推理引擎-Kernel优化

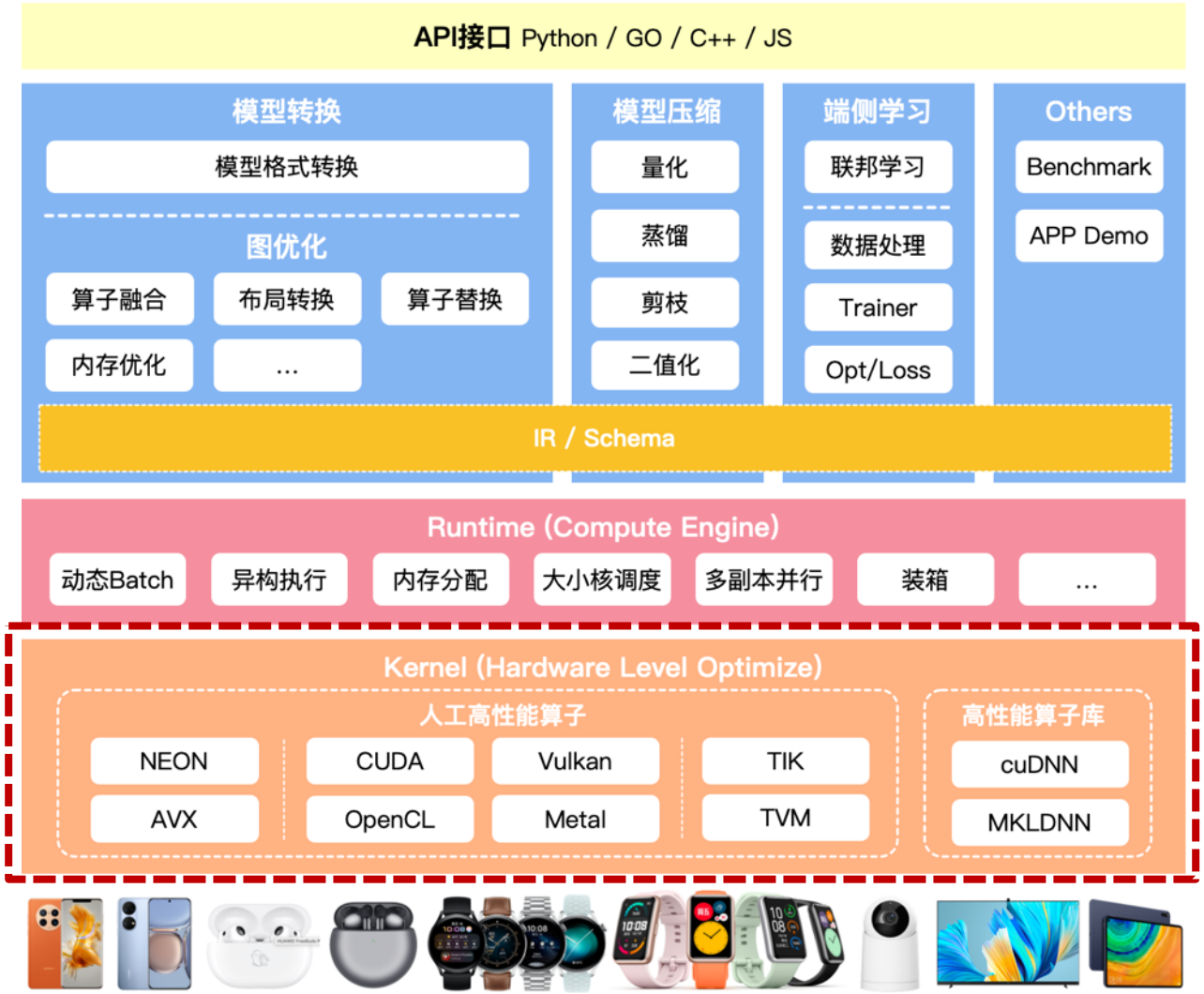
内存布局

ZOMI

Talk Overview

1. **推理系统介绍**：推理系统架构 – 推理引擎架构
2. **模型小型化**：CNN小型化结构 – Transform小型化结构
3. **离线优化压缩**：低比特量化 – 模型剪枝 – 知识蒸馏
4. **模型转换与优化**：模型转换细节 - 计算图优化
5. **Kernel 优化**
   - 算法优化 (Winograd / Strassen)
   - 内存布局 (NC1HWC0 / NCHW4)
   - 汇编优化 (指令与汇编)
   - 调度优化
6. **Runtime 优化**

