GPU 在 SIMT 编程本质

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem

Talk Overview

1. **AI 计算体系**
   - 深度学习计算模式
   - 计算体系与矩阵运算
2. **AI 芯片基础**
   - 通用处理器 CPU
   - 通用图形处理器 GPU
   - AI专用处理器 NPU/TPU
3. **GPU详解**
   - 英伟达GPU架构发展
   - Tensor Core和NVLink
4. **国外 AI 芯片**
   - 特斯拉 DOJO 系列
   - 谷歌 TPU 系列
5. **国内 AI 芯片**
   - 壁仞科技芯片架构
   - 寒武纪科技芯片架构
6. **AI芯片的思考**
   - SIMD&SIMT与编程体系
   - AI芯片的架构思路与思考

Talk Overview

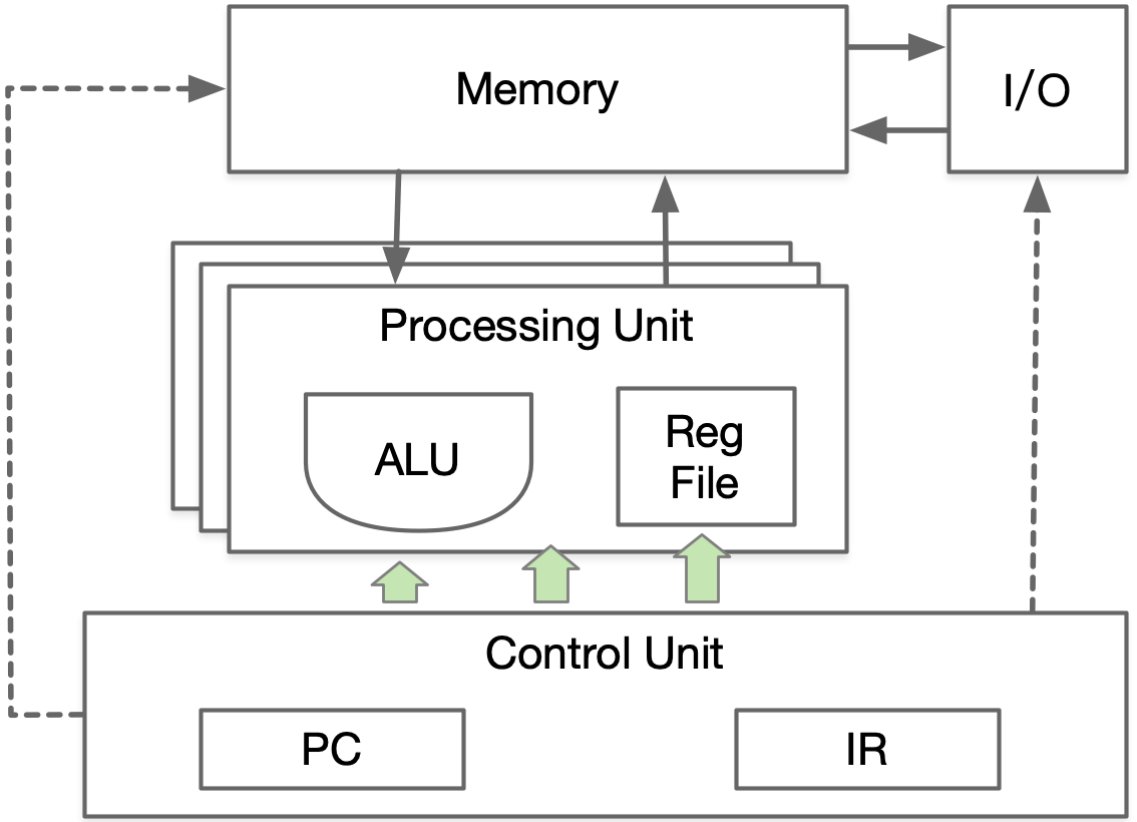
I. GPU 在 SIMT 编程本质

- SIMD vs SIMT 回顾
- 编程模型 vs 执行模型
- 并行的编程方式
- 英伟达 GPU 编程模型

1. SIMD SIMT回顾

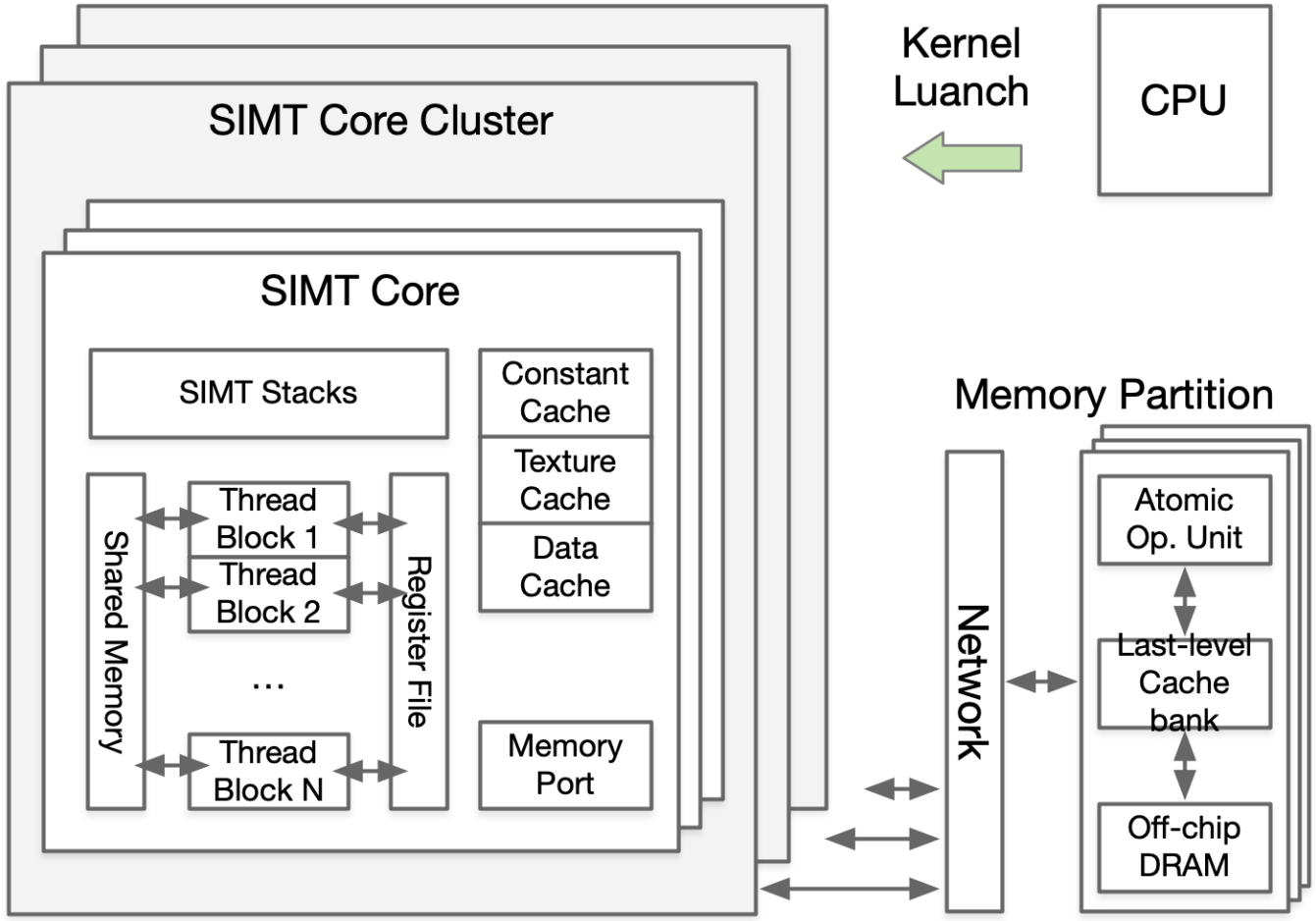
SIMD 计算本质

- SIMD 实现一次加法可以完成多个元素的加法。因此这要求硬件上，增加 ALU 单元的数量，同时也需要增加同一个功能单元的数据通路数量，才能够实现相同时钟周期内提升计算的吞吐量。



