MEGVII 旷视

深度学习框架
MegEngine CUDA Int4 推理详解

王彪 | wangbiao@megvii.com

MEGVII 旷视

目录

Contents

MEGVII 旷视

01

背景 & 动机

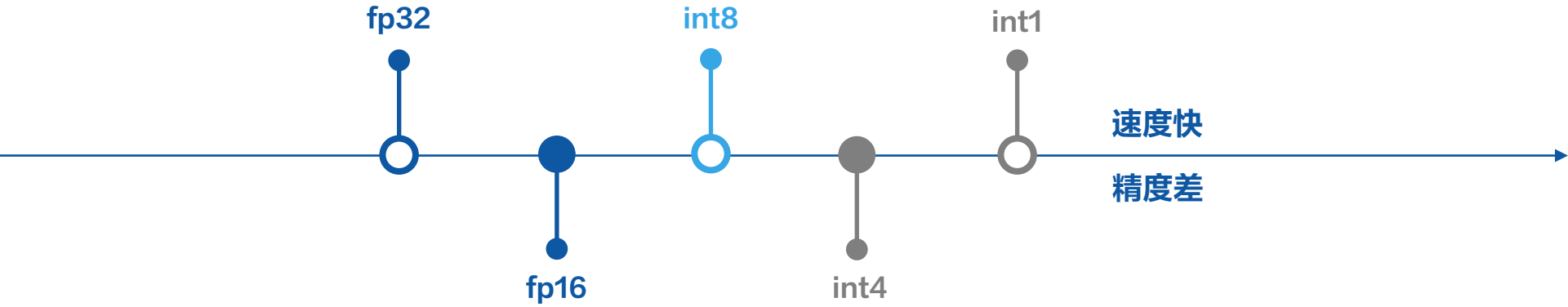
MegEngine 介绍

MEGVII 旷视

训练推理一体、多平台高效推理、高效训练

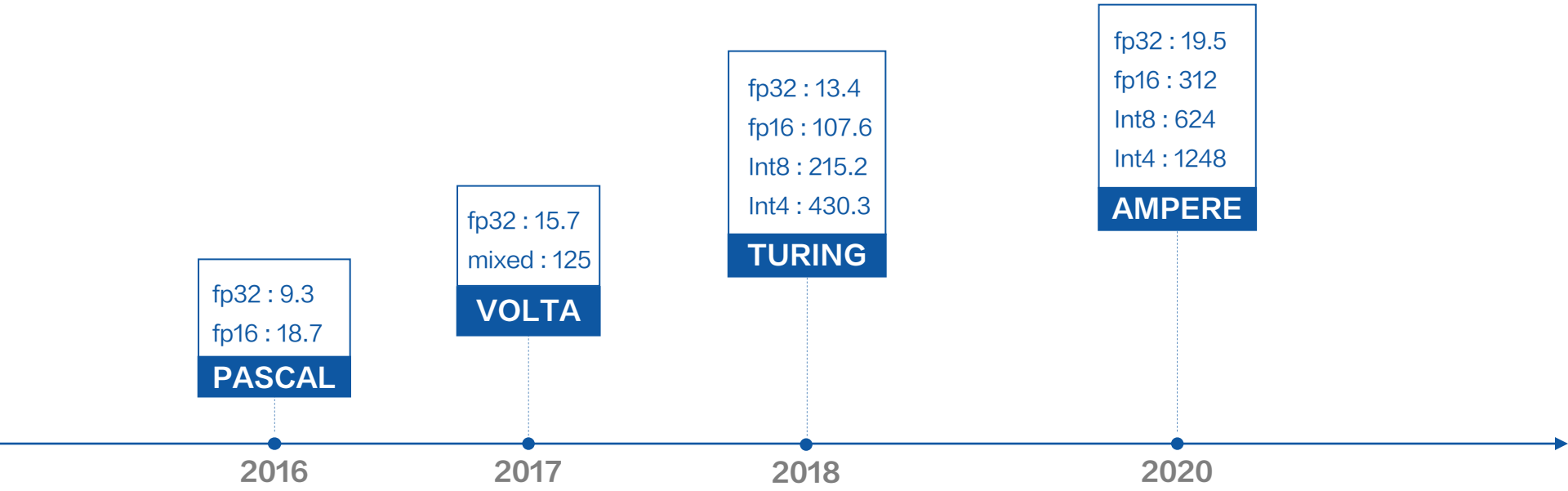
int4 量化模型落地的现状

MEGVII 旷视



更低 bit 量化模型落地是趋势

MEGVII 旷视



NVIDIA GPU 架构演进

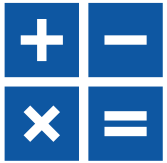
业务需求的多样性

MEGVII 旷视

速度/内存敏感型业务

MEGVII 旷视

高内存占用



高计算量

高精度

模型量化

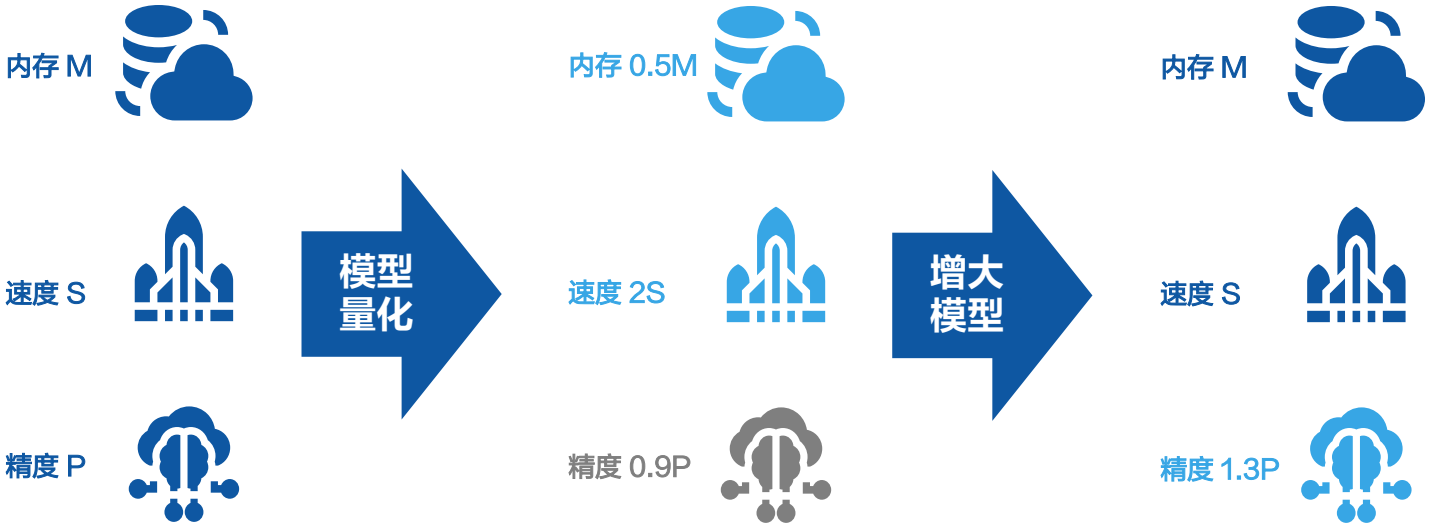
压缩内存占用

提升速度

损失精度

精度敏感型业务

MEGVII 旷视



int4 量化模型落地的现状

MEGVII 旷视

模型开发人员

- 不知道 Int4 量化有什么需要注意的地方，精度够不够
- 模型 Int4 量化之后在真实硬件下到底会不会快

工程优化人员

- 市面上少有 Int4 推理的可参考的例子，担心有坑
- Int4 在工程实现上具有一定难度，是否值得投入大量人力

MegEngine 源码开源 CUDA Int4 实现

MEGVII 旷视

模型开发人员

- MegEngine CUDA Int4 量化实践经验参考
- 利用 MegEngine 实测自己的模型 Int4 量化之后的性能表现

工程优化人员

- MegEngine CUDA Int4 模型推理流程经验参考
- MegEngine 源码开源 CUDA Int4 实现，供大家参考
- 获得更优的模型推理速度或精度，提升产品效果