推理引擎架构



高性能算子层

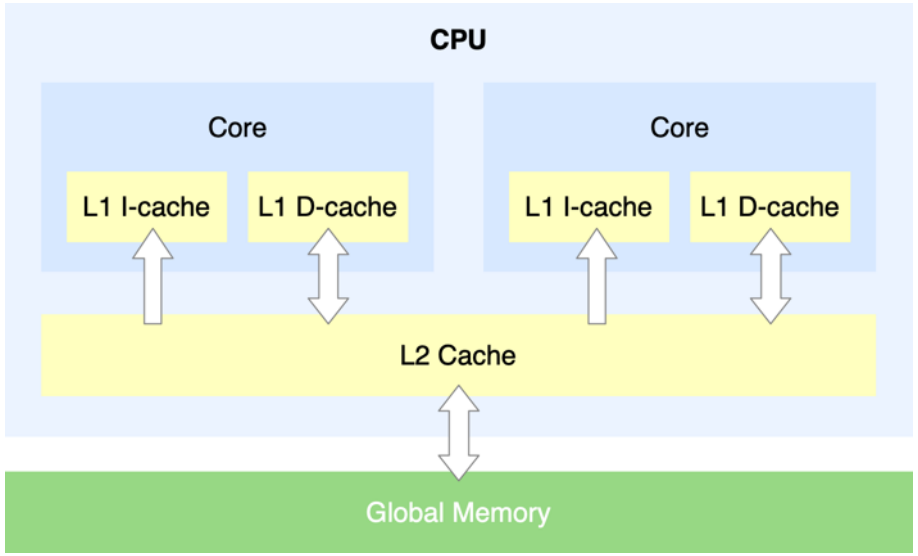
- 算子优化
- 算子执行
- 算子调度

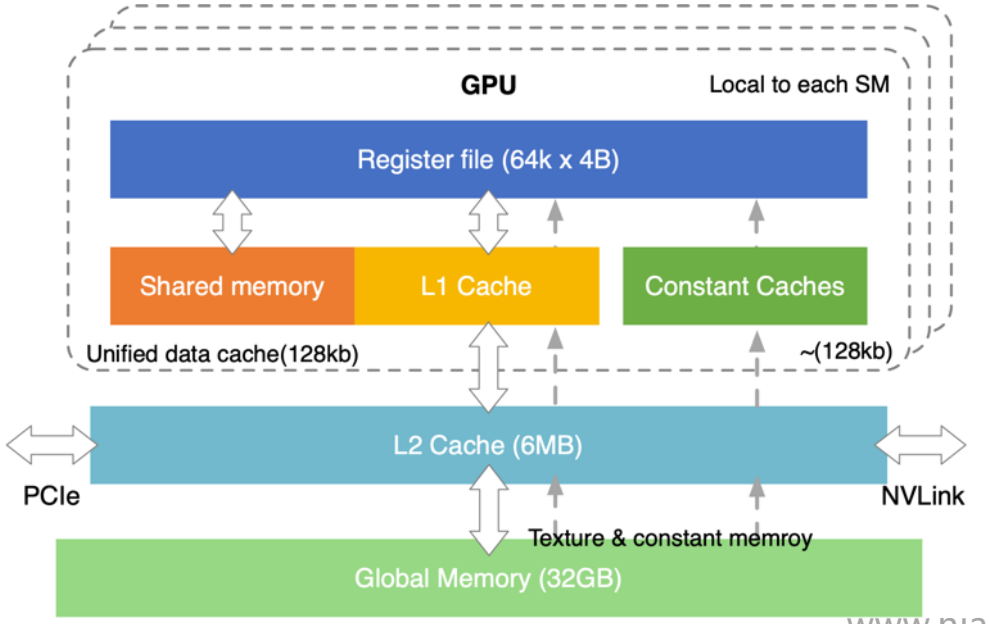
内存
Memory

基本介绍

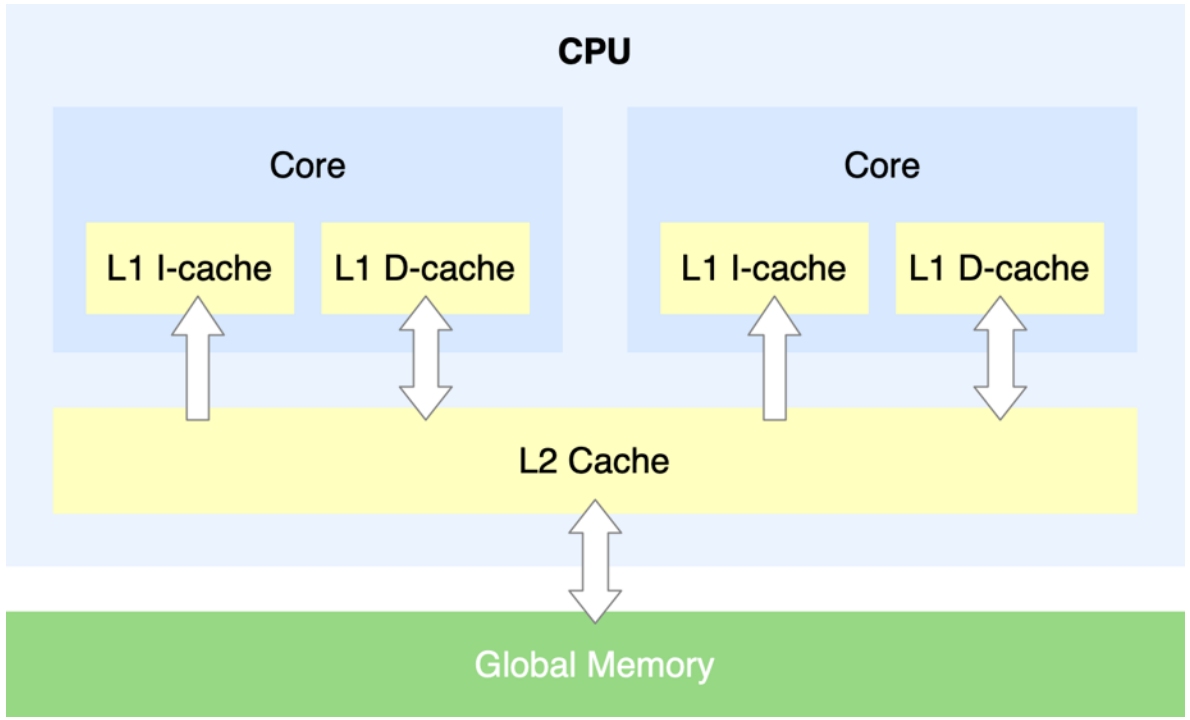
- RAM 主内存是较大存储空间，但速度较慢。CPU/NPU Cache 缓存的速度要快得多，但其规模较小，因此恰当地使用CPU/NPU Cache 缓存至关重要。

