
4.5 GPU 与 CPU 精度差别 …… 136
4.6 矩阵转置 …… 137
4.6.1 初次实现 …… 137
4.6.2 满足合并访问的实现 …… 137
4.6.3 没有存储体冲突的实现 …… 138
4.7 矩阵乘法 …… 139
4.7.1 初次实现 …… 140
4.7.2 矩阵分块实现 …… 140
4.8 本章小结 …… 141

第 5 章 OpenACC …… 143

5.1 OpenACC 编程模型 …… 143
5.1.1 执行模型 …… 144
5.1.2 存储器模型 …… 145
5.2 编译制导语句 …… 146
5.2.1 kernels 构造 …… 147
5.2.2 parallel 构造 …… 147
5.2.3 线程配置相关子句 …… 148
5.2.4 data 构造 …… 148
5.2.5 loop 构造 …… 150
5.2.6 atomic 构造 …… 151
5.2.7 dtype 子句 …… 151
5.2.8 reduction 子句 …… 152
5.2.9 变量可见性子句 …… 152
5.2.10 if 子句 …… 152
5.2.11 async 和 wait …… 153
5.3 OpenACC 和 CUDA 协作 …… 153
5.3.1 CUDA 使用 OpenACC 生产的数据 …… 153

5.3.2 OpenACC 使用 CUDA 生产的数据 …… 155
5.4 两小时性能提升 10 倍 …… 156
5.5 本章小结 …… 158

第 6 章 多核向量处理器架构及 OpenCL 程序映射 …… 159

6.1 多核向量处理器架构 …… 159
6.1.1 Intel Haswell CPU 架构 …… 160
6.1.2 ARM A15 多核向量处理器架构 …… 163
6.1.3 AMD GCN GPU 架构 …… 164
6.1.4 NVIDIA Kepler 和 Maxwell GPU 架构 …… 166
6.2 OpenCL 程序在多核向量处理器上的映射 …… 170
6.2.1 OpenCL 程序在多核向量 CPU 上的映射 …… 170
6.2.2 OpenCL 程序在 NVIDIA GPU 上的映射 …… 171
6.2.3 OpenCL 程序在 AMD GCN 上的映射 …… 174
6.3 本章小结 …… 177

第 7 章 利用多种技术优化图像处理中的算法性能 …… 178

7.1 图像滤波 …… 178
7.1.1 均值滤波 …… 178
7.1.2 中值滤波 …… 184
7.2 图像直方图 …… 189
7.2.1 OpenMP 实现 …… 189
7.2.2 CUDA 实现 …… 190
7.3 曼德勃罗集 …… 195
7.3.1 串行算法 …… 195
7.3.2 不适合进行向量化 …… 196
7.3.3 OpenMP 实现 …… 196
7.3.4 CUDA 实现 …… 197
7.4 本章小结 …… 197

第 8 章 利用多种技术优化线性代数中的算法性能 …… 198

8.1 两向量距离 …… 198
8.1.1 串行代码 …… 198
8.1.2 循环展开代码 …… 199
8.1.3 AVX 指令加速 …… 200
8.1.4 NEON 实现 …… 201
8.1.5 CUDA 实现 …… 203
8.2 稠密矩阵与向量乘法 …… 205
8.2.1 串行算法 …… 205
8.2.2 AVX 指令加速 …… 205
8.2.3 NEON 实现 …… 207
8.2.4 CUDA 实现 …… 208
8.2.5 OpenMP 实现 …… 214
8.3 本章小结 …… 216

第 1 章 Chapter 1

X86 SSE/AVX 指令集

SSE/AVX 是 Intel 公司设计的、对其 X86 体系的 SIMD 扩展指令集，它基于 SIMD 向量化技术，提高了 X86 硬件的计算能力，增强了 X86 多核向量处理器的图像和视频处理能力。

SSE/AVX 指令支持向量化数据并行，一个指令可以同时对多个数据进行操作，同时操作的数据个数由向量寄存器的长度和数据类型共同决定。例如，SSE4 向量寄存器（xmm）长度为 128 位，即 16 个字节，如果操作 float 或 int 数据，可同时操作 4 个，如果操作 char 数据，可同时操作 16 个。而 AVX 向量寄存器（ymm）长度为 256 位，即 32 字节，如果操作 char 类型数据，可同时操作 32 个，潜在地大幅度提升程序性能。