硬件支持

MegEngine 支持
CUDA Int4 量化推理

业务需求

MEGVII 旷视

02

如何尽量提高量化精度

速度和精度之间的平衡艺术

MEGVII 旷视

量化方法选择

核心算子的
量化方案

整个模型的
量化方案

软硬结合经验

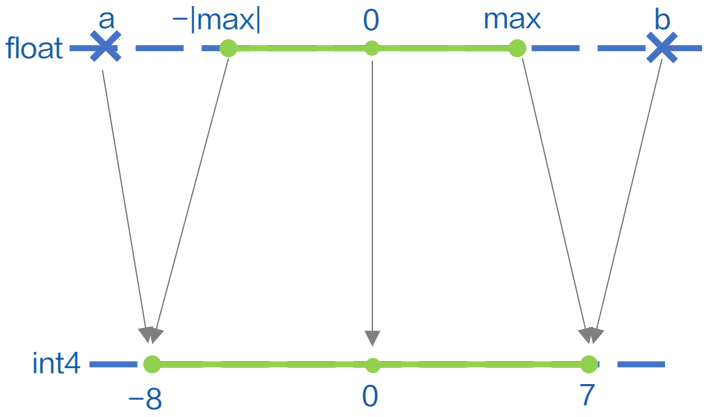
量化方法的选择

MEGVII 旷视

对称量化

Quantize : uint = round(float / S)

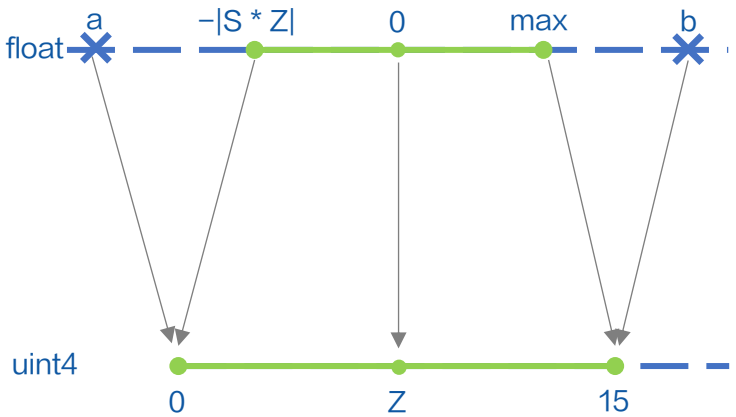
De-quantize : float’ = S * uint



非对称量化

Quantize : int = round(float / S) + Z

De-quantize : float’ = S * (int – Z)

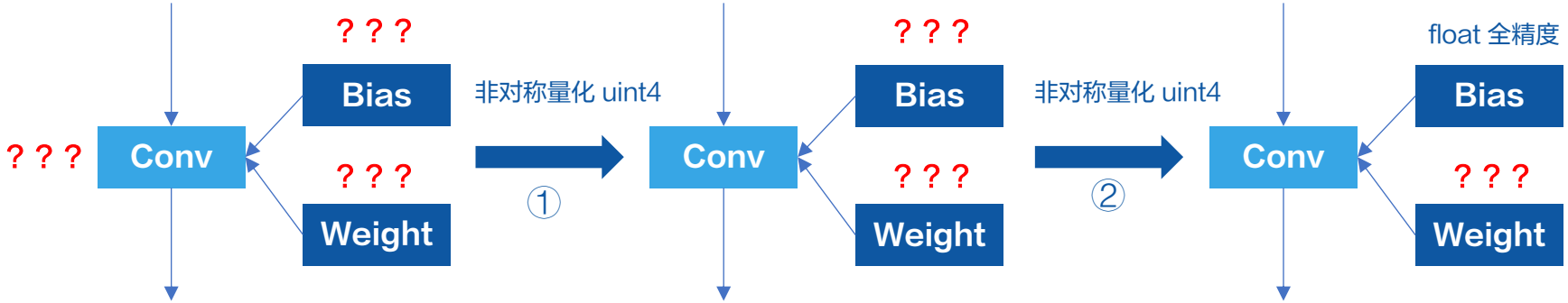


核心算子的量化方案

MEGVII 旷视

① 由于 relu 的影响， feature map 往往是非对称的，所以采用非对称量化比较合适

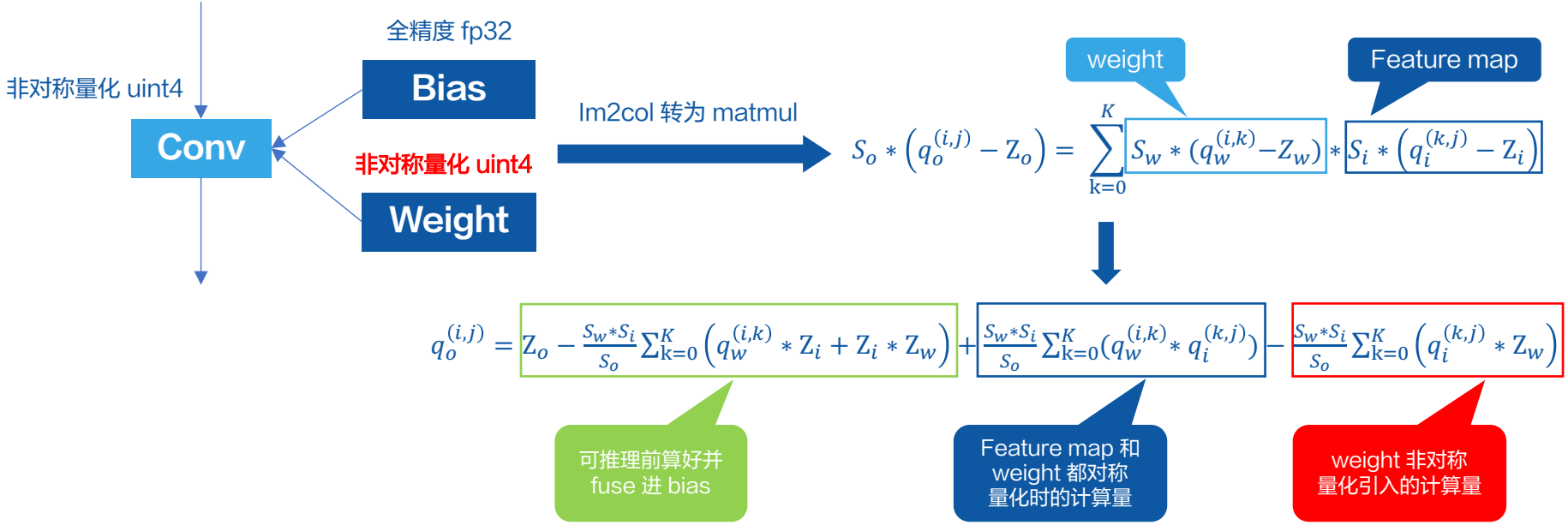
② 由于激活函数也会被融合进卷积中，同时为了提高精度， bias 使用全精度表示比较合适