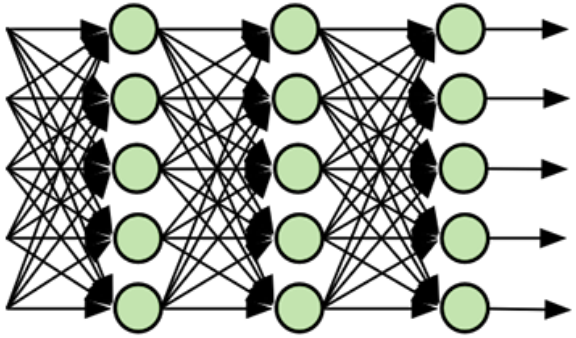
SIMT 硬件结构

- SIMT 提供一个多核系统，每个核有独立的寄存器文件 RF、ALU、Data Cache，但是只有一个 Program Counter寄存器、一个指令Cache和一个译码器，指令被同时广播给所有的 SIMT 核。
- 以 GPU 为例：由多个 SIMT Core Cluster 组成，每个SIMT Core Cluster 由多个 Core 构成，Core 中有多个 Thread Block。



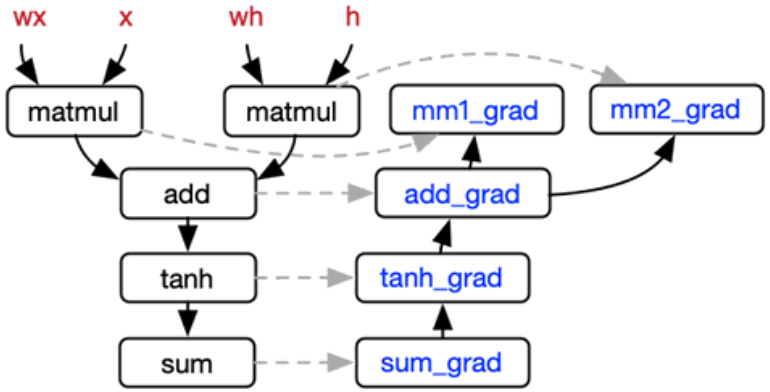
AI框架的开发流程

(a) 定义神经网络

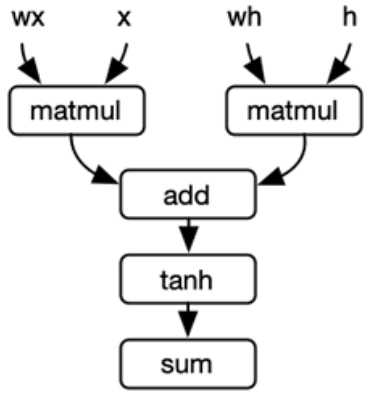


(b) 编写对应程序

```
x2h = ai.matmul(wx, x)
h2h = ai.matmul(wh, h)
next_h = x2h + h2h
next_h = ai.tanh()
next_h = next_h.sum(b)
```



(d) 根据自动微分原理构建反向图



(c) 程序构建正向图

CUDA 代码开发流程

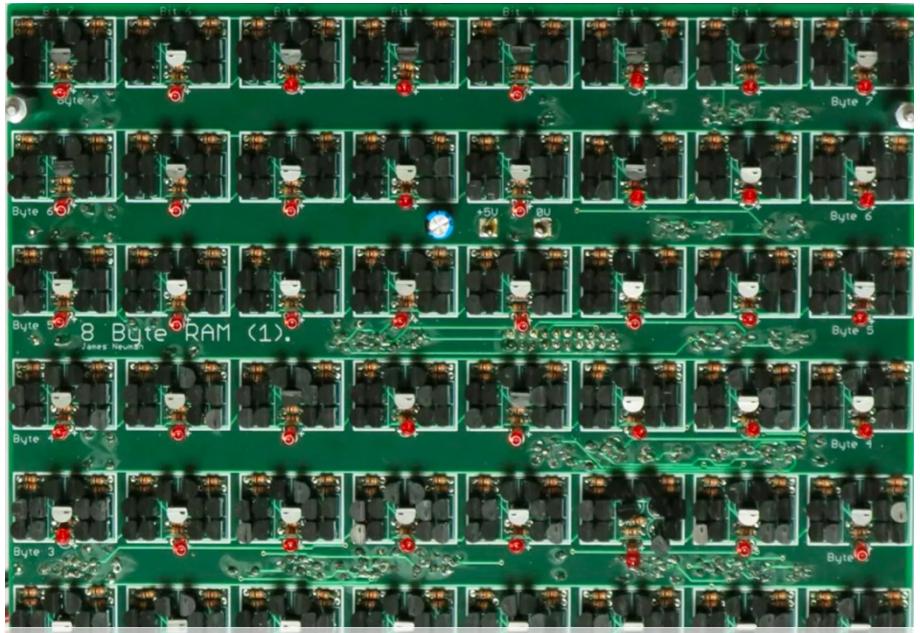
```cpp

#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>


__global__ void MatMul(int *M,int *N,int *P,int width)
{
    int x = threadIdx.x;
    int y = threadIdx.y;

    float Pervalue = 0;

    float elem1 = 0.0,elem2 = 0.0,value = 0.0;
    for(int i = 0;i < width;i++)
    {
        elem1 = M[y * width + i];//取M矩阵的一行
        elem2 = N[i * width + x];//取N矩阵的一列

        value += elem1 * elem2;//求和
    }

    P[y * width + x] = value;
}

```

2. 编程模型 vs 执行模型

什么是编程模型？什么是执行模型？

- Programming Model (Software)

- Execution Model (Hardware)



Programming Model vs. Hardware Execution Model

- **Programming Model** refers to how the programmer expresses the code
- **编程模型**：开发者如何去使用代码、程序去表达需要执行的计算任务

- **Execution Model** refers to how the hardware executes the code underneath
- **硬件执行模型**：硬件如何执行代码指令

- Execution Model can be very different from the Programming Model，两者之间的差异会非常大

3. 并行的编程模型

实现并行的编程方式

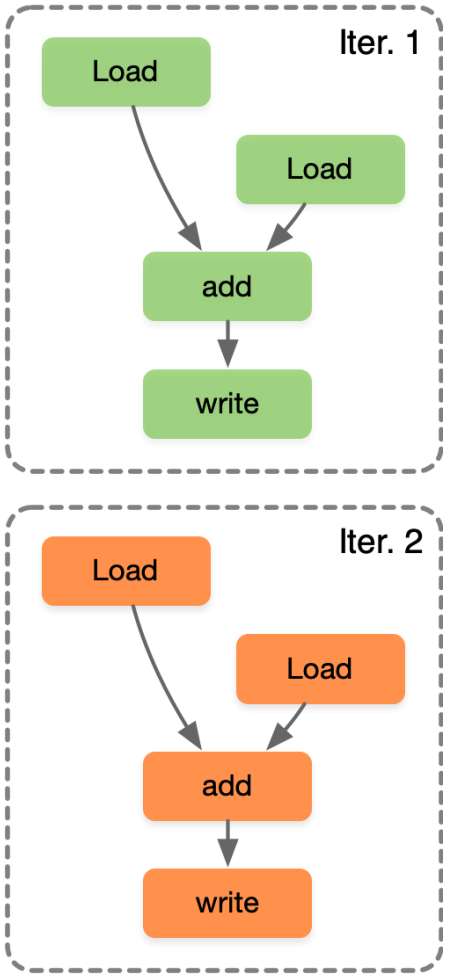
使用三种不同的编程模型来看看指令级别的执行方式：

1. Sequential (SISD)，利用传统 CPU 串行执行
2. Data-Parallel (SIMD)，现代 CPU & GPU 数据并行执行
3. Multithreaded (MIMD/SPMD)，利用特殊超算中心的多线程执行

```
1
2    for (int i = 0; i < N; ++i){
3        C[i] = A[i] + B[i];
4    }
5
```

