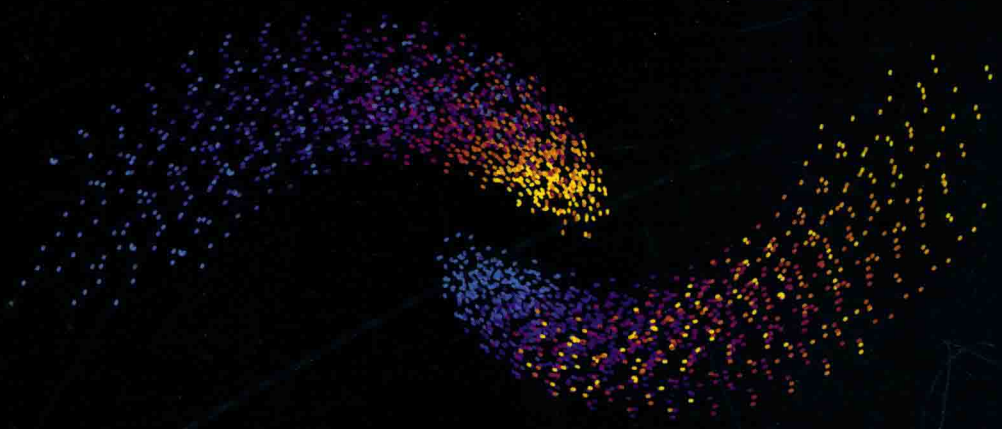
HZ BOOKS
华章IT

ELSEVIER

快速学习使用GPU加速MATLAB代码的全面指南
囊括各类工具箱和函数，以及不同领域的实践案例
灵活调用CUDA加速库，无须用其他语言重写代码

高性能计算技术丛书



GPU Programming in MATLAB

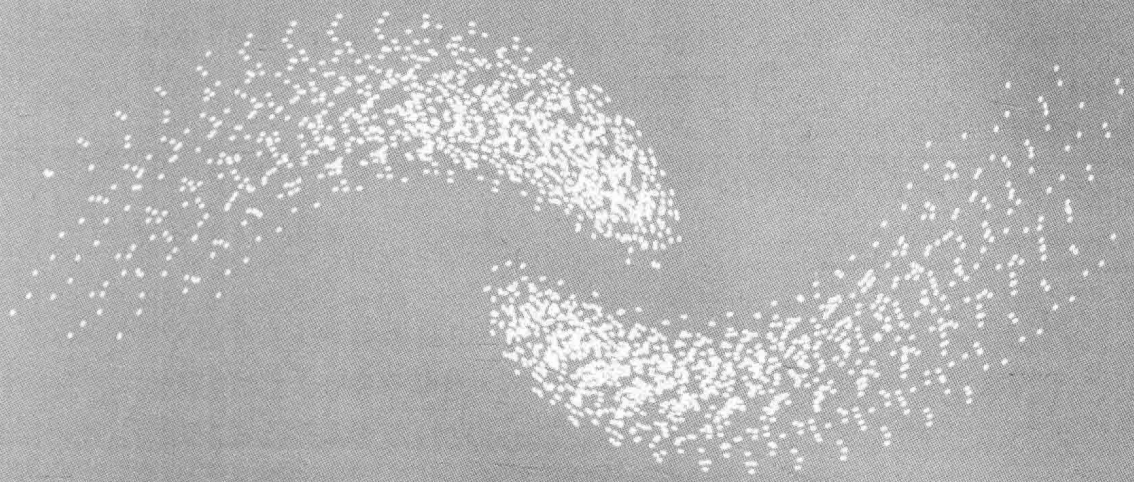
基于MATLAB的GPU编程

[希] 尼古劳斯·普洛斯卡斯（Nikolaos Ploskas）
尼古劳斯·萨马拉斯（Nikolaos Samaras） 著

张帆 倪军 李征 译

机械工业出版社
China Machine Press

高性能计算技术丛书



GPU Programming in MATLAB

基于MATLAB的GPU编程

[希] 尼古劳斯·普洛斯卡斯（Nikolaos Ploskas）
尼古劳斯·萨马拉斯（Nikolaos Samaras） 著

张帆 倪军 李征 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

基于 MATLAB 的 GPU 编程 /（希）尼古劳斯·普洛斯卡斯（Nikolaos Ploskas），（希）尼古劳斯·萨马拉斯（Nikolaos Samaras）著；张帆，倪军，李征译 . —北京：机械工业出版社，2019.5
（高性能计算技术丛书）
书名原文：GPU Programming in MATLAB