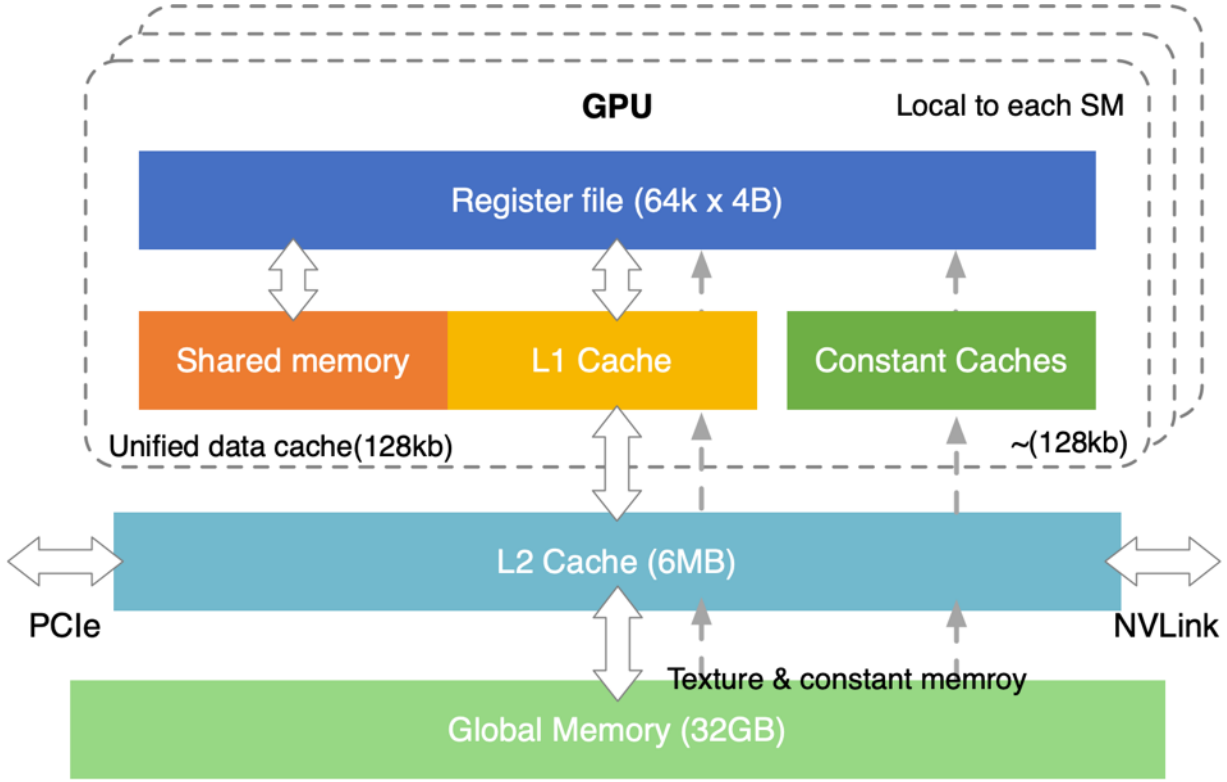
  每一次从 RAM 主内存中获取数据时，CPU/NPU 会将该数据及其邻近的数据加载到 Cache 缓存中，以便利用访问局部性（ locality of reference ）提升访问命中率。