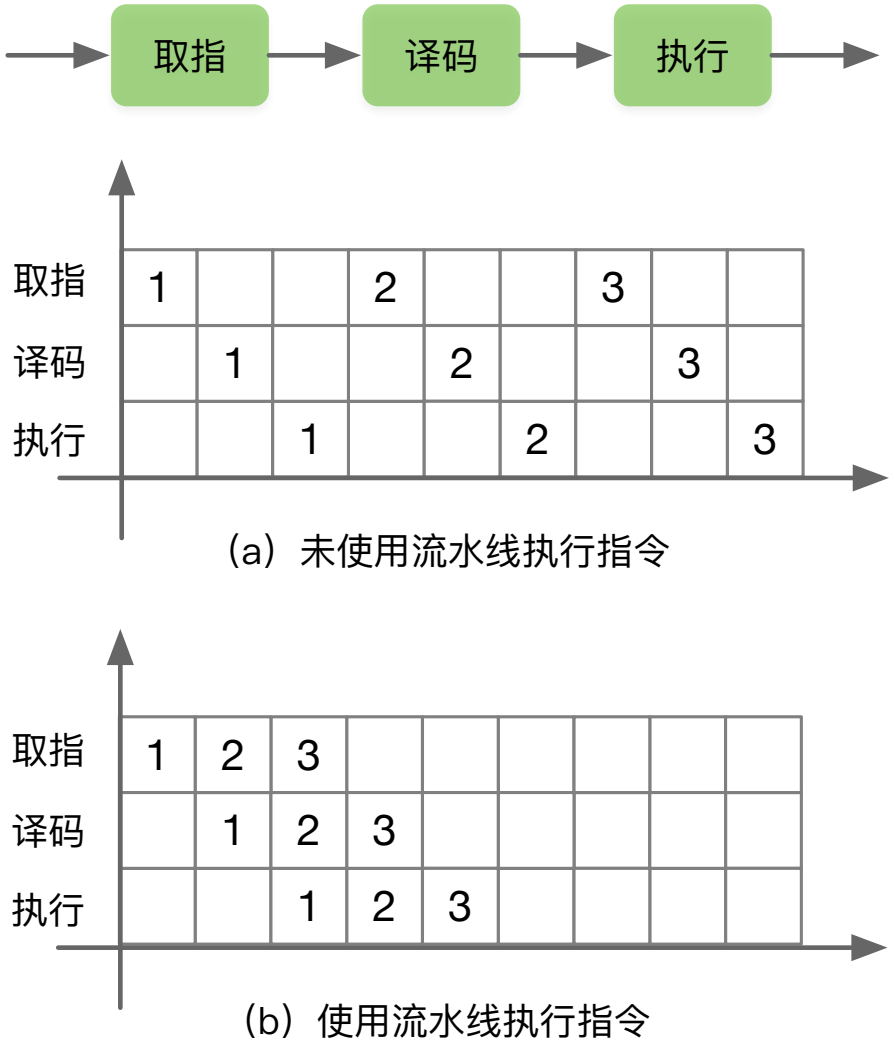
方式1: 串行执行，类似于 SISD

1. **流水执行 PPE**
   - 程序执行时,多条指令重叠进行操作的一种任务分解技术
2. **乱序执行 OOE**
   - 独立的指令执行
   - 把依赖性指令移到独立指令后
   - 循环由硬件通过指令动态展开
3. **超长指令集 VLIW 架构**
   - 把许多条指令连在一起，同一时钟周期执行多条指令，增加运算的速度



方式1: 串行执行，类似于 SISD

1. **流水执行 PPE**
   - 程序执行时,多条指令重叠进行操作的一种任务分解技术
2. **乱序执行 OOE**
   - 独立的指令执行
   - 把依赖性指令移到独立指令后
   - 循环由硬件通过指令动态展开
3. **超长指令集 VLIW 架构**
   - 把许多条指令连在一起，同一时钟周期执行多条指令，增加运算的速度

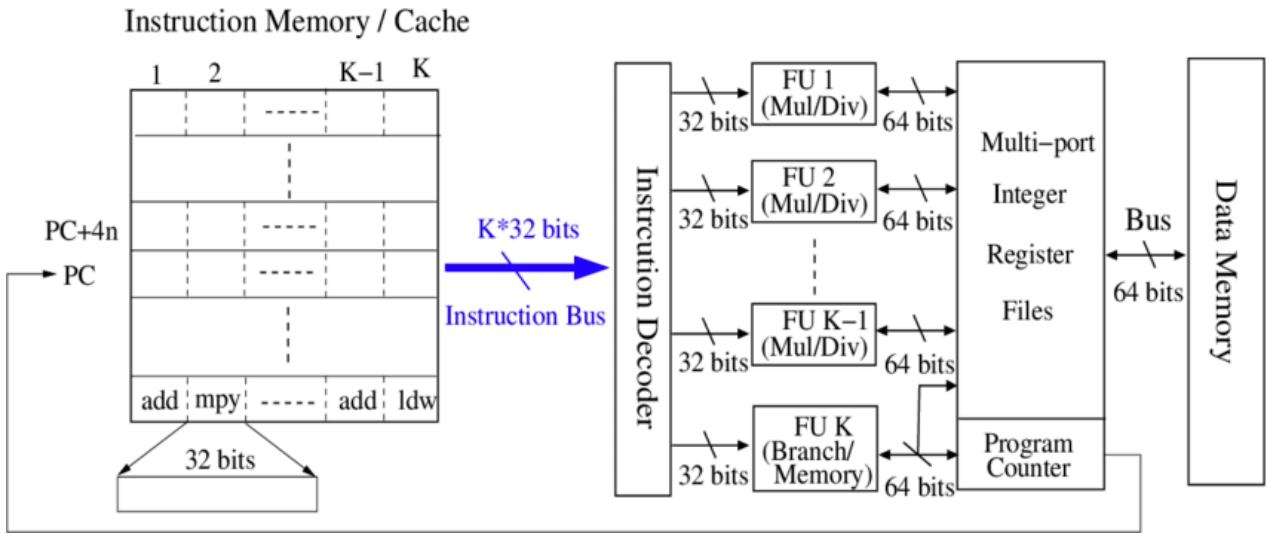


方式1: 串行执行，类似于 SISD

1. 流水执行 PPE
   - 程序执行时,多条指令重叠进行操作的一种任务分解技术
2. 乱序执行 OOE
   - 独立的指令执行
   - 把依赖性指令移到独立指令后
   - 循环由硬件通过指令动态展开
3. 超长指令集 VLIW 架构
   - 把许多条指令连在一起，同一时钟周期执行多条指令，增加运算的速度