ISBN 978-7-111-62585-8

I. 基… II. ①尼… ②尼… ③张… ④倪… ⑤李… III. Matlab 软件 - 应用 - 图象处理 - 程序设计 IV. TP391.41

中国版本图书馆 CIP 数据核字（2019）第 083334 号

本书版权登记号：图字 01-2018-4814

基于 MATLAB 的 GPU 编程

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：迟振春　　责任校对：殷 虹
印　　刷：北京市兆成印刷有限责任公司　　版　　次：2019 年 5 月第 1 版第 1 次印刷
开　　本：186mm × 240mm 1/16　　印　　张：16.5
书　　号：ISBN 978-7-111-62585-8　　定　　价：99.00 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　　读者信箱：hzit@hzbook.com

The Translator's Words 译 者 序

自2007年英伟达公司推出CUDA以来，众核并行的GPU通用计算已经发展了12年，并成功应用于各个自然科学领域。相较于20世纪90年代兴起的第一代GPU通用计算（GPGPU），即利用图形描述语言（如OpenGL）控制GPU硬件进行通用数值计算，本次基于CUDA的GPGPU在影响范围、应用范围、参与人群等方面取得了阶跃式的进展。究其原因，一方面是硬件的发展，使得GPU并行的优势更加凸显；另一方面是软件的易用，使得基于C、Fortran等高级语言的CUDA架构对于算法的实现更加直观、便捷。

从我自己多年的学习和教学经验来看，GPU编程还存在“入门容易、进阶难”的问题。和其他高级语言相比，CUDA C不仅要求程序员掌握众核并行的语法、算法，将常规的“串行编程思路”转换为“众核并行编程思路”，还要求对于GPU硬件工作机制有深入理解，以满足不断优化的需求。这对于非计算机背景的程序员、科研工作者和学生来说，无疑是雪上加霜。

MATLAB由于数据处理能力强大、编程友好、集成函数功能多等优点在工科领域得到广泛应用，特别是在算法研究、系统原型设计等初期设计阶段成为编程工具的首选。但是MATLAB对于大数据的处理还存在计算效率低的问题，虽然可以通过转化为C语言或者使用内置的并行语句来解决，但仍旧杯水车薪。尽管C语言改写的方式能够取得更好的效率，但是高集成度的MATLAB函数使得改写工作成为一项“浩大”的工程。

MATLAB的GPU计算功能无疑为算法设计人员打开了一扇窗，无论内置函数或者MEX文件等方式均能快速地将现有代码分发至GPU上运行，从而达到一个数量级以上的运行效率提升。因而，对于广大使用MATLAB的读者来说，经过简单的语法学习就能够节省大量的程序运行时间，从而缩短研究的迭代周期和开发的研制周期。本书所提供的代码能够让读者更加方便、快捷地学习到MATLAB的GPU编程方法，并将其应用到自己的工作中。

本书的翻译工作由张帆、倪军、李征负责，参与其中的还有吴优、王云冲、唐嘉昕、赵晨茜等硕士研究生。由于译者的时间和水平有限，翻译中难免存在错误和疏漏，希望得到各位同行和专家的批评指正。