虽然 SSE4/AVX 指令向量寄存器的长度为 128/256 位，但是同样支持 64 位长度的向量操作，64 位向量映射到向量寄存器的前 64 位，这保证了兼容性。在 64 位程序下，SSE4/AVX 向量寄存器的个数是 16。

通常 SSE 指令要求访问时内存地址对齐到向量长度，主要是为了减少内存或缓存操作的次数，如果内存地址没有对齐到向量长度，则会增加访问存储器的次数。SSE4 指令要求存储器地址必须 16 字节对齐，而 AVX 指令最好也 32 字节对齐。SSE4 及以前的 SSE 指令不支持不对齐的读写操作，为了简化编程和扩大应用面，AVX 指令集支持不对齐的读写，但是性能大约会下降 20%。为了性能，在可能的情况下，还是应当使用对齐的读写操作。

SSE4/AVX 加入了 stream 的概念，意为不使用缓存的读写，因为不使用缓存则留给其他读取操作的缓存就多些，可能会提高性能。实际上这可能是一个“搬石头砸自己脚”的技术，使用不好的话，可能会对程序性能有负面的影响。笔者并不是不建议使用它，而是让读者明白，使用它可能会适得其反，因此要特别注意。

Intel ICC 和开源的 GCC 编译器支持的 SSE/AVX 指令的 C 接口（intrinsic，内置函数）声明在 intrinsic.h 头文件中。其数据类型命名主要有 __m128/__m256、__m128d/__m256i，默认为单精度（d 表示双精度，i 表示整型）。其函数的命名可大致分为 3 个使用“_”隔开的部分，3 个部分的含义如下。

- 第一个部分为 _mm 或 _mm256。_mm 表示其为 SSE 指令，操作的向量长度为 64 位或 128 位。_mm256 表示 AVX 指令，操作的向量长度为 256 位。本节只介绍 128 位的 SSE 指令和 256 位的 AVX 指令。
- 第二个部分为操作函数名称，如 _add、_load、mul 等，一些函数操作会增加修饰符，如 loadu 表示不对齐到向量长度的存储器访问。
- 第三个部分为操作的对象名及数据类型，_ps 表示操作向量中所有的单精度数据；_pd 表示操作向量中所有的双精度数据；_pixx 表示操作向量中所有的 xx 位的有符号整型数据，向量寄存器长度为 64 位；_epixx 表示操作向量中所有的 xx 位的有符号整型数据，向量寄存器长度为 128 位；_epuxx 表示操作向量中所有的 xx 位的无符号整型数据，向量寄存器长度为 128 位；_ss 表示只操作向量中第一个单精度数据；si128 表示操作向量寄存器中的第一个 128 位有符号整型。

3 个部分组合起来，就形成了一条向量函数，如 _mm256_add_ps 表示使用 256 位向量寄存器执行单精度浮点加法运算。

由于使用指令级数据并行，因此其粒度非常小，需要使用细粒度的并行算法设计。SSE/AVX 指令集对分支的处理能力非常差，而从向量中抽取某些元素数据的代价又非常大，因此不适合含有复杂逻辑的运算。

1.1 SSE 内置函数

本节列出 SSE4 内置函数支持的运算，由于其命名和普通的 C 函数一致，因此不会详细解释其含义。为了避免增加不必要的细节，本节不包含 commit 指令和类型转换指令，感兴趣的读者可以参考 Intel 官方手册。

1.1.1 算术运算

SSE 内置函数中支持的算术操作如表 1-1 所示。从表中可以看出，SSE 几乎支持所有常用的算术运算。故可以预知：大多数算法只要满足向量化数据并行的特点，就有可能能够使用 SSE 指令进行向量化。