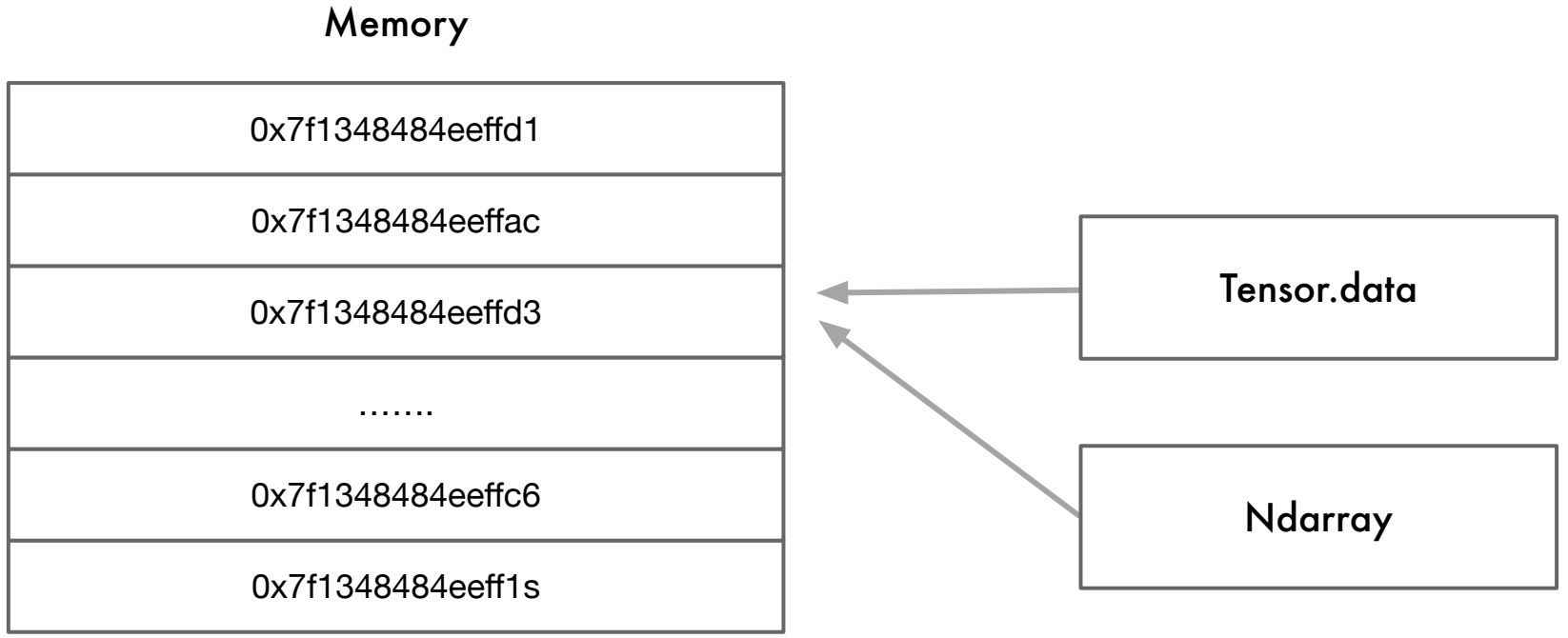
基本介绍





什么是内存对齐？

- 内存对齐和数据在内存中的位置有关。内存对齐以字节为单位进行，一个变量的内存地址如果正好等于它的长度的整数倍，则称为自然对齐。
- 在32位CPU下，一个u32的内存地址为0x00000004 ，则属于自然对齐。内存空间按照字节进行划分，理论上可以从任意地址开始读取，实际上会要求读取数据的首地址时某一个值的整数倍。