张帆

2019 年 1 月于北京

Foreword 推 荐 序

本书是对一系列专业 MATLAB 参考书的重要补充。很多 MATLAB 参考书通常将重点放在特定的工程领域，而本书主要针对那些熟悉 MATLAB 并希望使用多核处理器和 GPU 并行来提高程序运行速度的用户。近年来，并行计算技术虽然已被超级计算机广泛应用，但是对于普通 MATLAB 用户来说，还存在一定的技术门槛。现代计算机普遍安装了多核 CPU 和功能强大的 GPU，计算密集型 MATLAB 程序如果无法使用这些资源进行并行加速，将是一种严重的计算资源浪费。然而，鲜有资料能够详细介绍如何正确地将 MATLAB 程序运行在 GPU 上。此外，虽然 MATLAB 帮助文档中有对并行计算的初步介绍，但是对于专业的并行开发来说还是远远不够的。我相信本书能够填补这一空白。本书详细介绍了 MATLAB 中的 GPU 编程，从概述开始讨论了如何使用方便的 gpuArray，详细介绍了如何编译 CUDA 内核并将其集成到 MEX 文件中，读者可以从由浅入深、循序渐进的讲解中获取知识。本书介绍了很多实例，包括对内存或带宽等实际限制的讨论，使不同工程学科的用户都能够有所收获。针对难以理解和解决的 MATLAB 错误信息，本书也提供了相应的解决方案。虽然本书没有详细介绍 MATLAB 性能调优的更多细节，但是对于 MATLAB 的 GPU 编程的讨论是非常详细且新颖的。随着 GPU 编程的普及，这本专业、详细且可读性强的书籍将备受欢迎，成为任何一个想利用 GPU 将自己的 MATLAB 程序进行并行化的程序员的必读之作。

Yair Altman

《Accelerating MATLAB Performance》的作者，http://UndocumentedMatlab.com

前　言 Preface

MATLAB 是面向科学计算的高级语言，在很多科学领域它都作为一个快速的原型设计工具而被广泛使用。很多研究人员和公司使用 MATLAB 来解决计算密集型问题，并用它来加快代码运行速度。MATLAB 提供了并行计算工具箱，让用户可以使用多核处理器、计算机集群和 GPU 来解决计算密集型问题。

随着硬件的发展，在过去的十几年里 GPU 已经普及开来，并广泛应用在计算密集型问题上。目前主要有两个关于 GPU 的编程模型——CUDA 和 OpenCL，其中 CUDA 更加成熟稳定。程序员可以通过使用 CUDA C 编写 C/C++ 代码或使用 PGI 的 CUDA Fortran 编写 Fortran 代码等途径来使用 CUDA 架构。

本书采用的是另外一种使用 CUDA 架构的方法，适用的人群是学生、科研人员和工程师，他们使用 MATLAB 开发或维护应用程序，并希望在保留 MATLAB 优点的同时利用 GPU 编程实现加速。本书适用于对 MATLAB 编程有一定的经验但是不一定熟悉并行架构的读者，致力于帮助读者通过 GPU 来优化 MATLAB 程序，使其能够充分利用硬件优势来加速。

对于每个概念，本书都提供了示例，以便读者能够将理论运用到实际中。由于 MATLAB 使用者的专业背景不同，本书是以教程式学习方法为主，而不采用案例式学习方法。因此，本书的示例重点在于 GPU 实用编程技术，而不是特定的应用领域。所提供的例子包括图像处理、信号处理、优化、通信系统、统计学等多个领域的常见问题。

虽然有关 GPU 计算的 MATLAB 文档非常有用，但是文档对于 GPU 编程的具体实现问题并没有进行深入的讨论，从而使其对程序开发的指导作用非常有限。自 2010 年 MATLAB 支持 GPU 编程功能以来，已经开发了各种函数和工具箱，但是这些信息比较分散，本书的目的就是填补这个空白。另外，本书提供了很多来自不同科学领域的实际例子来阐述 MATLAB 的

GPU 功能。具有 CUDA C/C++ 编程经验的读者也能够通过利用 MATLAB 中的 CUDA C/C++ 代码或通过分析和优化 GPU 应用来获取更多的知识。

本书主要从两个方面着手进行介绍：

第一，MATLAB 为 GPU 编程所提供的一系列功能。这一方面将分为三个部分：

- 支持 GPU 的 MATLAB 内置函数（依赖并行计算工具箱）。
- GPU 的逐元素操作（不依赖并行计算工具箱）。
- 除了并行计算工具箱外，支持 GPU 的其他 MATLAB 内置函数工具箱，包括通信系统工具箱、图像处理工具箱、神经网络工具箱、相控阵系统工具箱、信号处理工具箱以及统计和机器学习工具箱等。

第二，当 MATLAB 不能在 GPU 上执行现有的代码段或者用户想要使用 CUDA 提供的加速库时，可以将 MATLAB 代码与 CUDA C/C++ 代码相链接。

