计算机组成原理
CH6_并行处理与计算

Computer Organization and Design

Chapter 6: Parallel Processing and Computing

马子睿

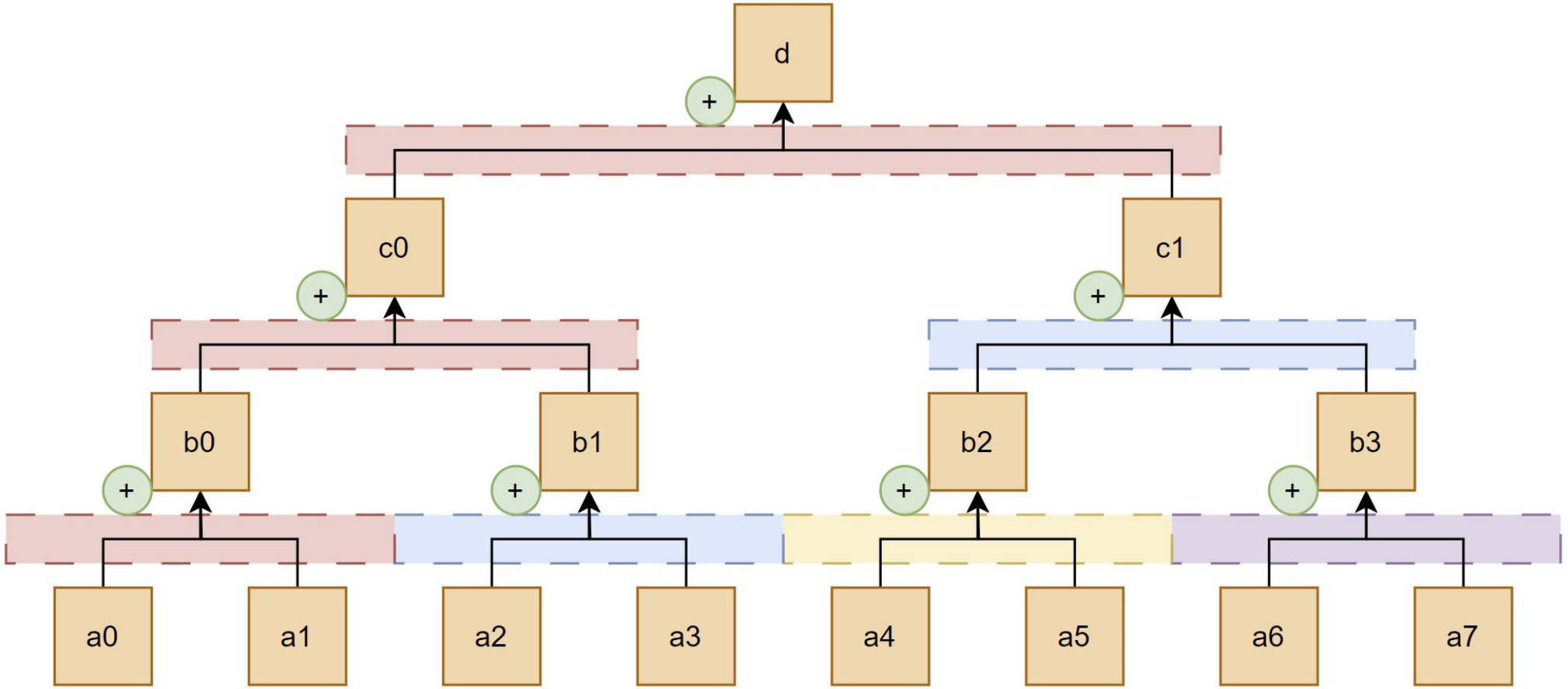