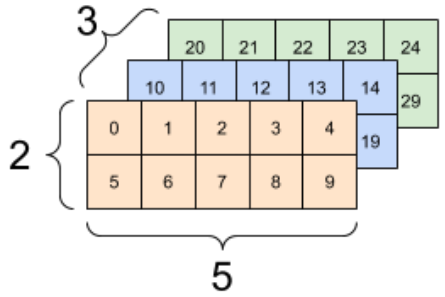
Tensor
内存布局

Tensor 值存储在内存中

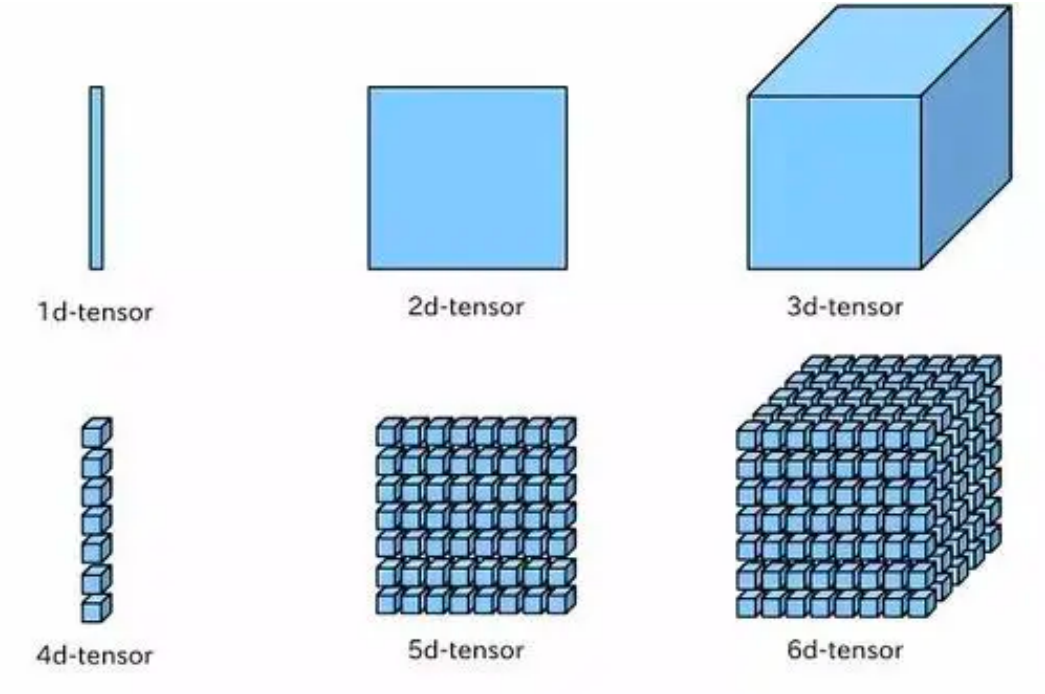
- 一个 Tensor 类的实例由一维连续的计算机内存段组成。结合Tensor 元素索引机制，可以将值映射到内存块中对应元素的位置， 而索引可以变化的范围由 Tensor 的 Shape 属性指定。 每个元素占用多少个字节以及如何解释这些字节由 Tensor 的 DataType 属性指定。