本书的主要目标群体是：

- 学过 GPU 编程课程并想用 MATLAB 实现并行的本科生或研究生。
- 使用 MATLAB 开发或者维护应用程序并想使用 GPU 加速 MATLAB 代码的科研工作者。
- 希望在 MATLAB 中加速其计算密集型程序而无须用其他语言（如 CUDA C/C++ 或 CUDA Fortran）进行代码重写的工程师。

非常感谢 MathWorks 通过 MathWorks Book Program 提供的 MATLAB 学术许可，也感谢 NVIDIA Academic Partnership 提供的硬件支持，特别感谢 Ioannis Athanasiadis 提供的 MATLAB 工具箱的 GPU 功能实现与实例展示，最后感谢家人多年来对我们工作的支持与关怀。

Nikolaos Ploskas

Nikolaos Samaras

关于作者 *About the Authors*

Nikolaos Ploskas 是希腊西马其顿大学信息与通信工程系的助理教授，他在希腊马其顿大学应用信息系获得了计算机系统理学学士、硕士和博士学位，曾在美国卡内基－梅隆大学化学工程系从事博士后研究，其主要研究方向包括：运筹学、数学规划、线性规划、并行编程、GPU 编程、决策支持系统。

Ploskas 博士参与了多项国际、国家级研究项目，发表论文 40 余篇，包括高影响力的期刊论文、会议论文以及书籍章节，并担任过许多学术期刊的审稿人。2014 年，他获得了 HELORS（希腊运筹学协会）颁发的最佳运筹学博士学位论文奖。

Nikolaos Samaras 是希腊马其顿大学信息科学学院应用信息系的教授，研究兴趣是利用计算机科学和运筹学的知识解决各种工程和科学系统的复杂问题，具体包括：

- 线性 / 非线性优化：理论、算法和软件
- 网络优化：理论、算法和软件
- 科学计算：高性能计算和 GPU 编程

Samaras 教授是《 Operations Research ： An International Journal 》的编委，是许多学术期刊的审稿人，并在 HELORS 担任过多个职务。2005 年，他获得了 Thomson ISI/ASIS&T 引文分析研究基金。

Samaras 教授已经在高影响力的期刊上发表了 30 余篇期刊论文，具体包括《 Computational Optimization and Applications 》《 Computers and Operations Research 》《 European Journal of Operational Research 》《 Annals of Operations Research 》《 Journal of Artificial Intelligence Research 》《 Discrete Optimization 》《 Applied Mathematics and Computation 》《 International Journal of Computer Mathematics 》《 Electronics Letters 》《 Computer Applications in Engineering Education 》《 Journal of Computational Science 》《 Applied Thermal Engineering 》等。此外，他还发表了超过 85 篇会议论文。

Contents 目　录

第 1 章 Chapter 1

引　言

本章目标

本章主要介绍并行编程和基于 CUDA 的 GPU 编程的一些关键特性，并给出一些可以通过 GPU 进行加速的实际应用例子。读完本章后，读者可以具备以下能力：

- ❑ 理解并行编程的关键概念。
- ❑ 理解 GPU 编程的关键概念。
- ❑ 描述支持 CUDA 的 GPU 架构。
- ❑ 列出可以使用并行和 GPU 编程的实际应用。

1.1 并行编程

1.1.1 并行计算导引

用作串行计算的软件可以在单处理器上完成整个执行过程。因此，这个执行过程可以被分解为一系列离散指令，这些指令串行地依次执行，如图 1-1 所示。

例如，计算两个列向量 a 和 b 的点积：

$$a=\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix},\quad b=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

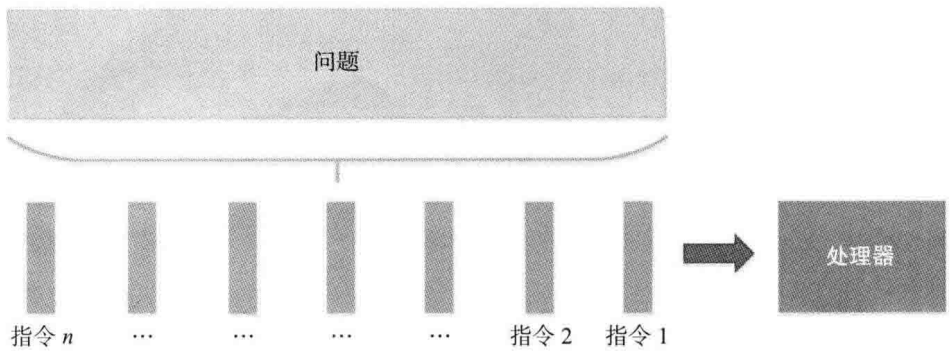


图 1-1 串行计算概览

在 MATLAB 中，两个向量的点积（内积）是通过以下串行程序实现的：

```
1.   sum = 0;
2.   n = length(a);
2.   for i = 1:n
3.      sum = sum + a(i) * b(i);
4.   end
```