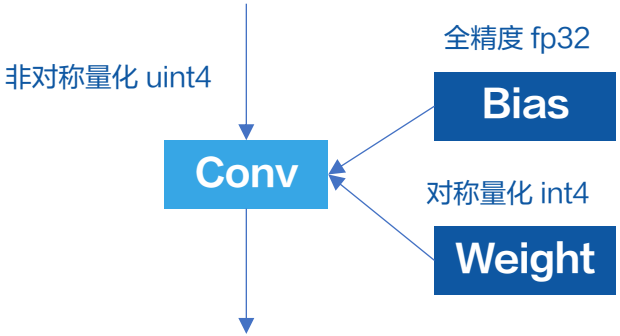
核心算子的量化方案

MEGVII 旷视

Weight 取非对称量化会增加计算量



核心算子的量化方案



Im2col 转为 matmul

量化公式：$r_\alpha^{(i,j)} = S_\alpha * (q_\alpha^{(i,j)} - Z_\alpha)$

weight

feature map

$$S_o * \left(q_o^{(i,j)} - Z_o\right) = \sum_{k=0}^{K} S_w * q_w^{(i,k)} * S_i * \left(q_i^{(k,j)} - Z_i\right)$$

$$q_o^{(i,j)} = Z_o - \frac{S_w * S_i}{S_o} \sum_{k=0}^{K} \left(q_w^{(i,k)} * Z_i\right) + \frac{S_w * S_i}{S_o} \sum_{k=0}^{K} \left(q_w^{(i,k)} * q_i^{(k,j)}\right)$$

可推理前算好并 fuse 进 bias

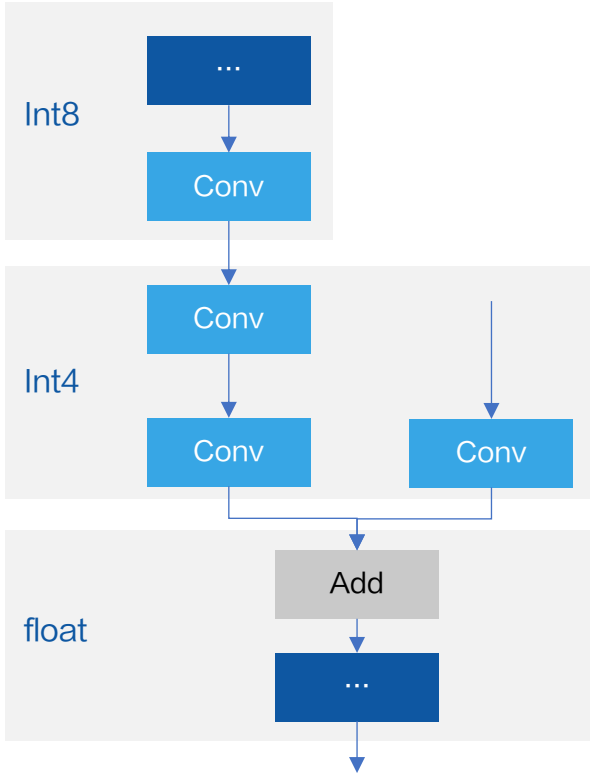
Feature map 和 weight 都对称量化时的计算量

整个模型的量化方案

MEGVII 旷视

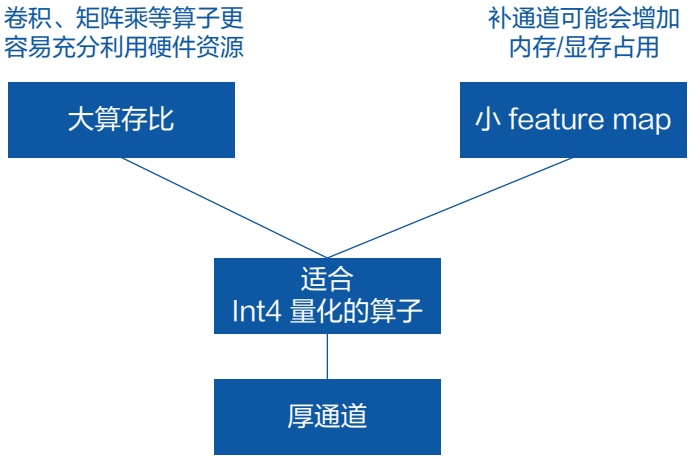
Int4 量化 = 整个模型计算占比大的算子都需要被量化为 Int4？

- 图像分类模型
  - 输入类型一般是 uint8/int8
    - 图像数据
  - 输出类型一般是 float
    - 目标分数
  - 中间部分一般为 Int4/Int8
    - **寻找精度和速度的平衡**