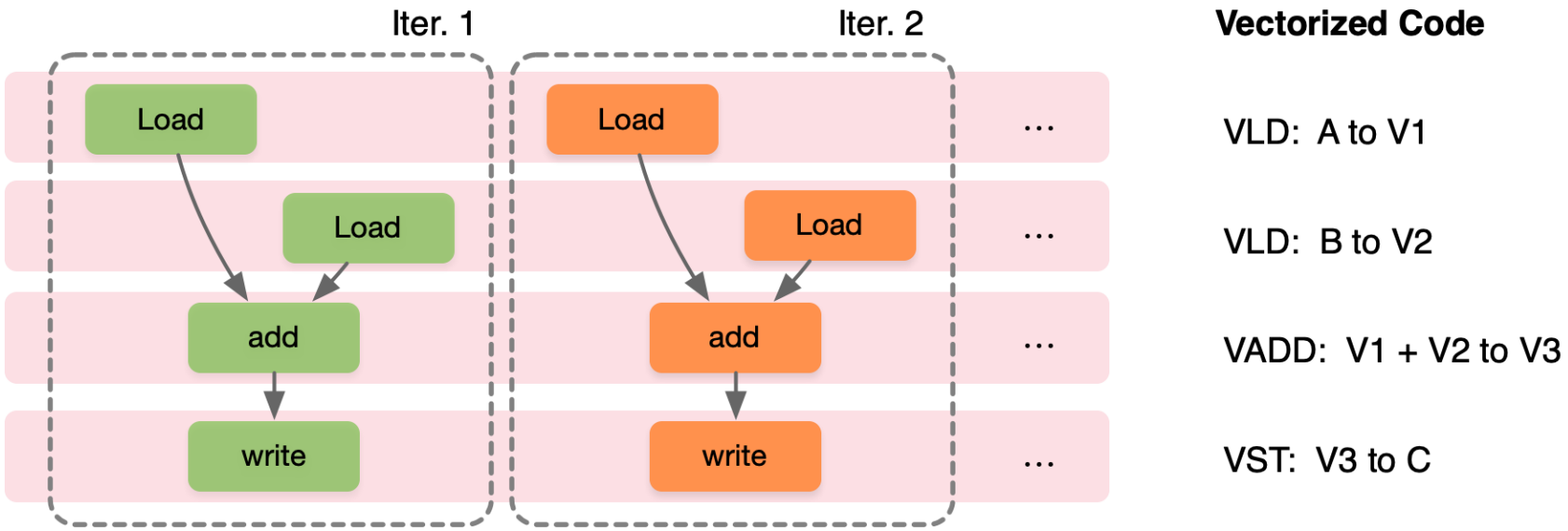
方式1: 串行执行，类似于 SISD

1. 流水执行 PPE
   - 程序执行时,多条指令重叠进行操作的一种任务分解技术
2. 乱序执行 OOE
   - 独立的指令执行
   - 把依赖性指令移到独立指令后
   - 循环由硬件通过指令动态展开
3. **超长指令集 VLIW 架构**
   - 把许多条指令连在一起，同一时钟周期执行多条指令，增加运算的速度



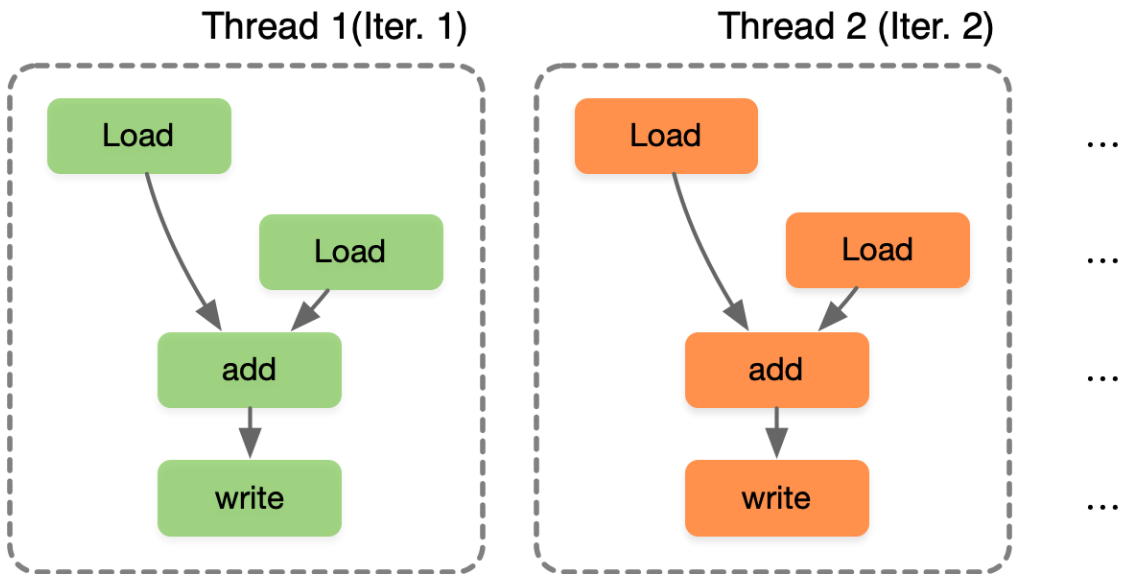
方式2: 数据并行, SIMD

- 通过循环中的每个迭代独立实现：
  - **程序上**：程序员编写SIMD指令或编译器生成SIMD指令，在不同数据的迭代中执行相同指令。
  - **硬件上**：通过提供 SIMD 较宽的 ALU 执行单元



方式3: 多线程，SPMD

- 通过循环中的每个迭代独立实现：
  - 程序上：程序员或编译器生成线程来执行每次迭代，使得每个线程在不同的数据上执行相同的计算
  - SIMT 独立的线程管理硬件来使能硬件处理方式



方式3: 多线程，SPMD

- 通过循环中的每个迭代独立实现：
  - 程序上：程序员或编译器生成线程来执行每次迭代，使得每个线程在不同的数据上执行相同的计算
  - SIMT 独立的线程管理硬件来使能硬件处理方式

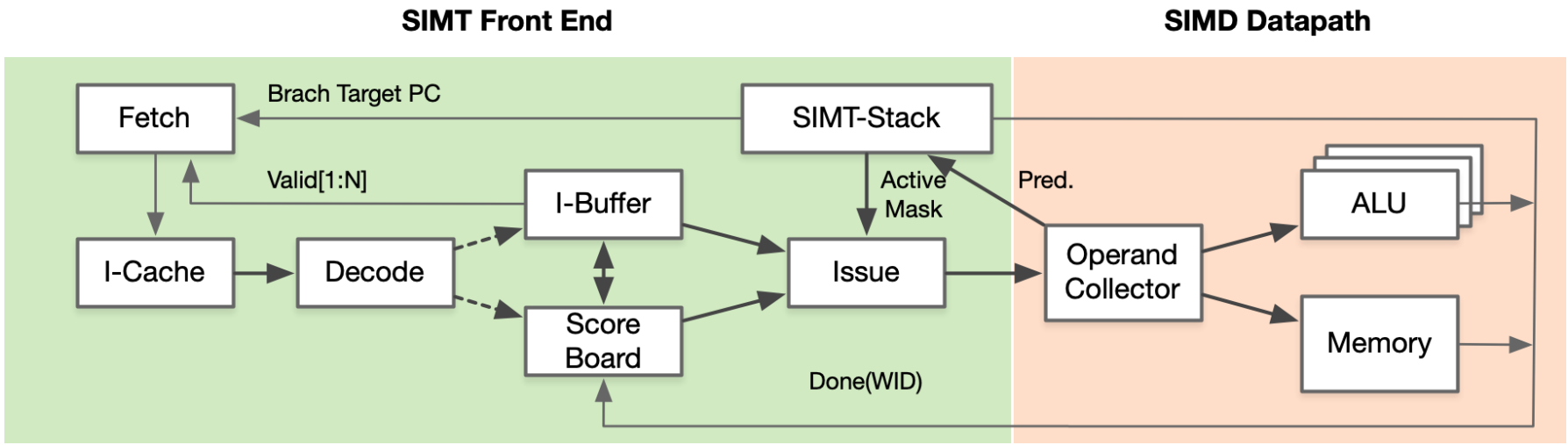
- SPMD：Single Program Multiple Data，单程序多数据的并行编程模式