这些程序指令会在一个处理器上串行地执行，如图 1-2 所示。

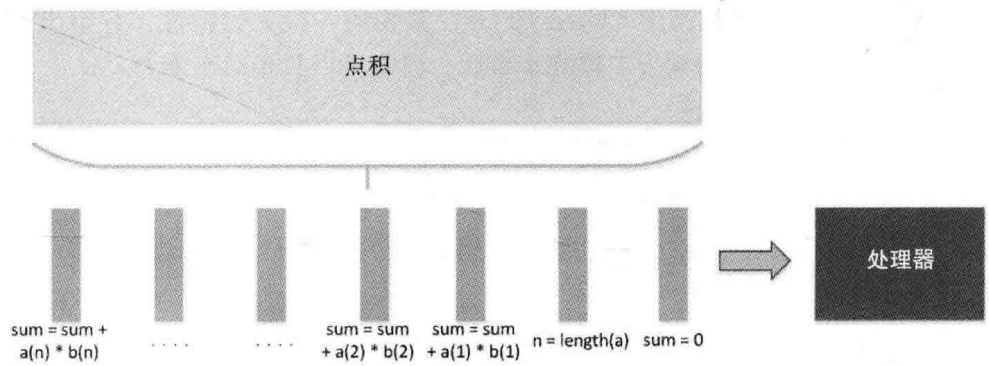


图 1-2 串行计算示例——向量的点积运算

与串行计算不同，并行计算软件是在多个计算资源上执行的。因此，一个计算问题被分解为多个可以同时求解的计算单元，每个计算单元通过一系列程序指令来实现，来自不同计算单元的指令会在不同的计算资源上同时执行。当然，来自相同计算单元的指令会在一个特定计算资源上串行地依次执行，如图 1-3 所示。