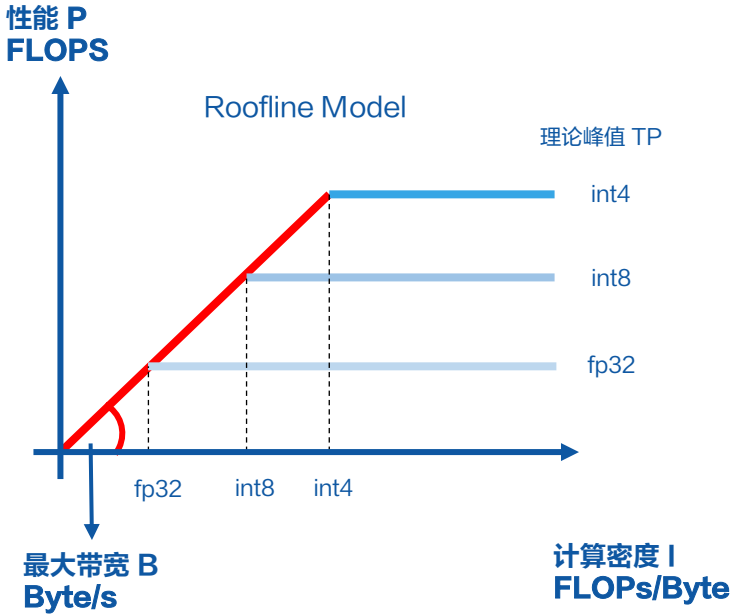


软硬结合经验

Int4 数据类型对应 NCHW64 的存储格式要求数据通道被 64 整除
框架内部会将通道补齐 64 的倍数



通道数最好是 64 的倍数，而且将通道补齐到 64 的倍数，
速度不变，精度可能更好。



MEGVII 旷视

03

如何优化 Int4 推理性能

如何优化 Int4 推理性能

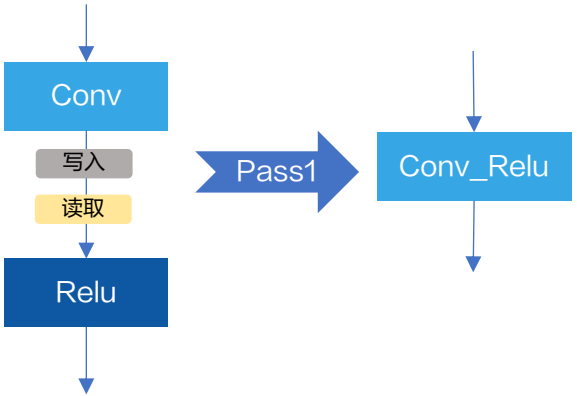
MEGVII 旷视

算子融合

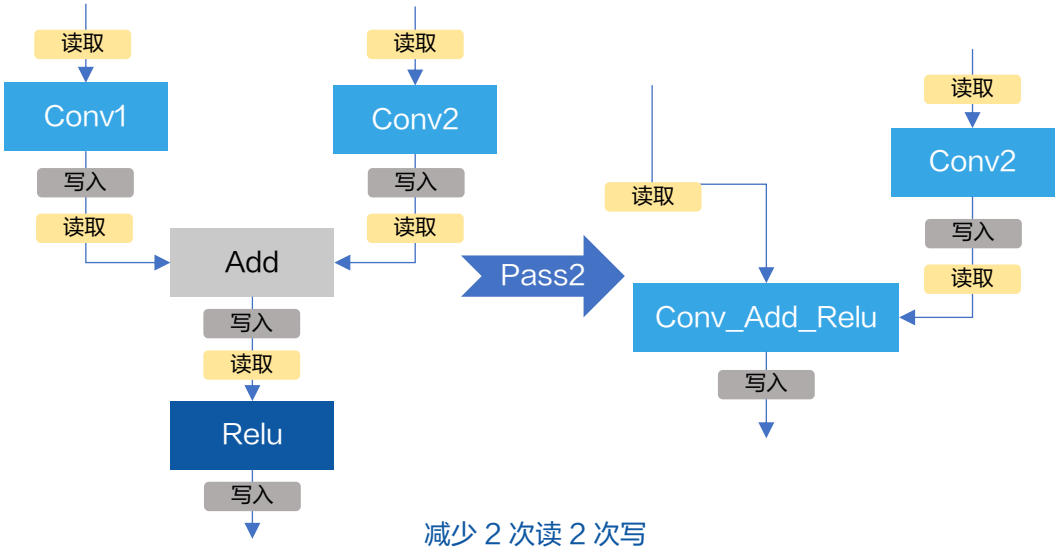
张量存储格式

核心算子开发

算子融合

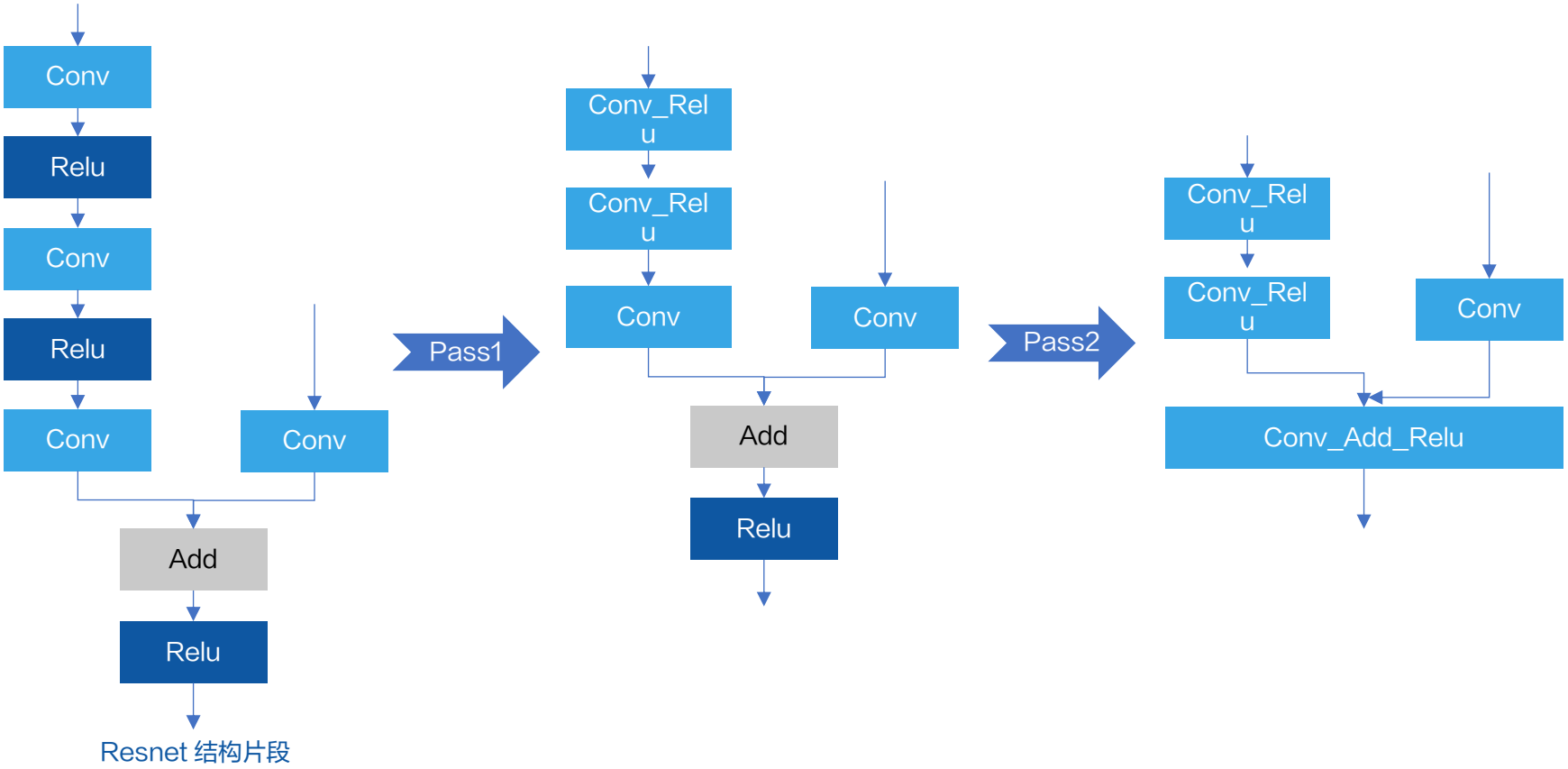


减少 1 次读 1 次写



减少 2 次读 2 次写

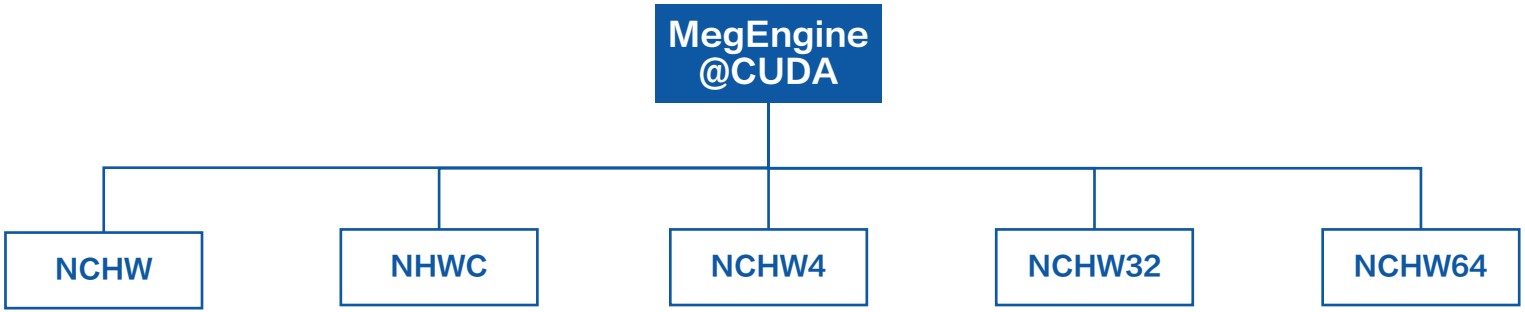
算子融合


Conv
Relu
Conv
Relu
Conv
Conv
Add
Relu
Resnet 结构片段
Pass1
Conv_Relu
Conv_Relu
Conv
Conv
Add
Relu
Pass2
Conv_Relu
Conv_Relu
Conv
Conv_Add_Relu