基本数据结构：Tensor 张量

- 高维数组，对标量，向量，矩阵的推广
- Tensor形状（Shape）：[3, 2, 5]
- 元素基本数据类型（Data Type）：int, float, string, etc.



Tensor 张量



图像张量化表示：

- （N，C，H，W）

自然语言张量化表示：

- （N，S，W）

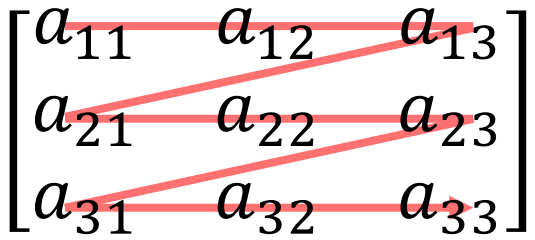
稀疏张量：

- 点云和图

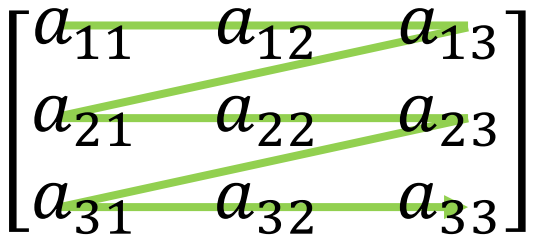
Tensor 值存储在内存中

- 一段内存本质上是连续的，有许多不同的方案可以将 N 维 Tensor 数组的项排列在一维块中。根据排列顺序的区别，又可以分为行主序和列主序两种风格，下面以最简单的 2 维情况进行举例：