考虑到需要计算大量向量的点积，可以在每一个计算资源上完成两个向量的点积运算，从而实现并行计算，如图 1-4 所示。

上述计算资源可以是处理器核心（单台计算机中单处理器的多个核心）或者处理器（单台计算机中的多处理器，或者通过网络连接的多台计算机中的多处理器）。

并行计算主要用于以下情况：

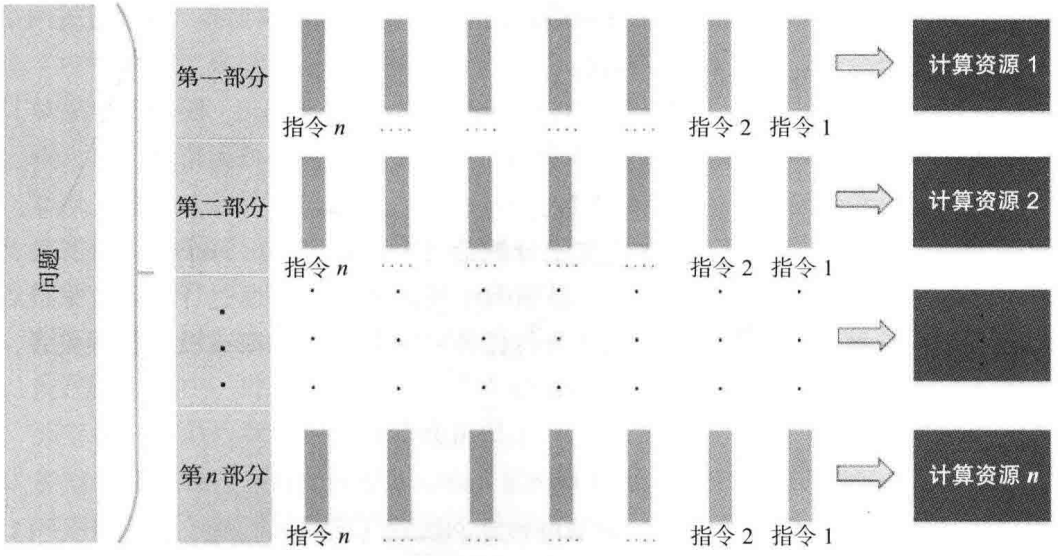


图 1-3 并行计算概览

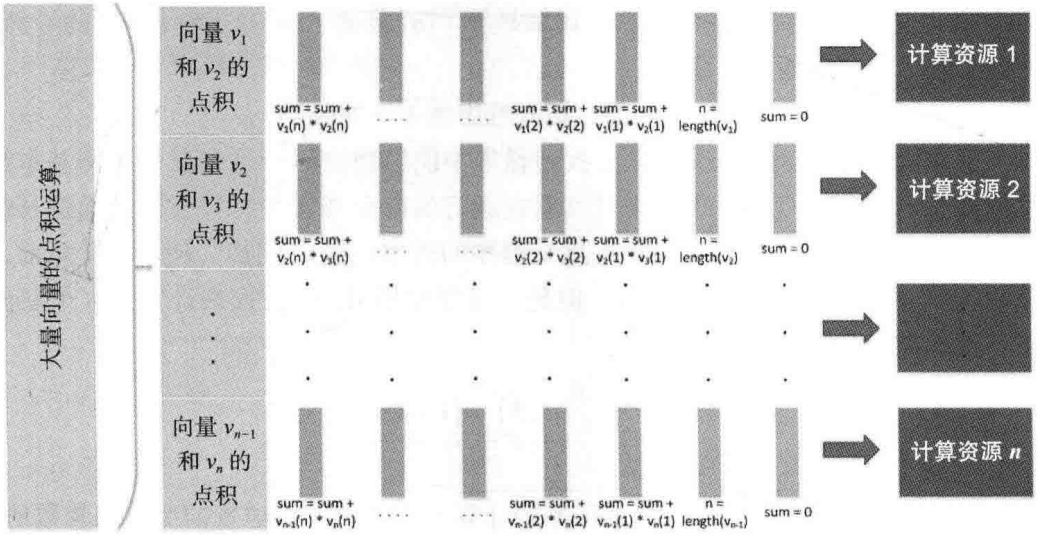


图 1-4 并行计算示例——向量的点积运算

- **减少程序的执行时间**：使用更多的计算资源来执行一个程序会减少其执行时间，同时这种执行时间的减少也可以节省计算成本。对于实时计算类型的应用来说，需要在特定的时间范围内执行完毕，因而减少运行时间是非常重要的。
- **并发地执行更多的计算任务**：单个计算资源只能同时执行一个特定的任务。使用多个计算资源将会增加同时并发执行的任务数量。
- **解决大计算量问题**：受单台计算机内存的限制而无法解决的大计算量问题可以通过使用更多的计算资源来完成。

但是，依然存在根本无法并行化或者难以进行并行化的计算问题。下面通过两个例子来理解这些问题是否可以被正确地并行化。

例 1 可并行问题

计算特定词语在大段文字中出现的次数。

这个问题可以并行地解决。将大段文字分配给每个计算资源，每个计算资源在其所对应的部分文字中查找该词语出现的次数，然后每个处理器的结果通过计算共享来最终确定该词语出现的总次数。这种类型的问题在并行任务之间几乎不存在或没有依赖关系，被称为完美的并行任务。

例 2 不可并行问题

使用公式 $f(n)=f(n-1)+f(n-2)$ 计算斐波那契数列（0，1，1，2，3，4，5，13，…）中的前 n 个数，其中 $f(0)=0$，$f(1)=1$。

这个问题是无法进行并行化的程序中具有代表性的例子，因为每一个斐波那契数的计算都需要依赖其他斐波那契数的计算。比如计算 $f(n)$ 需要 $f(n-1)$ 和 $f(n-2)$ 的计算结果，因此这三个数不能独立地开展计算。