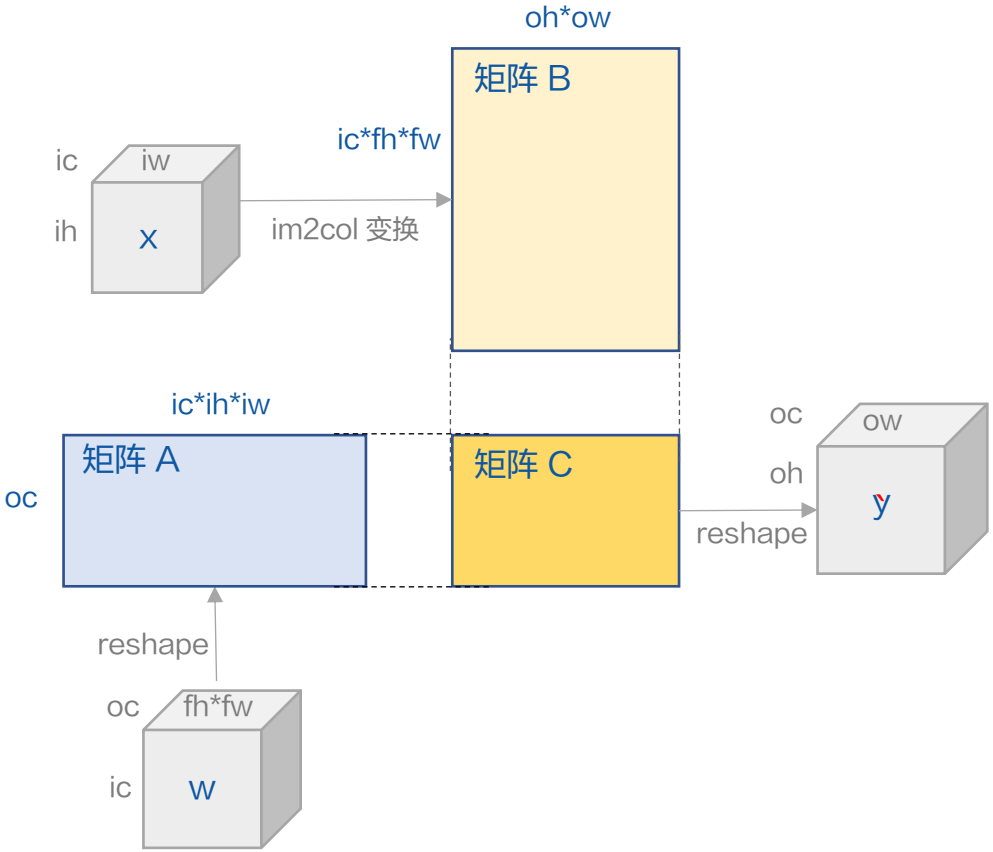
MEGVII 旷视

MegEngine 中的张量存储格式

- 通常张量是用 NCHW 格式存储的
- 硬件特征如 SIMD 和 TensorCore 决定了对齐的格式往往更容易充分利用硬件资源。如 cudnn int8 TensorCore 要求输入数据是 NCHW32(int8) 格式



核心算子开发

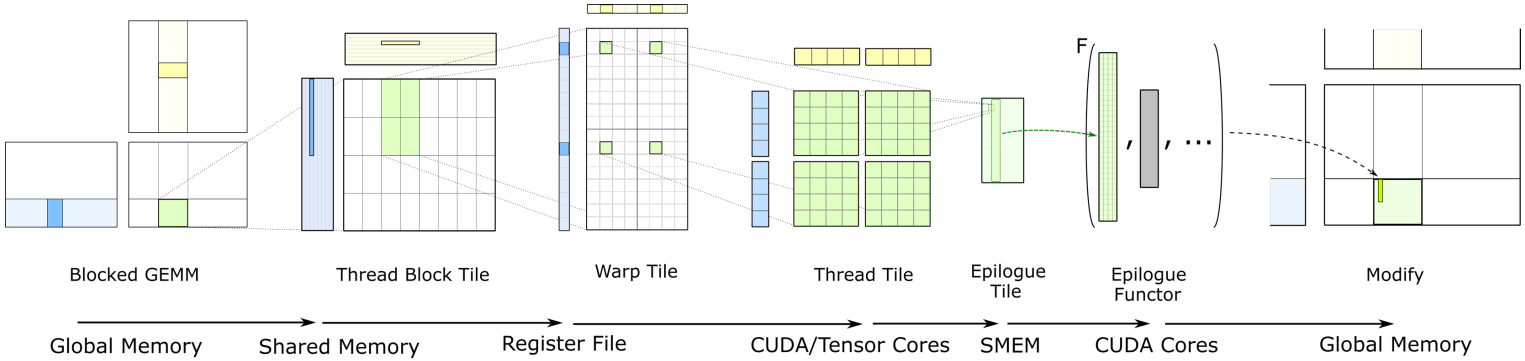


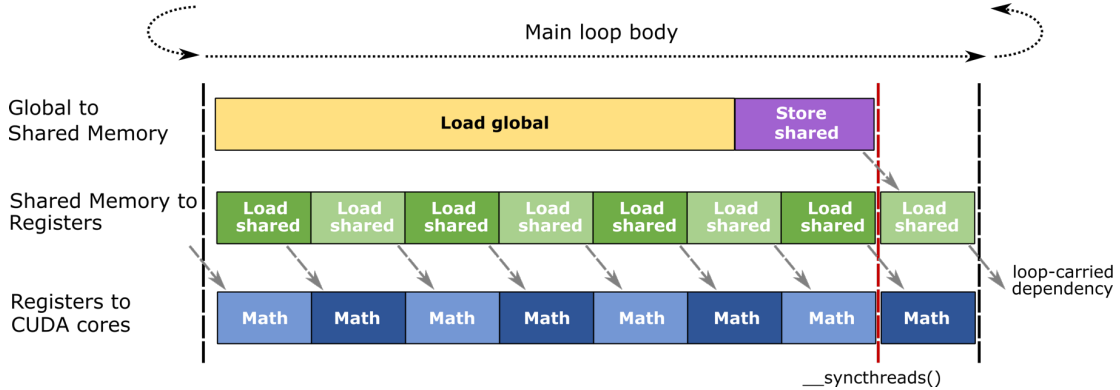
Implicit Gemm 计算 Convolution

$$y(oc, oh, ow) = \sum_{ic=0}^{IC} \sum_{fh=0}^{FH} \sum_{fw=0}^{FW} x(ic, ih, iw) * w(oc, ic, fh, fw)$$

```
0: GEMM_M = OC
1: GEMM_N = OH * OW
2: GEMM_K = IC * FH * FW
3: for i in range(GEMM_M):
4:     oc = i
5:     for j in range(GEMM_N):
6:         accumulator = 0
7:         oh = j / OW
8:         ow = j % OW
9:         for k in range(GEMM_K):
10:            ic = k / (FH * FW)
11:            k_res = k % (FH * FW)
12:            fh = k_res / FW
13:            fw = k_res % FW
14:            ih = oh * stride_h − pad_h + fh
15:            iw = ow * stride_w − pad_w + fw
16:            accumulator = accumulator + x(ic, ih, iw) * w(oc,
ic, fh, fw)
17:        y(oc, oh, ow) = accumulator
```

核心算子开发





https://github.com/NVIDIA/cutlass/blob/master/media/docs/efficient_gemm.md

核心算子开发

MEGVII 旷视

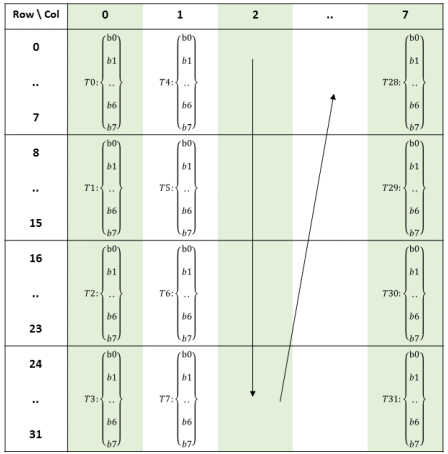
适用于 int4 数据类型的 TensorCore PTX 指令

mma.m8n8k32 所要求的数据排布

Fragment A: s4/u4

Fragment B: s4/u4

Fragment C: s32



Fragment B

Row\Col	0 .. 7	8 ... 15	16 ... 23	24 ... 31
0	T0:{a0, a1, ..., a7}	T1:{a0, a1, ..., a7}	T2:{a0, a1, ..., a7}	T3:{a0, a1, ..., a7}
1	T4:{a0, a1, ..., a7}	T5:{a0, a1, ..., a7}	T6:{a0, a1, ..., a7}	T7:{a0, a1, ..., a7}
2				
..				
7	T28:{a0, a1, ..., a7}	T29:{a0, a1, ..., a7}	T30:{a0, a1, ..., a7}	T31:{a0, a1, ..., a7}