SPMD执行在SIMD机器上

- SIMT：Single Instruction Multiple Thread，单指令多线程的硬件执行模式

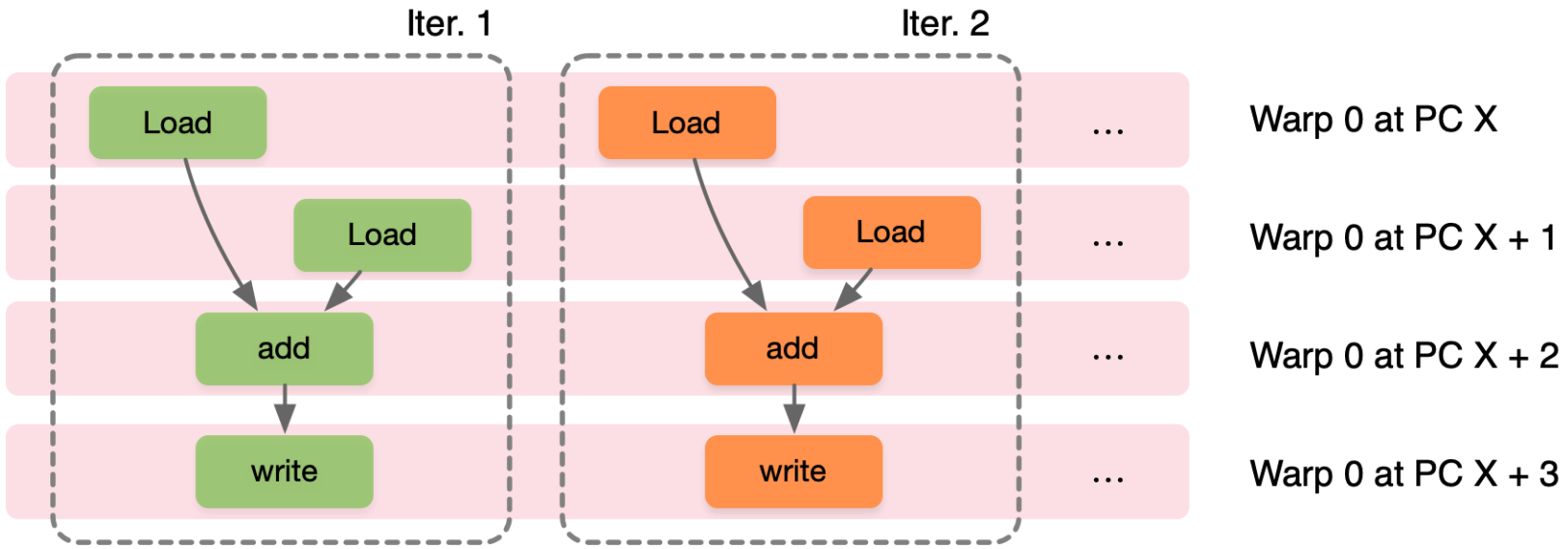
GPU SIMD (SIMT) Machine

1. **具体硬件执行 SIMD 的编程指令**
2. **并行编程模式使用 SPMD 来控制线程的方式**
   - 每个线程对不同的数据执行执行相同指令代码；每个线程都有独立的上下文；
3. **执行相同指令一组线程由硬件动态分为一组 Wrap**
   - 硬件 Warp 实际上是由 SIMD 操作形成的



GPU SIMD (SIMT) Machine

- Warp：执行相同指令的线程集合，作为GPU 的硬件 SM 调度单位，Warp里的线程执行 SIMD。
- CUDA：CUDA 的编程模式实际上是 SPMD，单程序多数据；
- SIMT：GPU 的硬件执行模式；



概念澄清

- SIMD、SIMT、DSA、SPMD 等词虽然经常放到一起讨论，但指代关系还是比较混乱，有时候指的是执行指令方式、有时候指的是硬件体系结构、有时候又是指系统编程模型。在每个层面边界也不是很清晰，于是就有了一定偷换概念的空间。

1. SIMD 和 SIMT 这里代指指令的执行方式和对应映射的硬件体系结构。
2. SPMD 这里指代一种具体的编程模型。
3. DSA 指代具体的特殊硬件架构。

4. GPU的编程模型

SIMD vs. SIMT 执行模式

- **SIMD**：单顺序的指令流执行 -> 每条指令多个数据输入同时执行
  - [VLD, VLD, VADD, VST], VLEN
- **SIMT**：标量指令的多个指令流 -> 动态地把线程按wrap分组执行
  - [LD, LD, ADD, ST], NumThreads

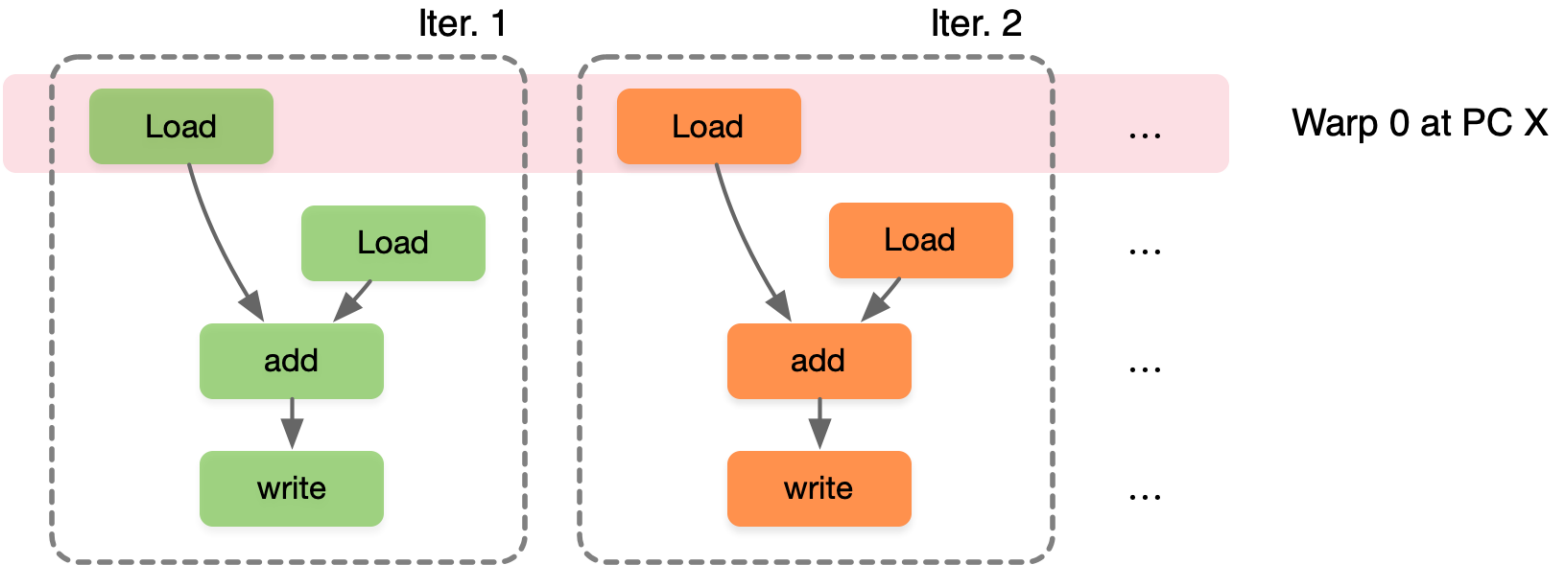
SIMD vs. SIMT 执行模式

- **SIMD**：单顺序的指令流执行 -> 每条指令多个数据输入同时执行
  - [VLD, VLD, VADD, VST], VLEN
- **SIMT**：标量指令的多个指令流 -> 动态地把线程按wrap分组执行
  - [LD, LD, ADD, ST], NumThreads