综上，在开始并行化串行程序之前，请先考虑如下一些技巧：

（1）确定程序能否被并行化。尽力找到程序中能够独立运行且并行任务间基本没有通信需求的部分。如果发现程序中没有可以独立运行的部分或者并行任务间的通信开销非常大，那么需要考虑寻找其他算法来解决这个程序的效率问题。例如，例 2 中计算前 n 个斐波那契数的问题是无法进行并行化的，但是，可以使用 Binet 公式来计算第 n 个斐波那契数，如下所示：

$$f(n)=\frac{\left(1+\sqrt{5}\right)^n-\left(1-\sqrt{5}\right)^n}{2^n\sqrt{5}}$$

根据这个公式，一个位置的斐波那契数的计算不再依赖其他位置的斐波那契数的计算，并行任务间的依赖关系没有了，因而能够容易地并行化前 n 个斐波那契数的计算问题。

（2）使用分析器或者性能分析工具来寻找程序中的密集计算部分，然后将并行优化工作的重点放在这些程序上。

（3）找出可能导致可并行化任务停止的瓶颈。

（4）尽可能使用第三方高度优化的并行库。

1.1.2 并行计算机的类别

根据 Flynn 分类法[1]，多处理器计算机架构可以划分为两种独立的维度：（i）指令流（Instruction stream, I）；（ii）数据流（Data stream, D）。每一个维度都有如下两种可能状态之

一：（i）单指令流、单数据流（Single, S）；（ii）多指令流、多数据流（Multiple, M）。因此，会产生 4 种可能的计算机类型（如图 1-5 所示）：

- **单指令单数据**（Single Instruction Single Data, SISD）。通常指在任何给定的时钟周期内对单个数据流执行单条指令的串行计算架构。最老式的单处理器 / 核计算机采用的就是这种老式的计算机架构。
- **单指令多数据**（Single Instruction Multiple Data, SIMD）。通常指在任何给定的时钟周期内对多个不同数据流执行一条相同指令的并行计算架构。这种架构多用于处理器阵列（如 MasPas MP-1 以及 MP-2）、向量处理流水线（如 IBM 9000）以及大多数现代计算机中。此外，图形处理器（Graphical Processing Unit, GPU）也是用这种计算架构，在稍后章节将做具体的介绍。
- **多指令单数据**（Multiple Instruction Single Data, MISD）。通常指在任何给定的时钟周期内对单个数据流执行多条指令的并行计算架构。这种并行架构是非常理论化的，目前没有任何已知的并行计算机在使用该架构。
- **多指令多数据**（Multiple Instruction Multiple Data, MIMD）。通常指在任何给定的时钟周期内对多个数据流执行多条指令的并行计算架构。这种并行架构一般用在最新的超级计算机和多核计算机中。