Fragment A

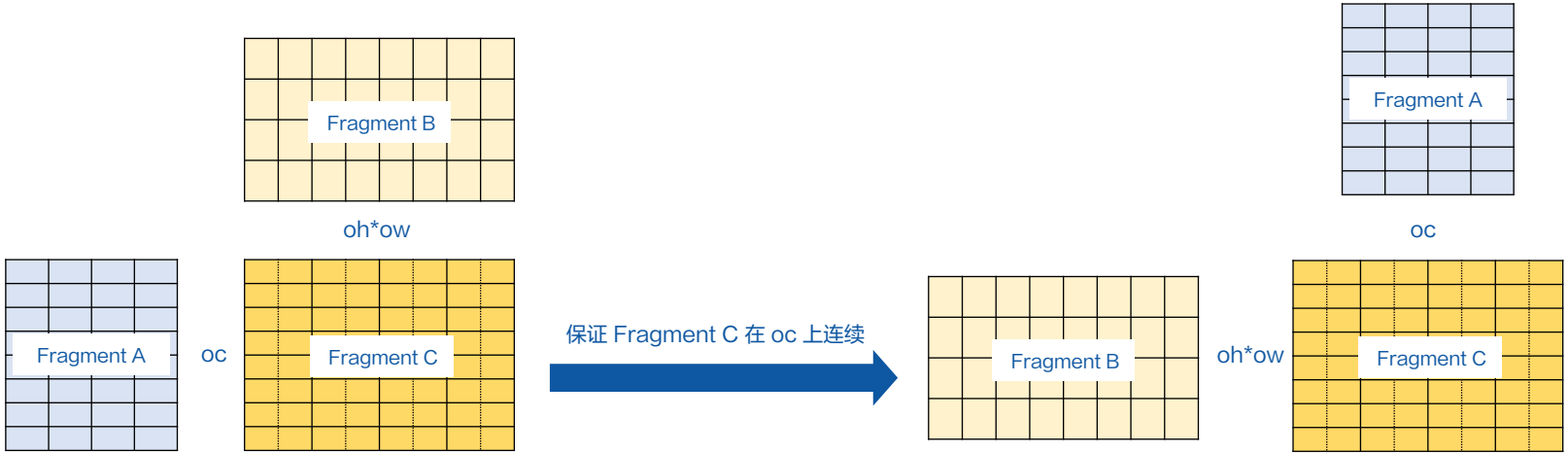
Row\Col	0 1	2 3	4 5	6 7
0	T0: {c0, c1}	T1: {c0, c1}	T2: {c0, c1}	T3: {c0, c1}
1	T4: {c0, c1}	T5: {c0, c1}	T6: {c0, c1}	T7: {c0, c1}
2				
..				
7	T28: {c0, c1}	T29: {c0, c1}	T30: {c0, c1}	T31: {c0, c1}

Fragment C

MEGVII 旷视

global memory 访存经验准则

- 相同 warp 中的连续线程访问连续地址，会触发访存合并
- 一个 memory transaction 为 128 位，相同数据量情况下访存位宽越高，所需的访存指令越少