Row-major order

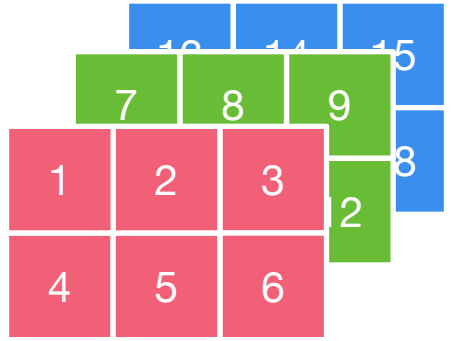


Column-major order



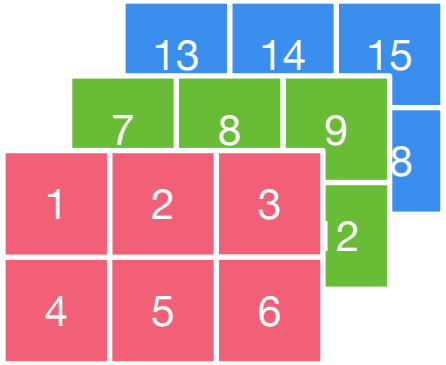
NCHW

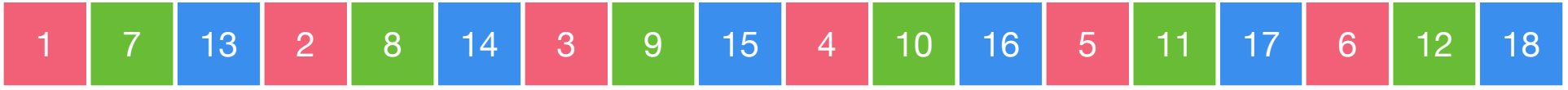
- NCHW 同一个通道的数据值连续排布，更适合需要对每个通道单独运算的操作，如 MaxPooling
- NCHW 计算时需要的存储更多，适合GPU运算，利用 GPU 内存带宽较大并且并行性强的特点，其访存与计算的控制逻辑相对简单：



NHWC

- NHWC 其不同通道中的同一位置元素顺序存储，因此更适合那些需要对不同通道的同一数据做某种运算的操作，比如“Conv1x1”
- NHWC 更适合多核CPU运算，CPU 内存带宽相对较小，每个数据计算的时延较低，临时空间也很小，有时计算机采取异步的方式边读边算来减小访存时间，因此计算控制灵活且复杂





框架的默认选择

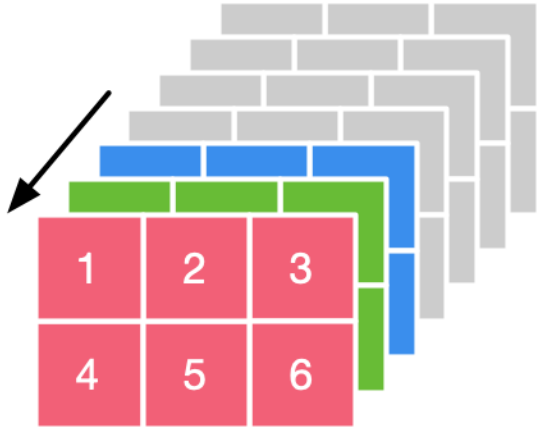
- 由于数据只能线性存储，因此这四个维度有对应的顺序。不同深度学习框架会按照不同的顺序存储特征图数据：
  - 以 NPU/GPU 为基础的 PyTorch 和 MindSpore 框架默认使用 NCHW 格式，排列顺序为[Batch, Channels, Height, Width]
  - Tensorflow 采用了 NHWC，排列顺序为[Batch, Height, Width, Channels]，何面向移动端部署 TFLite 只采用 NHWC 格式

NCHW 和 NHWC

- 对于”NCHW”而言，其同一个通道的像素值连续排布，更适合那些需要对 **每个通道单独做运算** 的操作，比如”MaxPooling”。
- 对于”NHWC”而言，其不同通道中的同一位置元素顺序存储，因此更适合那些需要对 **不同通道的同一像素做某种运算** 的操作，比如“Conv”。

NCHWX [Batch, Channels/X, Height, Width, X=4，32或64]

- 由于典型的卷积神经网络随着层数的增加，其特征图在下采样后的长和宽逐渐减小，但是channel 数随着卷积 filter 个数不断增大也越来越大，如 channel=128/256 等。为了充分利用有限的矩阵计算单元，进行对 Channel 维度的拆分称为一种很好的利用空间的方案。



NCHWX

1. NCHWX 的格式能够更好的适配SIMT，其中 NCHW4 可以针对 Arm int8 数据类型，利用 CUDA 的 dp4a 模块进行计算；
2. 而 NCHW32 和 NCHW64 分别针对int8和int4数据类型，能够更好的利用 CUDA 的 Tensor core计算单元进行计算；
3. 对 Cache 更友好，减少 Cache miss；Kernel 实现层面易进行 Padding，减少边界分支判断，代码逻辑简单。

引用

1. https://discuss.pytorch.org/t/how-to-know-the-memory-allocated-for-a-tensor-on-gpu/28537
2. https://towardsdatascience.com/optimize-pytorch-performance-for-speed-and-memory-efficiency-2022-84f453916ea6
3. https://oneapi-src.github.io/oneDNN/dev_guide_understanding_memory_formats.html