表 1-1　SSE 算术运算

操　作	数　据	描　述
add	ss ps epi8 epi16 epi32 epi64 sd pd	加
hadd	pd ps epi16 epi32	相邻数据相加
sub	ss ps epi8 epi16 epi32 epi64 sd pd	减
hsub	pd ps epi16 epi32	相邻数据相减
addsub	ps pd	偶数索引减，奇数索引加
mul	ss ps epi32 epu32 sd pd	乘
mulhi	epi16 epu16	取乘法结果的高位
mullo	epi16 epi32	取乘法结果的低位
div	ss ps sd pd	除
max	ss ps epi16 epu8 sd pd epi8 epi32 epu32 epu16	最大值
min	ss ps epi16 epu8 sd pd epi8 epi32 epu32 epu16	最小值
minpos	epu16	返回最小值及其索引
rsqrt	ss ps	开方的倒数
sqrt	ss ps sd pd	开方
ceil	pd ps sd ss	向上取整
floor	pd ps sd ss	向下取整
abs	epi8 epi16 epi32	求绝对值
avg	epu8 epu16	求均值
dp	pd ps	依据 mask 做乘法
blend	pd ps epi16	类似 C 中的三元运算符
blendv	pd ps epi8	类似 C 中的三元运算符
sign	epi8 epi16 epi32	依据参数改变参数符号
sad	epu8	计算差的绝对值
round	pd ps sd ss	舍入
rcp	ps ss	求倒数
popcnt	u32 u64	求二进制数中为 1 的位数

表 1-1 中有一些不太常见的指令，本节简要介绍一些。

hadd 表示将一个向量相邻的两个元素相加，并要保持原来的向量大小。因此它具有两个参数，如下例所示：

```
__m128 _mm_hadd_ps(__m128 a, __m128 b);
r[0] = a[0]+a[1];
r[1] = a[2]+a[3];
```

```
r[2] = b[0]+b[1];
r[3] = b[2]+b[3];

__m128 _mm_addsub_ps(__m128 a, __m128 b);
r[0] = a[0]-a[1];
r[1] = a[2]+a[3];
r[2] = b[0]-b[1];
r[3] = b[2]+b[3];
```

其中：r 表示结果，使用数组表示法就是表示向量中的第几个元素，如 [2] 表示其为向量寄存器中保存的第三个元素。

使用 hadd 能够比较容易地实现多个向量求内积，如代码清单 1-1 所示。

代码清单1-1 使用hadd求向量内积

```
__m128 a = _mm_mul_ps(b, c);
__m128 zero = _mm_setzero_ps();
a = _mm_hadd_ps(a, zero);
a = _mm_hadd_ps(a, zero);
```

运算完成后，a 中的第一个元素即为 b、c 两个向量的内积。

其实 addsub 应该写作 subadd，因为第一个操作是减法。hsub 和 hadd 类似，只是执行的运算是减法。

比较有意思的是 dp 操作，对于单精度浮点类型的数据，其定义如下：

```
__m128 _mm_dp_ps(__m128 a, __m128 b, const int mask);
__m128 tmp;
tmp[0] = (mask[4]==1) ? (a[0]*b[0]) : 0.0;
tmp[1] = (mask[5]==1) ? (a[1]*b[1]) : 0.0;
tmp[2] = (mask[6]==1) ? (a[2]*b[2]) : 0.0;
tmp[3] = (mask[7]==1) ? (a[3]*b[3]) : 0.0;
__mm32 tmp4
tmp4 = tmp[0] + tmp[1] + tmp[2] + tmp[3];
r[0] = (mask[0]==1) ? tmp4 : 0.0;
r[1] = (mask[1]==1) ? tmp4 : 0.0;
r[2] = (mask[2]==1) ? tmp4 : 0.0;
r[3] = (mask[3]==1) ? tmp4 : 0.0;
```

很明显，一条 dp 操作可以直接实现两个向量的内积运算，如代码清单 1-2 所示。

代码清单1-2 使用dp操作实现两个向量的内积运算

```
int mask = 1+(1<<4)+(1<<5)+(1<<6)+(1<<7);
__m128 r = _mm_dp_ps(b, c, mask);
```

blend 和 blendv 操作类似于 C/C++ 中的三目运算符（?:），blend 依据掩码的位选择，而 blendv 依据掩码向量中各元素的符号位选择，示例如下：

```
__m128 _mm_blend_ps(__m128 a, __m128 b, const int mask);
```