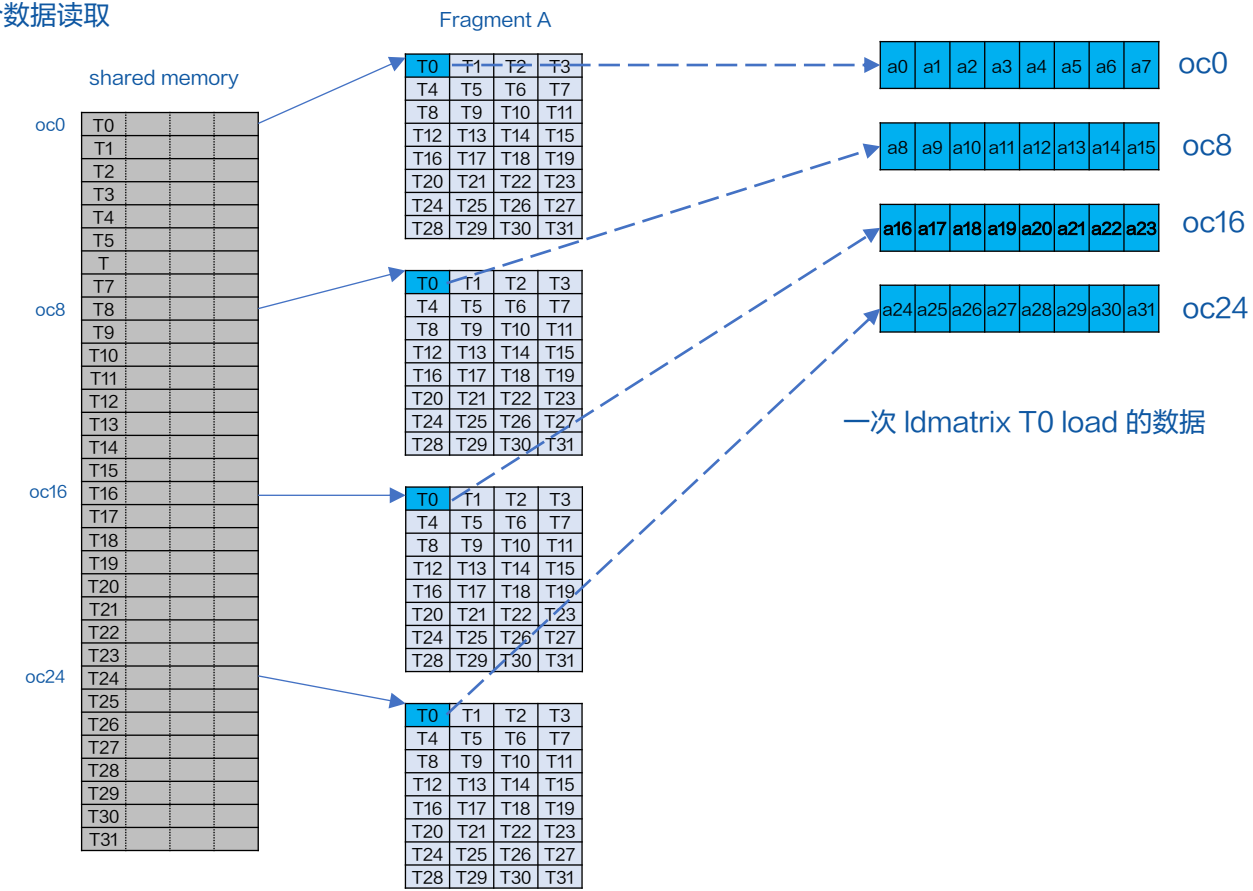


对 TensorCore 矩阵乘指令的特殊处理

核心算子开发

MEGVII 旷视

ldmatrix 指令数据读取

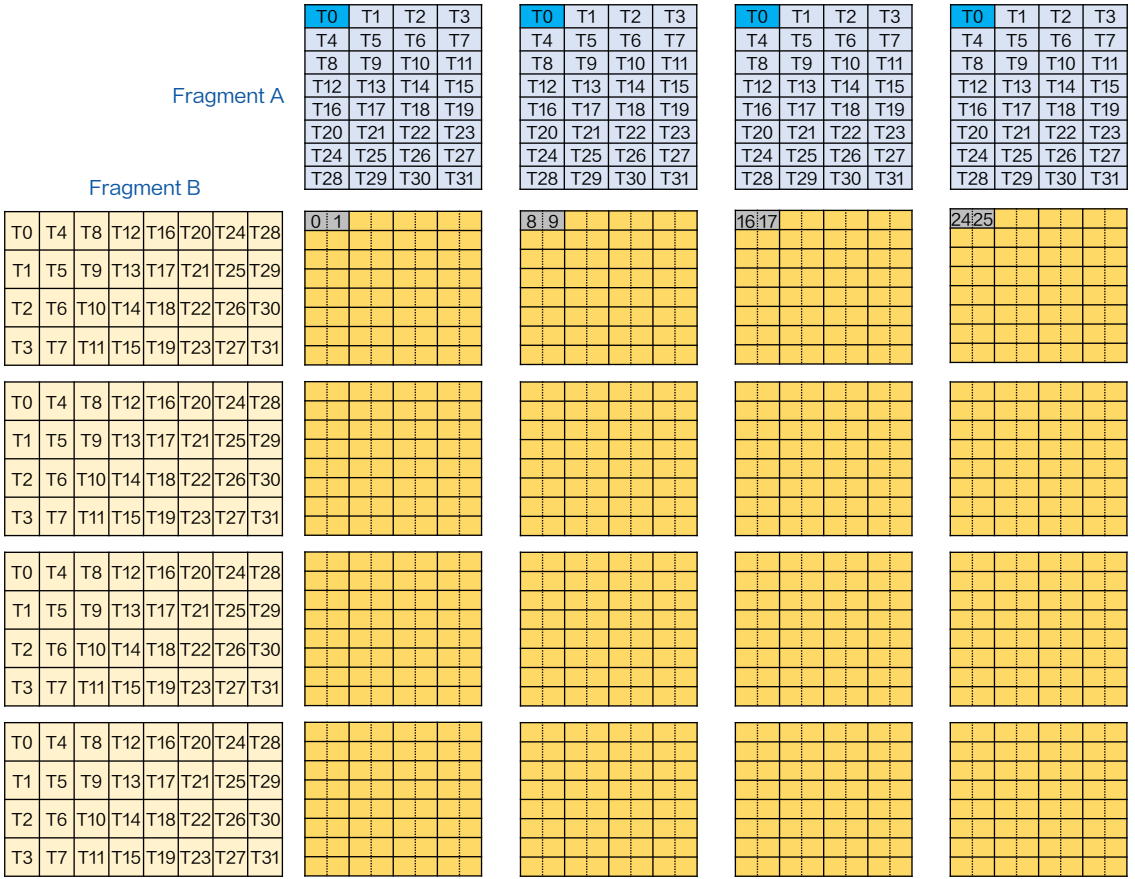


核心算子开发

MEGVII 旷视

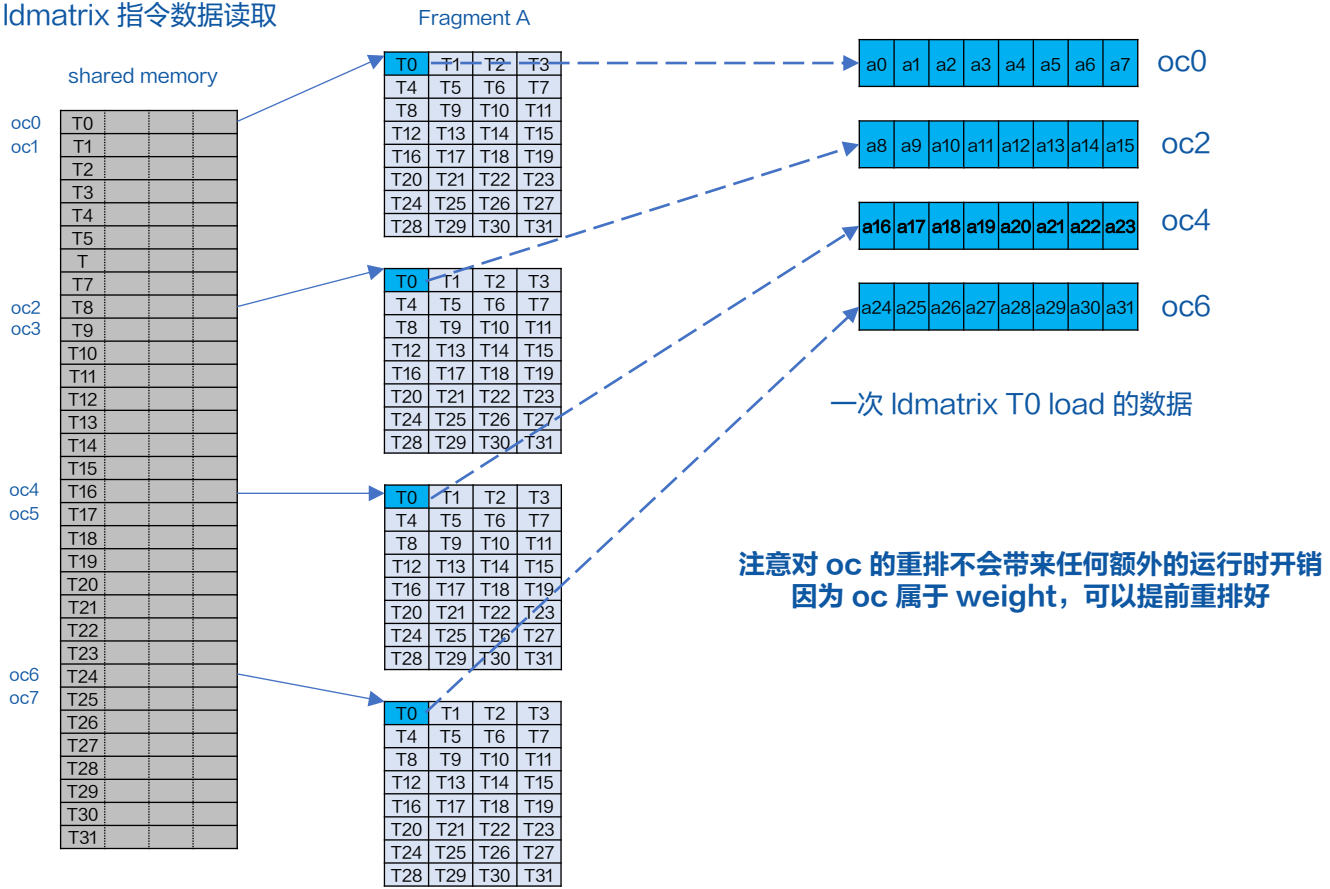
2 次 ldmatrix，16 次 mma

- 线程持有的结果数据并不连续，写回之前需要将多个线程持有的结果合并，造成冗余计算和访存



核心算子开发

重排 oc 之后的 ldmatrix 指令数据读取



注意对 oc 的重排不会带来任何额外的运行时开销
因为 oc 属于 weight，可以提前重排好

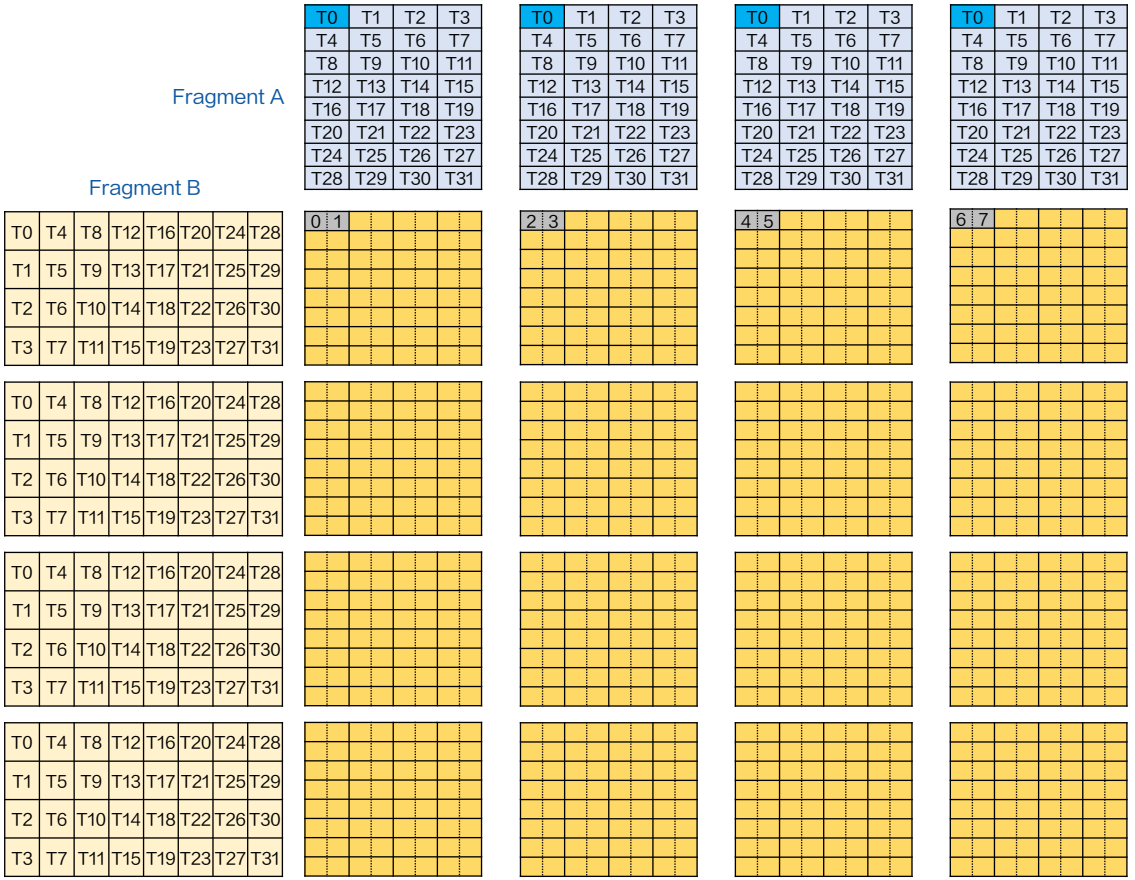
核心算子开发

MEGVII 旷视

2 次 ldmatrix，16 次 mma

- ✓ 线程持有的结果数据是连续的
- ✓ 相邻线程之间写回的地址是连续的，触发合并访存

- 只计算了 32 个连续的 oc
- 每个线程 32 位写回

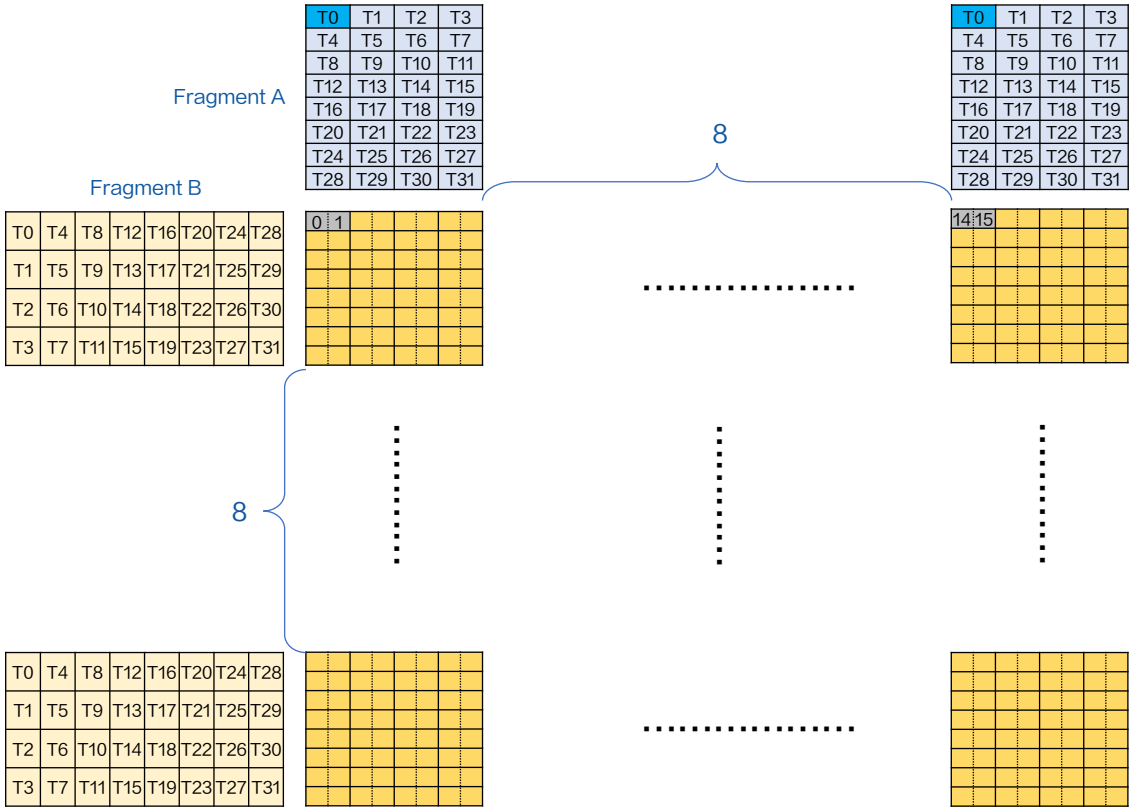


核心算子开发

MEGVII 旷视

4 次 ldmatrix，64 次 mma