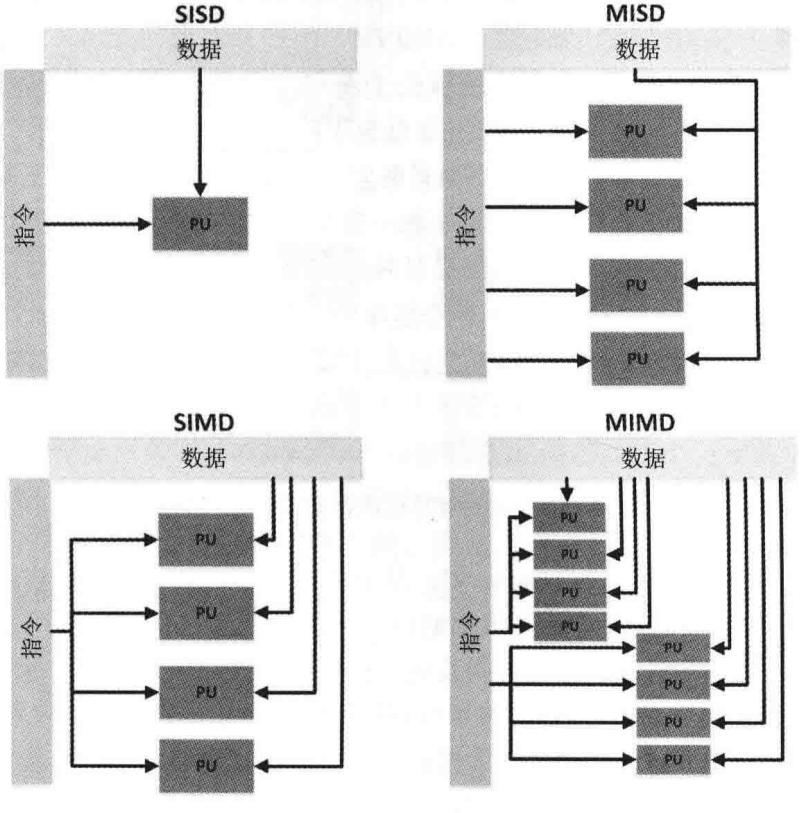


图 1-5 并行计算机的类别

1.1.3 并行计算机的内存架构

在当前的计算机架构中，有三种并行计算机的内存架构（如图 1-6 所示）：

（1）**共享内存**（shared memory）。在共享内存并行计算机中，多个处理器能够独立地访问并共享内存数据。如果一个处理器对某个内存位置中的数据进行修改，那么其他多个处理器获取的该内存位置的数据将是修改后的数值。共享内存并行计算机可以进一步划分为如下两类：

- **均匀访存模型**（Uniform Memory Access, UMA）：指相同类型的处理器具有相同的内存访问时间。这种架构常用在对称多处理器（Symmetric Multiprocessor, SMP）计算机中。
- **非均匀访存模型**（Non-Uniform Memory Access, NUMA）：许多 SMP 连接在一起时，一个 SMP 可以直接访问其他 SMP 的内存。因此，并非所有处理器都能以相同的时间访问所有的 SMP 内存。此外，在该架构下，处理器访问本地内存的速度比访问其他 SMP 内存的速度要快一些。

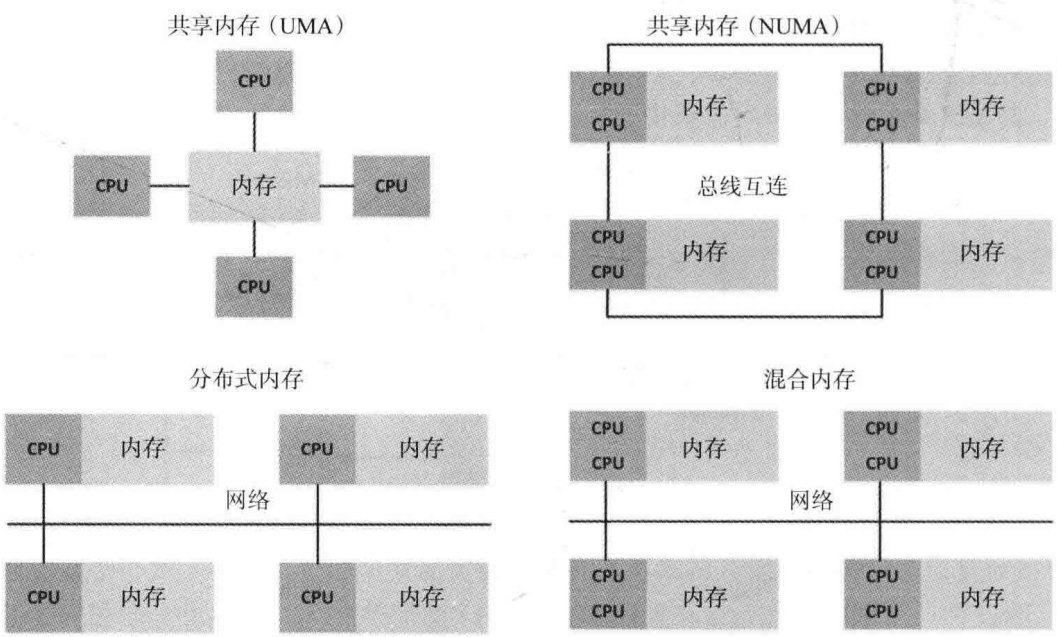


图 1-6 并行计算机内存架构的类型

（2）**分布式内存**（distributed memory）。所有处理器都有自己的本地内存，并且可以独立地访问它。在该架构下，所有计算机都通过网络连接起来，例如以太网，并且可以从其他计算机请求数据（一台计算机无法访问其他计算机的内存，但是可以通过编程方式将数据从一台计算机传输到另一台计算机）。连接计算机的网络类型会影响数据传输的速度。

（3）**混合内存**（hybrid memory）。大多数现代超级计算机使用了综合共享内存和分布式内存的混合内存架构。一台机器中的所有处理器可以共享内存，也可以从其他计算机请求