目录

并行处理程序

Parallel Processing Program

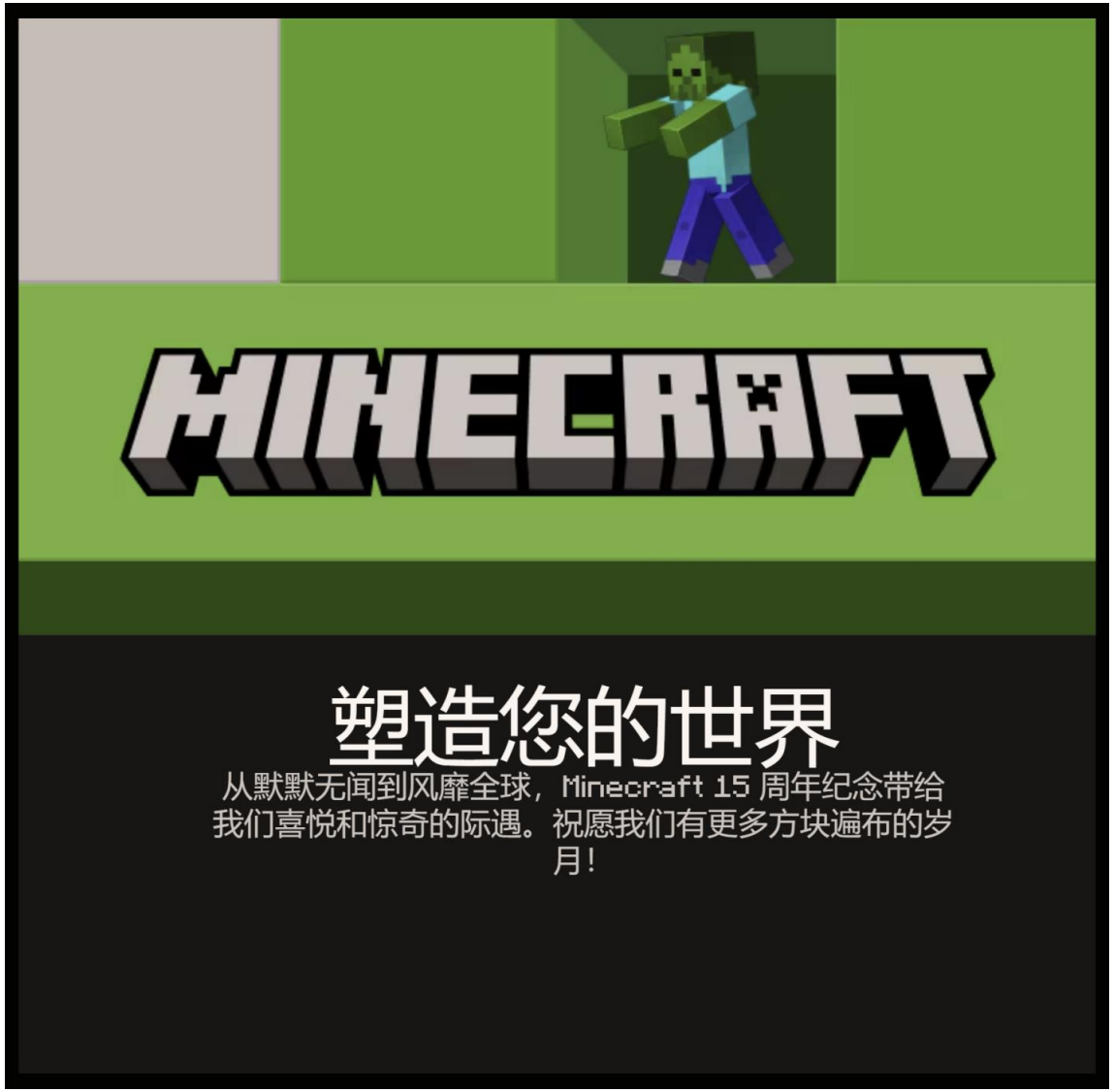
并行程序引入

- 用4个同样的处理器将8个整数求和，你会怎么做?