- ✓ 线程持有的结果数据是连续的
- ✓ 相邻线程之间写回的地址是连续的，触发合并访存
- ✓ 计算了所有的 64 个连续的 oc，算存比更高
- ✓ 每个线程 64 位写回，所需指令更少

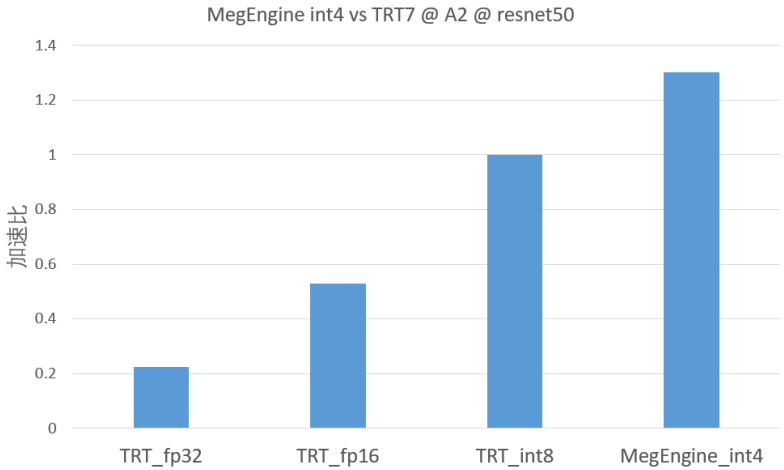


MEGVII 旷视

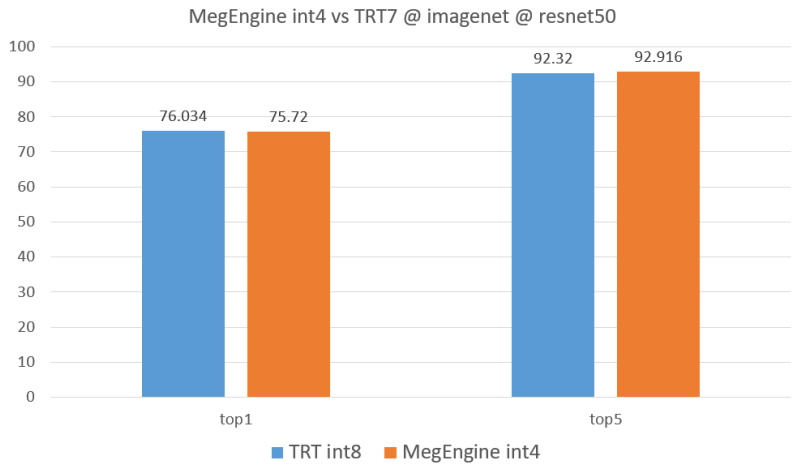
04

总结&展望

总结&展望



速度，TRT int8 性能被标准化为 1



精度

MegEngine 对低 bit 量化做了微小的贡献

1. 我们接触的模型有限，更多的业界模型在 int4 量化下的表现还有待发掘
2. 只做了 CUDA Int4 的实现，其他计算设备的 int4 适配还没有做
3. ……

团队招人中

如果你对高性能计算，深度学习框架，AI 编译器感兴趣…

欢迎加入 **MegEngine** 团队

wangbiao@megvii.com 投递简历

欢迎关注 MegEngine

- 项目源码：https://github.com/MegEngine/MegEngine
- 示例模型：https://github.com/MegEngine/examples
- 官　　网：https://www.megengine.org.cn/

Q & A