- **SIMT的优势：**
  - 无需开发者费力把数据凑成合适的矢量长度
  - 从硬件设计上解决大部分 SIMD data path 的流水编排问题
  - 线程可以独立执行，使得每个 thread 相对灵活，允许每个线程有不同的分支
  - 一组执行相同指令的线程由硬件动态组织成 warp，加快了 SIMD 的计算并行度

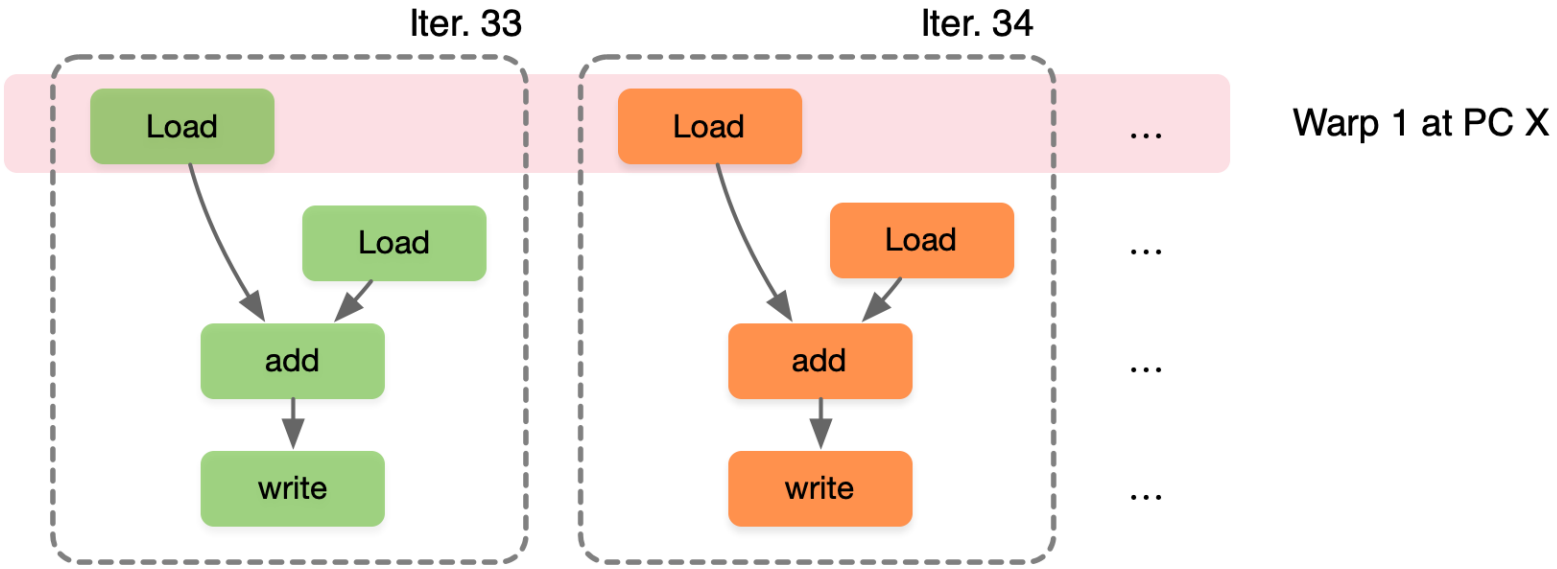
Multithreading of Warps

- GPU中线程调度的基本单位是Warp
- 假设一个 warp 包含32个线程
- 现在要进行 32K 次迭代，每个迭代一个线程 -> 1K wrap



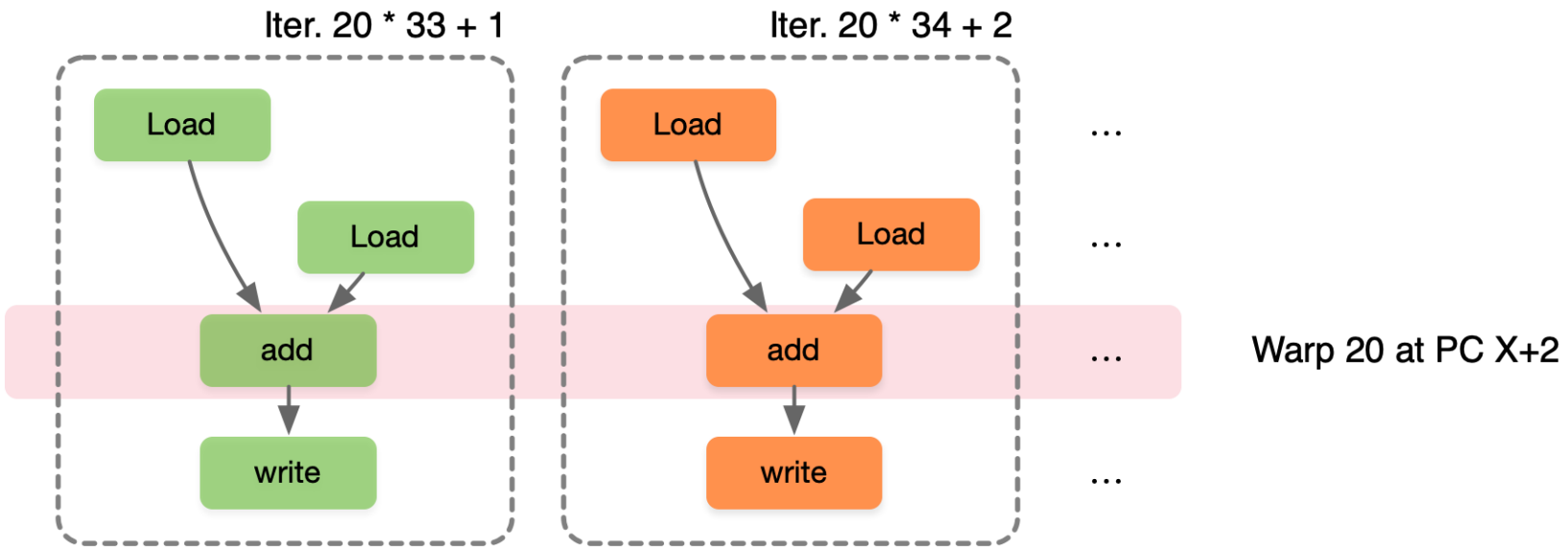
Multithreading of Warps

- GPU中线程调度的基本单位是Warp
- 假设一个 warp 包含32个线程