并行程序的创建

- **并行程序**
  - 能够并行（在多个处理器上）运行的单个程序
- **并行程序的主要困难在于软件**
  - 软件重写：一核有难，八核围观
  - 提高性能和效率：同步、通信成本不可忽视
- **并行程序的组成**
  - 并行部分：二叉树每层加法
  - 串行部分：二叉树层间归约



并行程序加速比度量——Amdahl定律

- 假定计算负载固定不变，使用增加处理器数量来提高计算速度
- 无额外开销的Amdahl定律形式化表述：

$$S = \frac{W_s + W_p}{W_s + W_p / p}$$

其中，W_s为串行分量，W_p为并行分量，p为处理器核心数，S为加速比

- 引入串行占比f，则其表述可以变化为：

$$S = \frac{f + (1-f)}{f + (1-f)/p} = \frac{p}{1 + f(p-1)}$$

- Amdahl定律给出了一个悲观的结论：即便处理器数目无限增大，加速比上限也最多为1/f

并行程序加速比度量——Amdahl定律

- 事实上，并行处理需要有同步、通信等代价，它们不可以被忽略
- 有额外开销的Amdahl定律形式化表述：

$$S = \frac{W_s + W_p}{W_s + W_p / p + W_O}$$

其中，W_s为串行分量，W_p为并行分量，W_O为额外开销，p为处理器核心数，S为加速比

- 引入串行占比f，当p趋近于无穷大时，其表述可以变化为：

$$S = \frac{1}{f + W_O / W}$$

- **串行分量越大、并行额外开销越大，加速比越小**

Amdahl定律：例题

- 某并行程序中，串行部分计算时间占比20%，并行部分计算占比80%，不考虑额外开销，请计算：
  - 若使用5个处理器对其进行加速，则加速比为多少？
  - 根据Amdahl定律，该程序最大并行加速比为多少？

$$S = \frac{W_s + W_p}{W_s + W_p / p} = \frac{20\% + 80\%}{20\% + 80\% / 5} = 2.78$$

$$S = \frac{1}{f} = 5$$

指令流和数据流

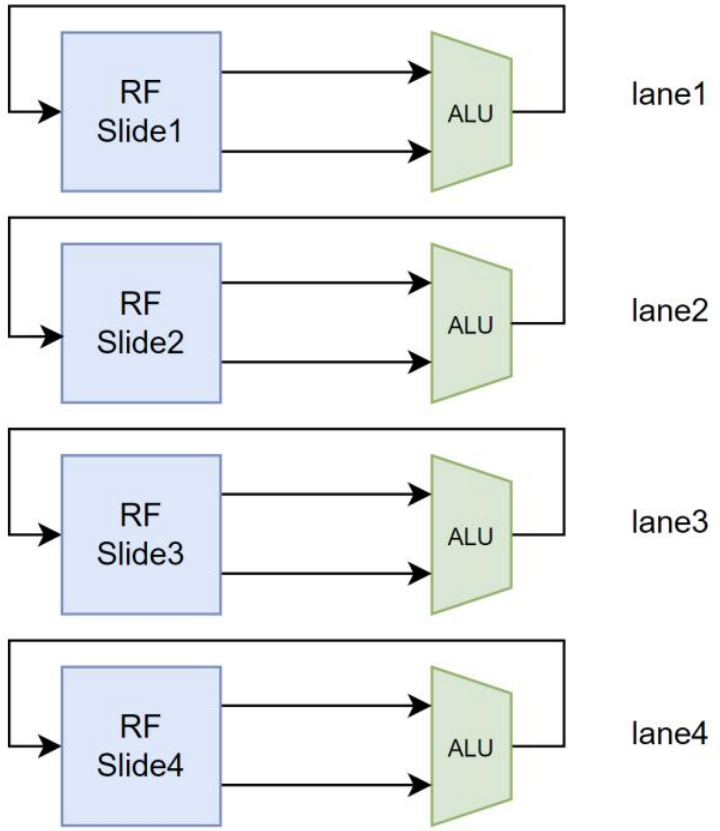
Instruction Stream and Data Stream

指令流与数据流

- **SISD (Single Instruction, Single Data)**
  - 单指令单数据
  - 示例：五级流水线
- **MISD (Multiple Instruction, Single Data)**
  - 多指令单数据
  - 示例：冗余系统和错误检测系统
- **SIMD (Single Instruction, Multiple Data)**
  - 单指令多数据
  - 示例：向量指令扩展
- **MIMD (Multiple Instruction, Multiple Data)**
  - 多指令多数据
  - 示例：多核处理器和分布式系统

SIMD与向量指令

- **向量体系结构**
  - SIMD的一种重要实现
  - 一条指令可以操作连续的多个数据
  - 在处理器内部集成了向量寄存器堆，采用车道（lane）策略进行计算
- **向量指令示例**
  - vadd：向量加法
  - vload：向量加载



SPMD（Single Program, Multiple Data）

- **SPMD**
  - **特点**：类似于SIMD，但是在SPMD中，每个处理单元执行的是相同的程序，但是操作的数据可能不同。
  - **工作原理**：所有的处理单元同时开始执行相同的程序，但是它们可能处理的是不同的数据集合或者数据部分。这种模型通常用于解决大规模计算问题，其中数据可以被分割成多个部分并行处理，从而加速计算过程。
- **SPMD举例：**
  - 假设有一个大型的矩阵计算任务，可以将矩阵分成多个子矩阵，然后在不同的处理单元上并行执行相同的矩阵计算程序，每个处理单元处理一个子矩阵的计算。

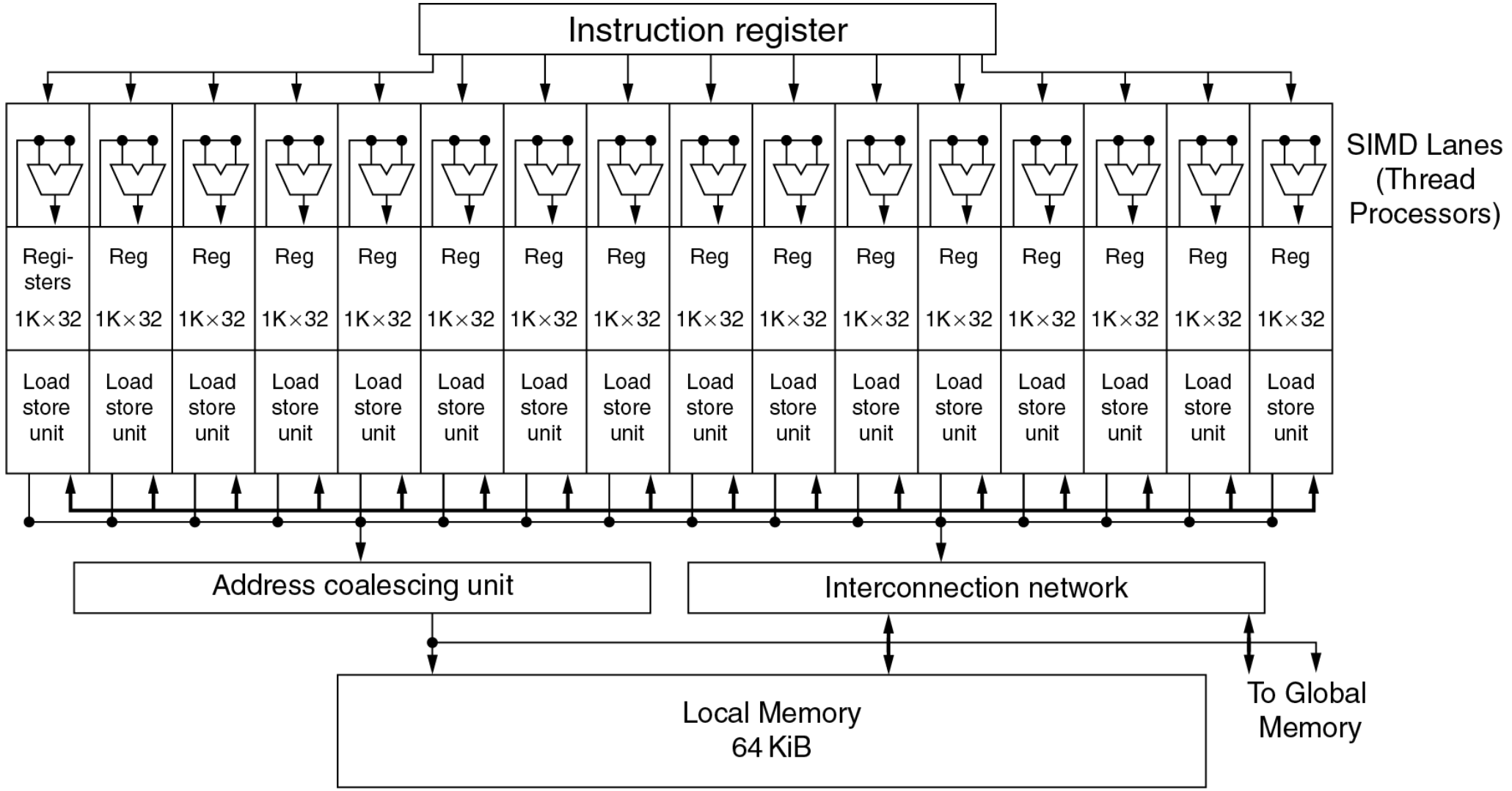
GPU简介

Introduction to GPU

GPU的历史

- **早期的图形加速器：**
  - 20世纪80年代，个人计算机开始出现，图形用户界面（GUI）变得流行。最早的图形加速器主要用于简单的2D图形加速，帮助计算机显示图形界面，例如，IBM PC/AT上的VGA。
- **3D图形加速器的崛起：**
  - 20世纪90年代，随着计算机游戏的兴起和图形应用的增加，3D图形加速器开始出现。3Dfx Interactive的Voodoo图形加速器是最早的3D图形加速器之一，它大大提高了计算机游戏的图形性能和质量。
- **通用计算：**
  - 随着GPU的性能和并行处理能力的提高，人们开始意识到GPU不仅可以用于图形处理，还可以用于通用计算。2006年，NVIDIA推出了CUDA平台，使开发人员能够利用GPU的并行计算能力进行通用计算。同样，AMD也推出了类似的技术，如AMD的OpenCL。

GPU的结构


Instruction register
SIMD Lanes (Thread Processors)
Regi-sters 1K×32
Reg 1K×32
Load store unit
Address coalescing unit
Interconnection network
Local Memory 64 KiB
To Global Memory

GPU的编程模型

- **使用线程 (SPMD）编程模型，不是用SIMD指令编程**
  - 每个线程执行同样的代码，但操作不同的数据元素
  - 每个线程有自己的上下文（即可以独立地启动/执行等）
- **计算由大量的相互独立的线程完成**
  - **核函数**：核函数是在GPU上并行执行的计算单元，它们由程序员编写并在GPU上执行。核函数通常是一些短小的代码片段，用于执行特定的计算任务。
  - **线程和线程块**：在GPU编程中，核函数会被分配给多个线程执行。这些线程通常被组织成线程块，每个线程块包含多个线程，这些线程可以协作执行核函数中的计算任务。
  - **网格**：线程块可以被组织成网格，网格是线程块的集合。程序员可以根据任务的需求来组织线程块和网格，以便在GPU上高效地执行计算任务。

GPU的执行模型

- **一组执行相同指令的线程由硬件动态组织成线程束（warp）**
  - GPU硬件会将连续的线程按照一定规则组织成线程束，通常是以32个线程为一组。这些线程会同时执行相同的指令，但操作的数据可能不同。
  - 这种组织方式有助于利用GPU的并行计算能力，因为可以同时执行相同的指令，而不需要对每个线程都分别发送指令，从而提高了计算效率。
- **一个warp是由硬件形成的SIMD操作**
  - GPU的处理器架构在设计上就支持将一组线程组织成warp，并且这些线程同时执行相同的指令。
  - 这种组织方式是由GPU硬件内部的控制逻辑实现的，而不是由编程人员显式地控制。只需将任务分配给线程，而不需要担心具体的线程组织和调度细节。


感谢！
Thank you!
马子睿
2024-6-7