- 现在要进行 32K 次迭代，每个迭代一个线程 -> 1K wrap



Multithreading of Warps

- GPU中线程调度的基本单位是Warp
- 假设一个 warp 包含32个线程
- 现在要进行 32K 次迭代，每个迭代一个线程 -> 1K wrap



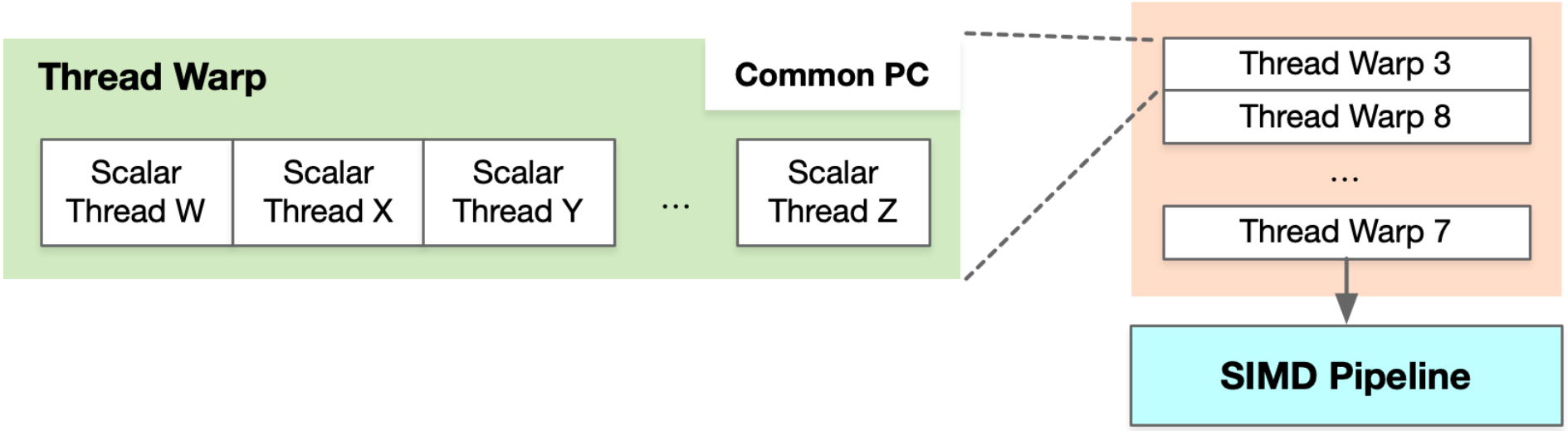
解决并行的瓶颈

程序并行执行最大的瓶颈是访存和控制流：

- SIMD 架构中单线程 CPU 通过大量控制逻辑进行超前执行、缓存、预取等机制来强行缓解瓶颈。
- SIMT 架构通过细粒度的多线程（FGMT）调度来实现访存和计算并行。

Warps 和 Warp-Level FGMT 关系

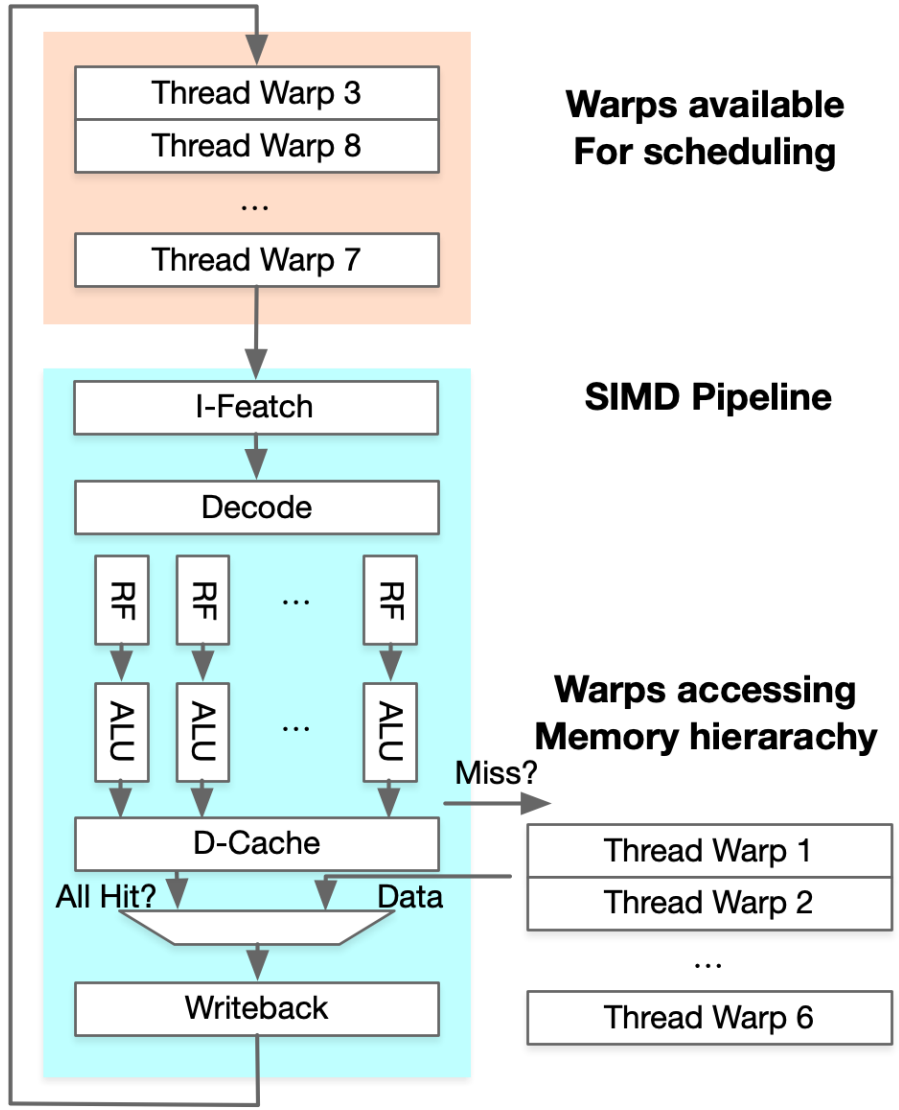
- Warp: 在不同地址数据下，执行相同指令的线程集合
- 所有线程执行相同的代码



通过 Wrap-level FGMT 隐藏延迟

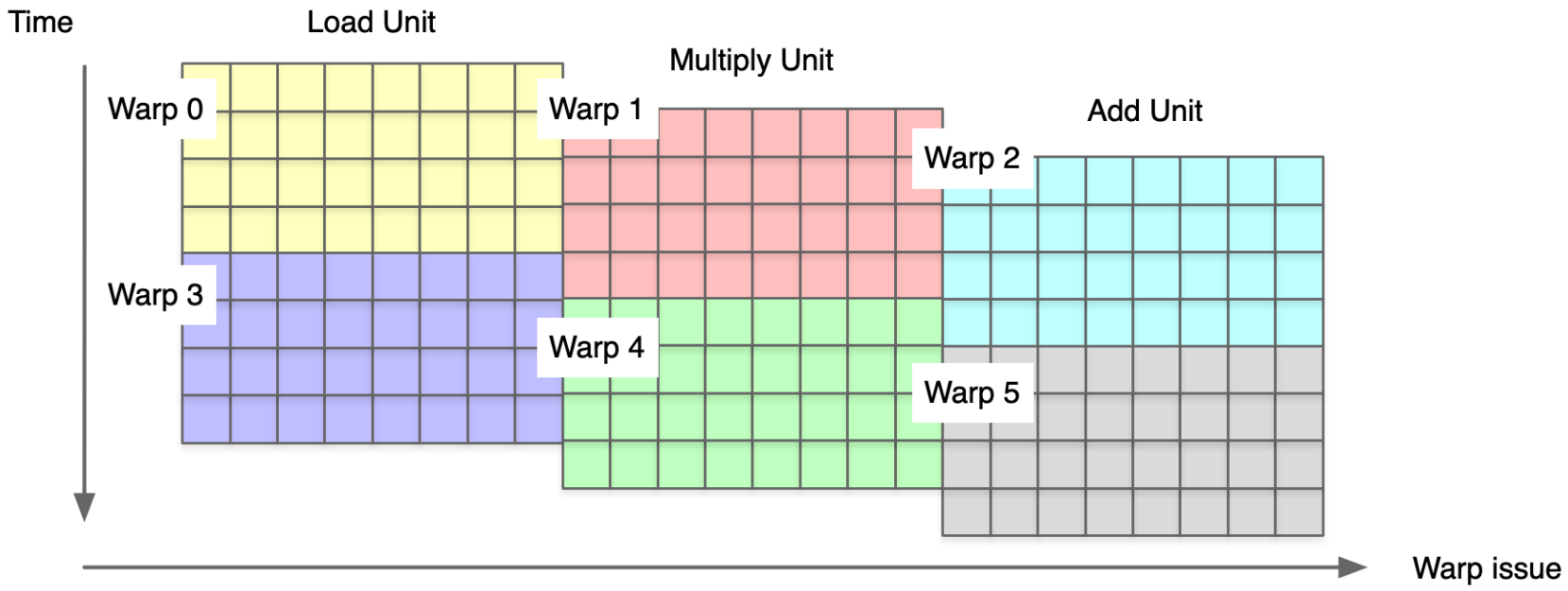
- SIMT 架构通过细粒度多线程（FGMT）实现：
  - Pipeline 中中每个线程一次一条指令
  - Warp 乱序执行以隐藏访存延迟
  - 线程寄存器值都保留在 RF 中
  - FGMT 允许长延迟

- **先访存完毕 Warp 去执行尽可能多的指令，隐藏其它 Warp 的访存时间。**

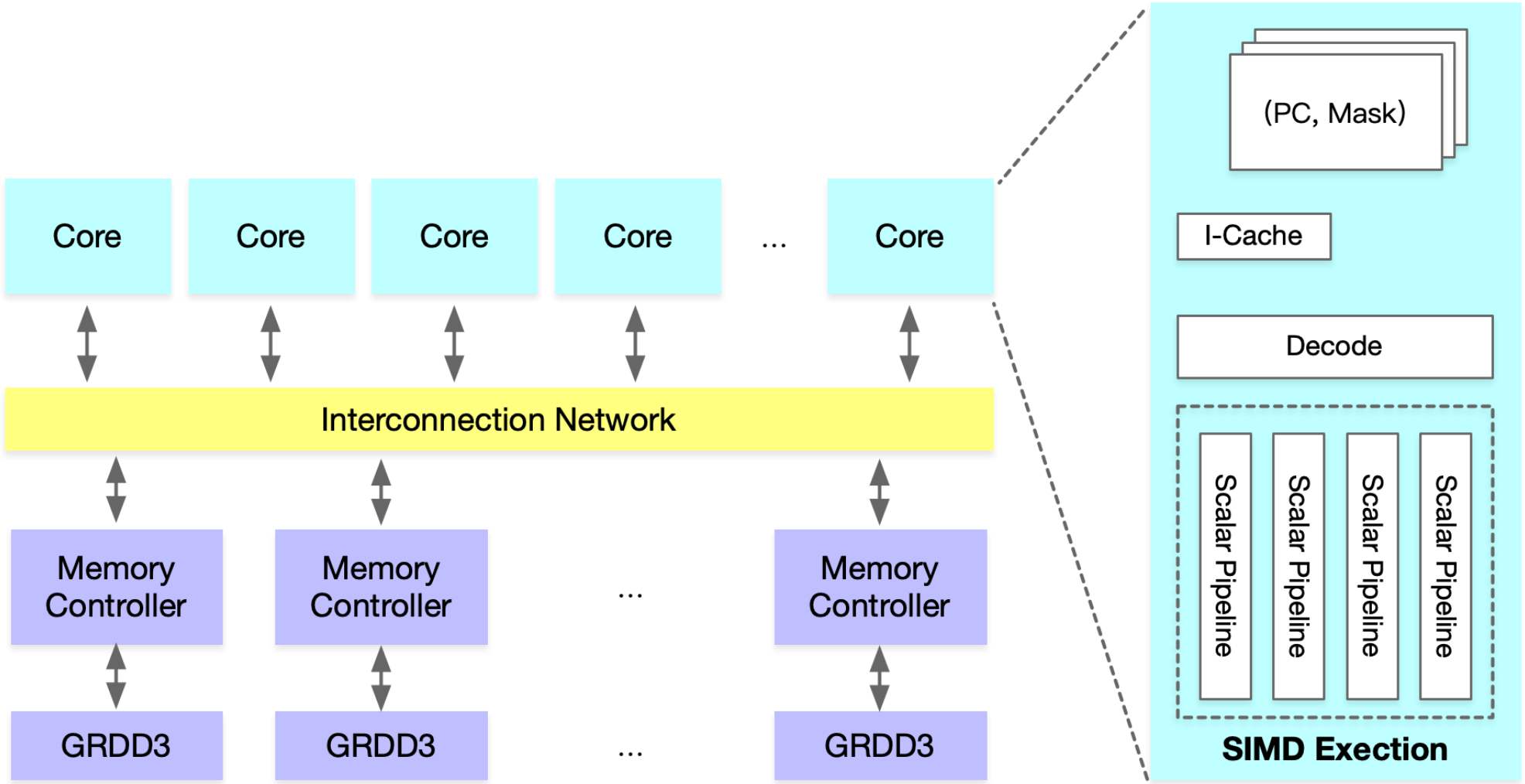


Warp 指令级并行

- SIMT 相比 SIMD 在可编程性上最根本性的优势在于硬件解决了大部分流水编排的问题：
  - 示例中每个 warp 有 32 个线程和 8 条执行通道
  - 完成 24 Operations/cycle，同时发出1 Warp/cycle



High-level view of GPU



5. 总结

执行模型： Traditional SIMD vs Warp-base SIMD(SIMT)

Traditional SIMD

- 包含单条指令执行
- ISA包含矢量/SMD指令信息
- SIMD指令中的锁同步操作，即顺序指令执行
- 编程模型是直接控制指令，没有额外线程控制，软件层面需要知道数据长度

Warp-base SIMD (SIMT)

- 以 SIMD 方式执行的多个标量线程组成
- ISA是标量，SIMD 操作可以动态形成
- 每条线程都可以单独处理，启用多线程和灵活的线程动态分组
- 本质上，是在 SIMD 硬件上实现 SPMD 编程模型

编程模型：SPMD

- **通过单个程序，控制多路数据**
- **针对不同的数据，单个线程执行相同的过程代码**
- **本质上，多个指令流执行同一个程序**
  - 每个程序：1）处理不同数据，2）在运行时可以执行不同的控制流路径
  - 在 SIMD 硬件上以 SPMD 的方式对 GPGPU 进行编程控制 -> CUDA 编程

思考

- AMD 的显卡也是有大量的计算单元和计算核心，为什么没有 SIMT 的编程模式？
- NVIDA 推出 CUDA 并遵循自定义的 SIMT 架构做对了什么？

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem