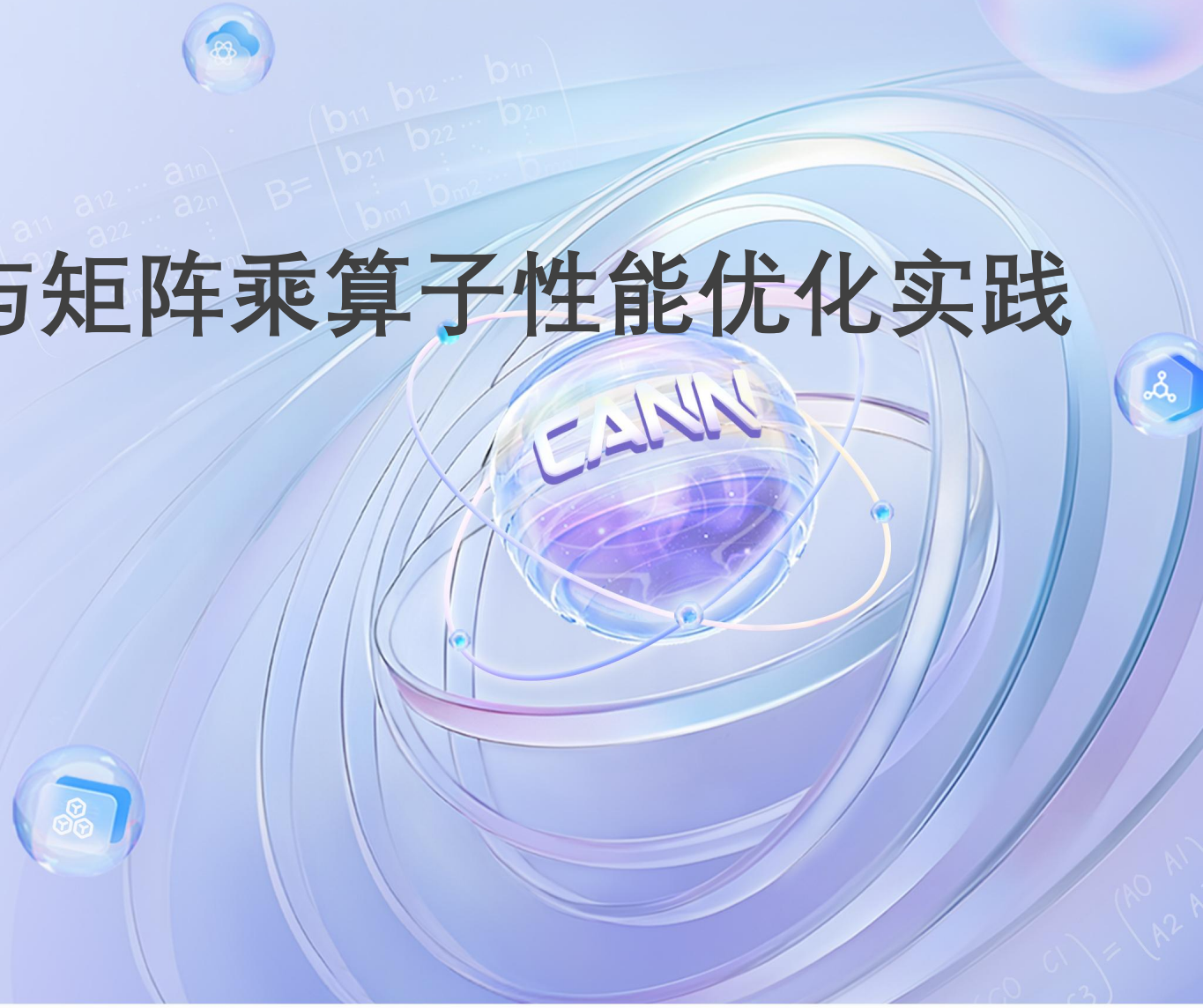
ops-nn仓开源介绍与矩阵乘算子性能优化实践



https://gitcode.com/cann

CANN

目录

ops-nn仓基础介绍

ops-nn是CANN （Compute Architecture for Neural Networks）算子库中提供神经网络计算能力的高阶算子库，包括matmul类、activation类等算子，算子库架构图如下

https://gitcode.com/cann

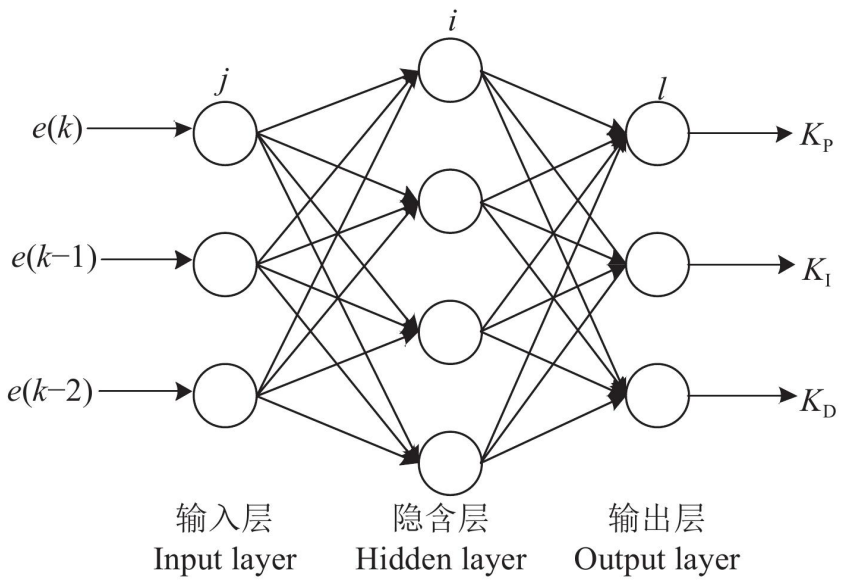
https://gitcode.com/cann/ops-nn

CANN

ops-nn仓典型算子功能介绍

算子类目	算子类目	核心功能介绍
foreach	循环操作	遍历输入数据（如张量、数组）的每个元素或维度，对单个元素 / 维度执行预设操作（如计算、转换），支持批量循环处理，避免手动迭代逻辑，提升代码简洁性与执行效率。
control	控制流	支持在神经网络中实现条件分支（if-else）、循环控制（while）、流程跳转与断言检测，并可依据中间计算结果动态调整计算路径，从而满足复杂模型（如动态层数网络）的流程控制需求。
index	索引操作	对张量进行索引访问、切片或重塑，包括按位置索引（如取特定行 / 列）、布尔索引（按条件筛选元素）、高级索引（如不规则切片），用于提取、重组或筛选张量中的目标数据。
hash	哈希操作	对输入数据（如特征值、张量元素）计算哈希值，支持哈希映射（如将特征映射到固定维度的哈希表）、哈希编码（如特征离散化），常用于高效数据检索、特征降维或哈希 - based 模型（如哈希嵌入）。
vfusion	融合操作	融合多个独立算子的计算逻辑，减少算子间数据传输开销（如内存读写），提升硬件计算利用率，尤其在端侧或高性能推理场景中优化运行速度。
rnn	循环神经网络	处理序列数据（如文本、时序信号）的核心算子，通过隐藏状态传递历史信息。
pooling	池化操作	对特征图进行下采样，降低维度并保留关键特征，常见类型包括最大池化（取局部区域最大值，突出显著特征）、平均池化（取局部区域平均值，保留整体趋势）、自适应池化（动态调整输出尺寸），用于减少计算量、防止过拟合。
activation	激活函数	为神经网络引入非线性变换，解决线性模型无法拟合复杂数据的问题，常见类型包括 ReLU（修正线性单元，缓解梯度消失）、Sigmoid（将输出映射到 0-1，用于二分类）、Tanh（将输出映射到 - 1-1，增强非线性表达）等。
loss	损失函数	衡量模型预测结果与真实标签的差异，是模型训练的优化目标。
optim	优化器	实现神经网络训练中的参数更新策略，通过计算梯度调整模型权重以最小化损失函数
norm	归一化	对张量数据进行标准化处理，减少内部协变量偏移，加速模型训练并提升稳定性，常见类型包括 BatchNorm（批归一化，按批次统计均值 / 方差）、LayerNorm（层归一化，按特征维度统计）、InstanceNorm（实例归一化，常用于生成模型）等。
quant	量化操作	将高精度数据（如 32 位浮点数）转换为低精度数据（如 8 位整数、16 位浮点数），减少模型存储占用与计算开销，同时尽量保留模型精度，支持训练后量化（PTQ）、量化感知训练（QAT），适用于端侧部署（如手机、嵌入式设备）。
matmul	矩阵乘法	实现两个张量的矩阵乘法运算，是神经网络中全连接层、注意力机制等模块的核心计算单元，支持高维张量广播（如 batch 维度的批量矩阵乘法）、转置乘法（如matmul(A, B, transpose_b=True)），适配不同硬件的并行计算优化
conv	卷积操作	卷积神经网络（CNN）的核心算子，通过卷积核（滤波器）与输入特征图进行滑动窗口计算，提取局部空间特征（如边缘、纹理、形状），支持 2D 卷积（图像处理）、3D 卷积（视频 /volumetric 数据处理）、等类型。

ops-nn仓典型算子功能介绍



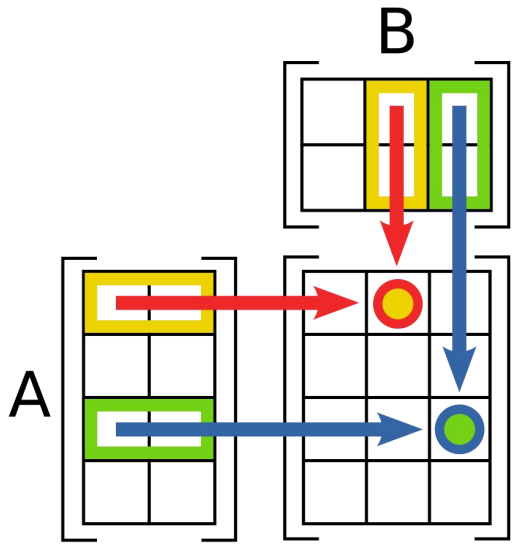

```python
import torch
import torch.nn as nn
import torch_npu  # 导入昇腾NPU支持库

# 定义网络结构
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size) # 底层matmul
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size) # 底层matmul
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

# 设置超参数
input_size = 10
hidden_size = 5
output_size = 1
learning_rate = 0.01
epochs = 100

device = torch.device("npu:0") # 使用第0块NPU

# 初始化模型并迁移到NPU
model = SimpleNN(input_size, hidden_size, output_size).to(device)

# 损失函数和优化器（使用NPU支持的优化器）
criterion = nn.MSELoss()  # 注意修正了原代码中的MseLoss小写问题
# 使用NPU版本的Adam优化器
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

目录

ops-nn开源算子能力深度解析——MatMul

矩阵乘法 Matrix multiplication

- 矩阵乘法是一种二元运算，它从两个矩阵生成一个矩阵。
- 对于矩阵乘法来说，第一个矩阵的列数必须等于第二个矩阵的行数。
- 生成的矩阵，称为矩阵乘积，具有第一个矩阵的行数和第二个矩阵的列数。



网络类型	矩阵乘法占比（计算量）	主要来源
全连接网络	90%-100%	全连接层的W·x操作
CNN（如 ResNet）	70%-90%	卷积层（im2col+matmul）
Transformer	85%-95%	注意力机制（QK^T, AV）和 FFN
RNN/LSTM	60%-80%	门控机制中的矩阵乘法

$$设A=\begin{bmatrix}1 & -2\\ -1 & 0\end{bmatrix},\ B=\begin{bmatrix}2 & 0\\ 3 & 1\end{bmatrix},\ 计算AB:$$

$$AB=\begin{pmatrix}1 & -2\\ -1 & 0\end{pmatrix}\cdot\begin{pmatrix}2 & 0\\ 3 & 1\end{pmatrix}$$

$$=\begin{pmatrix}1\times 2+(-2)\times 3 & 1\times 0+(-2)\times 1\\ (-1)\times 2+0\times 3 & (-1)\times 0+0\times 1\end{pmatrix}$$

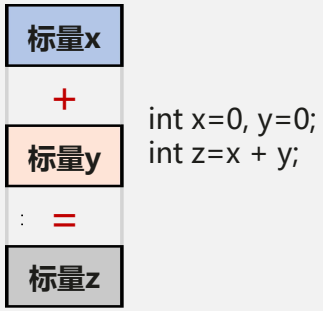
$$=\begin{pmatrix}-4 & -2\\ -2 & 0\end{pmatrix}$$

CANN

采用NPU进行计算加速的小知识

标量计算单元

向量计算

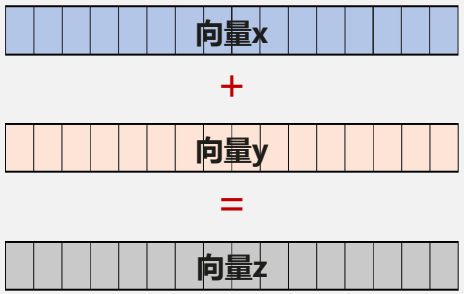



```
int x=0, y=0;
int z=x + y;
```

Vector计算单元

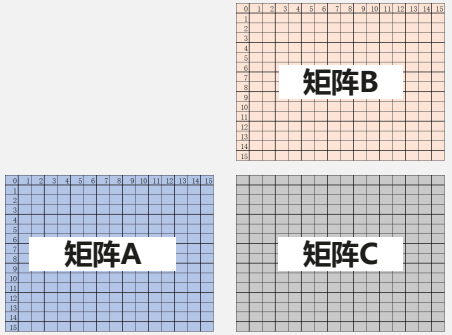
向量计算

单核1个cycle可以处理128个fp16数据



Cube计算单元

矩阵计算



- 针对AI计算特点，包含**大Cube计算单元，矩阵乘**计算效率高，达到**极致面效比**
- **16*16*16**大Cube设计，**AI大模型场景计算效率更高**

Cube计算单元负责执行矩阵运算，Fp16算力理论值计算公式：

$$16 * 16 * 16 * 频率 * AI核数量 * 2（乘加）$$

$$Tcal = \frac{number\ of\ Flops}{BWcal}$$

即一个核可以在1个时钟周期处理完成对FP16的16*16*16的矩阵乘加运算

CANN

MatMul算子能力深度解析

算子的硬件江湖：十八般武艺

Arithmetic Intensity = FLOPs / Bytes Moved

$$Tcal = \frac{number\ of\ Flops}{BWcal}$$

$$Tmem = \frac{number\ of\ bytes}{BWmem}$$

当 $Tcal \geq Tmem$

软件实现才可能是计算bound

与算力共舞：

- 优化分核逻辑，充分发挥算力

搬运数据少一点：

- 合理分块: 获取最优计算访存比
- 矩阵全载: 小shape矩阵之道
- 外积&混合内外积：提升访存效率

搬运带宽大一点：

- 聚合与调度：提高L2访存
- 大包搬运：提升带宽利用率
- weight预取：让数据先飞一会儿

搬运计算合理调度：

- 双缓冲流水并行，让计算连续

融合算子，硬件之上的算法跃动：

- 灵活使用指令跳跃，算子无损消除
- CV融合

CANN

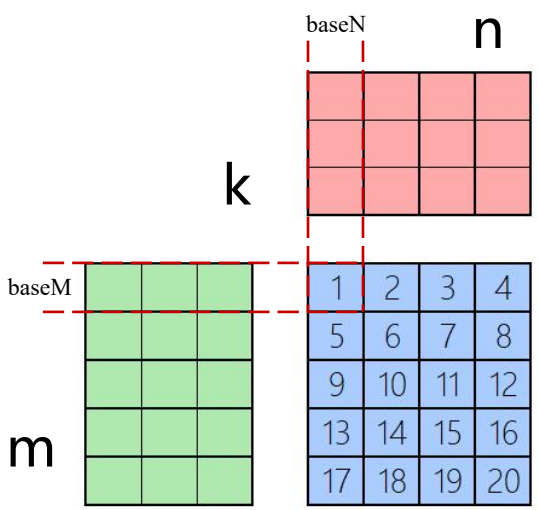
与算力共舞

昇腾Cube计算单元负责执行矩阵运算，Fp16算力理论值计算方式：

16 * 16* 16 * 频率 * AI核数量 * 2（乘加）

即一个核可以在1个时钟周期处理完成对FP16的16*16*16的矩阵乘加运算

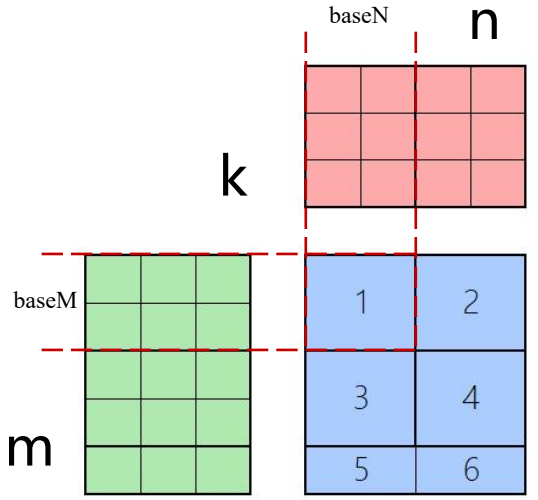
启动更多的核同时计算，可以提高计算并行度，充分发挥算力空间



20核负载均衡

A · B = C

VS



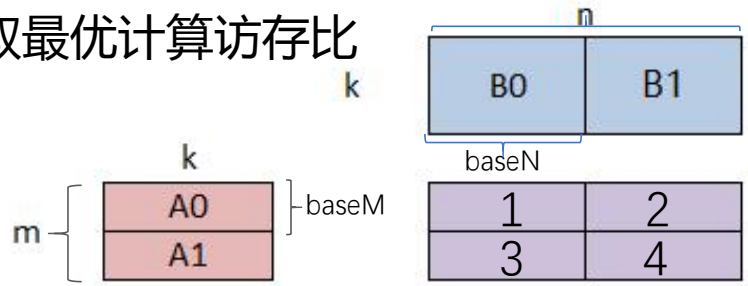
6核负载不均衡

$$Tcal = \frac{2 \times m \times k \times n}{2 \times 16 \times 16 \times 16 \times 核数 \times 频率}$$

CANN

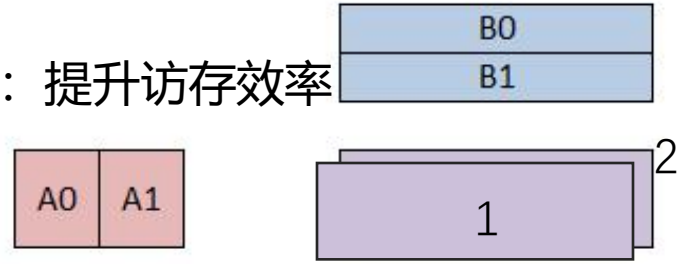
搬运数据少一点

- 合理分块: 获取最优计算访存比



- 矩阵全载: 小shape矩阵之道

- 外积&混合内外积：提升访存效率



$$Tmem = \frac{number\ of\ bytes}{BWmem}$$

$$= \frac{\left(m \times \frac{n}{baseN} + n \times \frac{m}{baseM}\right) \times k * 2Byte}{BWmem}$$

$baseM \times baseN \times 4Byte \leq$ 核输出buffer空间

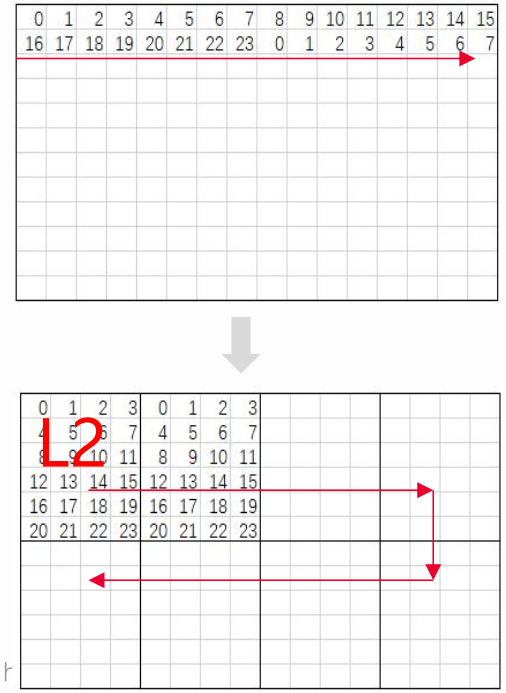
最优baseM=256，baseN=128；*(910B为例)*

搬运带宽大一点

$$Tmem = \frac{number\ of\ bytes}{BWmem}$$

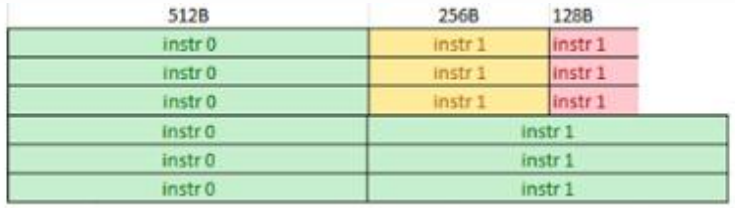
聚合与调度：提高L2访存

- shape超L2 cache的场景，传统的数据搬运方式，导致victim严重，影响访存效率
- 增加L2层切分，让多核在同一时间访问相同内存区域，提升L2命中率
- 控制L2区块访问顺序，复用L2数据



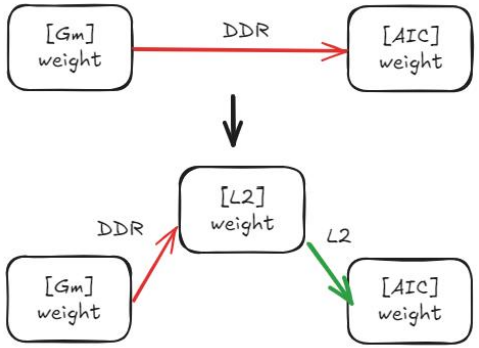
大包搬运：提升带宽利用率

- 内轴小于cache line，受小包搬运影响，导致访存效率低
- 推理场景下将weight预处理成硬件亲和格式，保证数据搬运搬运满足cache line，提高访存效率



weight预取：让数据先飞一会儿

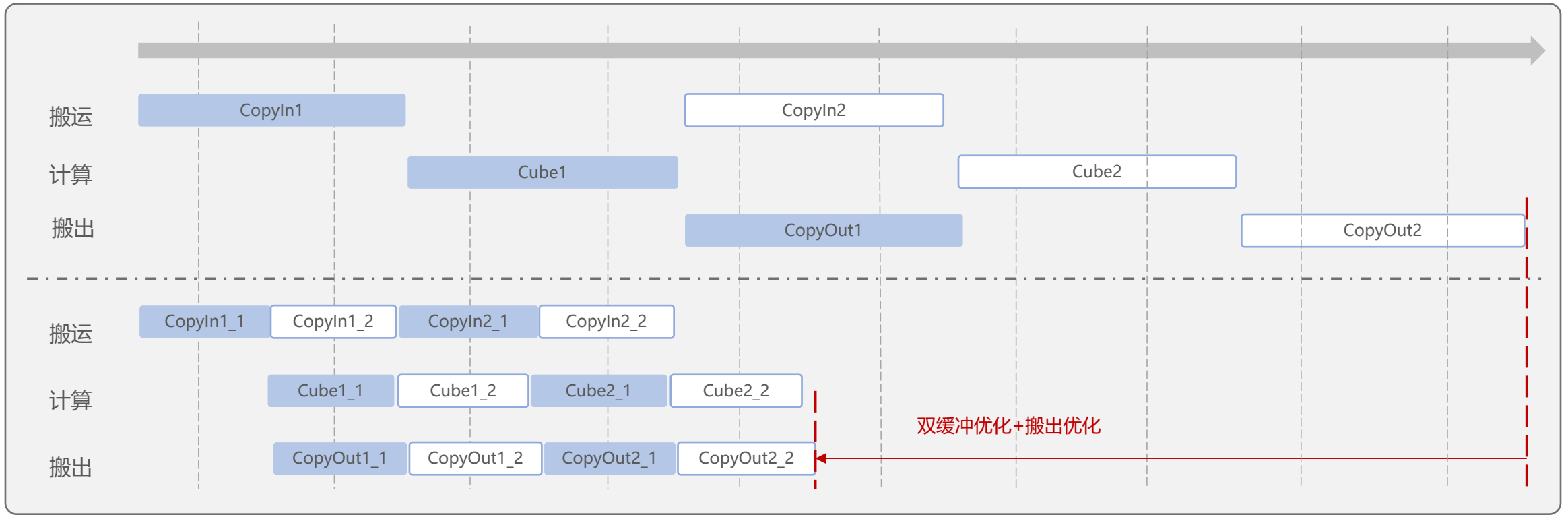
- 在weight矩阵是常量，可以采用weight预取方式（提前读取到L2中），减少数据访存耗时



CANN

搬运计算合理调度

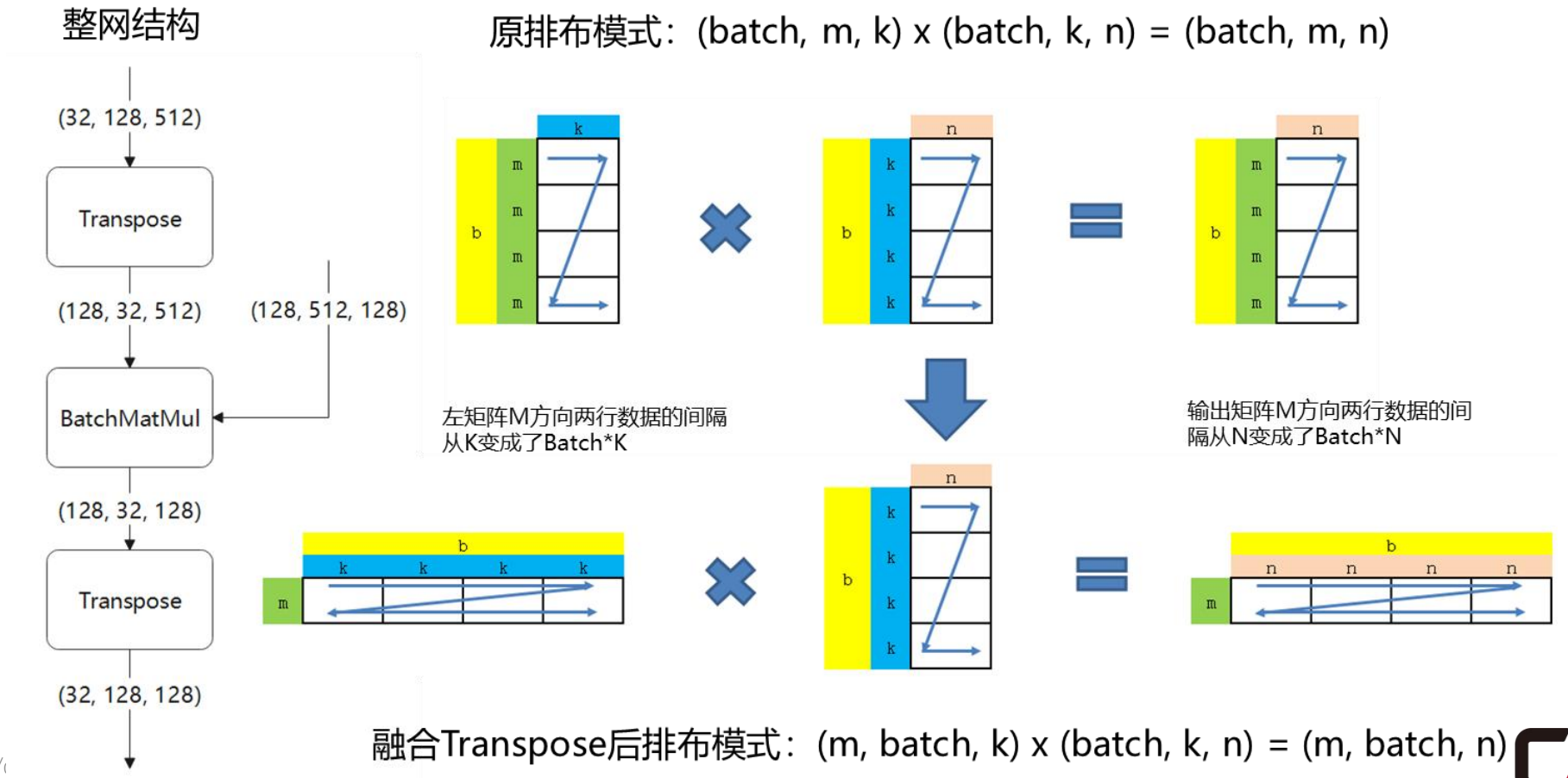
充分利用**并行执行单元**，提高硬件利用率，降低处理延迟
设计优化**数据并行处理**，提高数据吞吐量，保证计算连续



CANN

融合算子，硬件之上的算法跃动

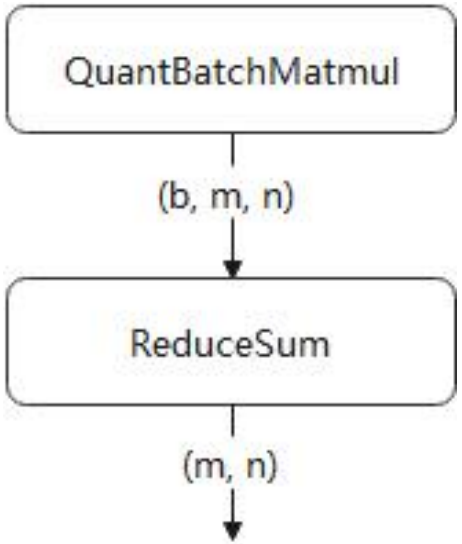
- Deepseek网络中存在Transpose+BatchMatMul算子组合，前后两个Tranpose算子增加数据搬运耗时，影响网络竞争力。
- 通过调整搬运指令的stride，从原来的K适配成Batch*K，N适配成Batch*N，修改输入输出矩阵取址的offset，完成转置排布数据处理；
- 该方法只修改搬运的排布，没有变更算子搬运的逻辑，是一种无损的优化，能够去掉前后两个Transpose算子耗时。

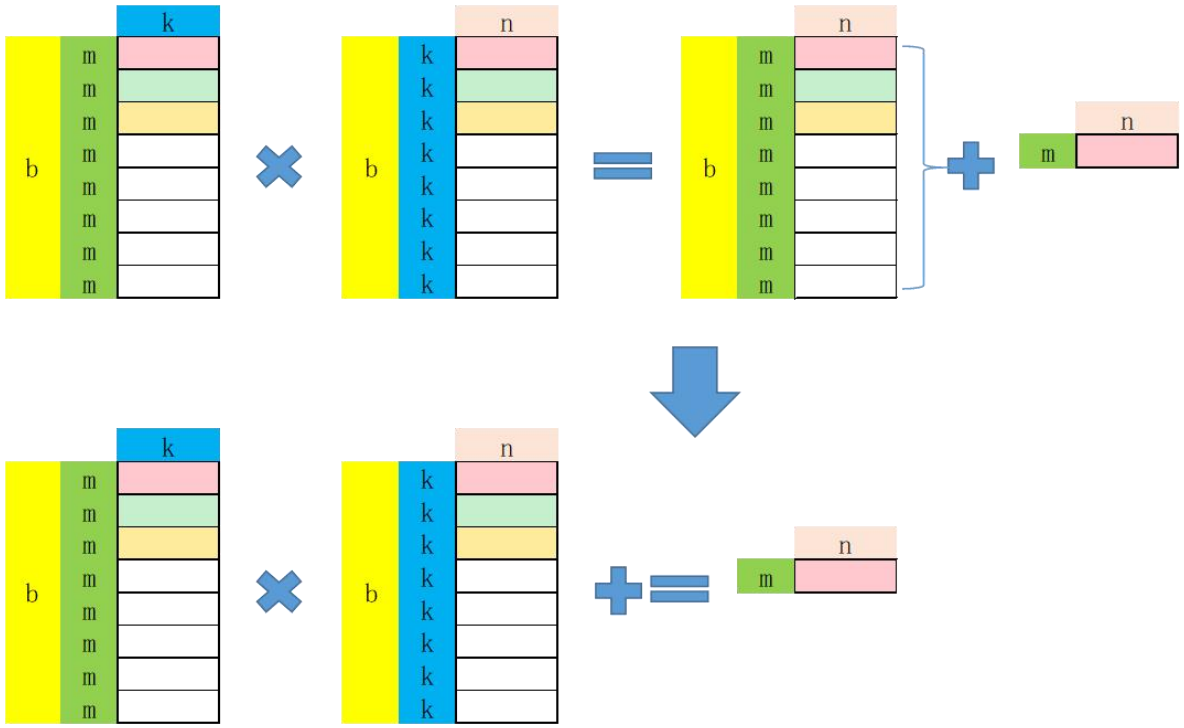


CANN

融合算子，硬件之上的算法跃动

- DeepSeek网络中存在QuantBatchMatmul+ReduceSum组合。需要在MM计算结束后，对其batch轴进行ReduceSum，这样会增加一个ReduceSum算子的耗时，影响网络性能竞争力。
- 融合QuantBatchMatmul和ReduceSum算子：在QuantBatchMatmul输出结果时，直接对batch轴进行atomic累加完成ReduceSum操作。(b,m,k) x (b,k,n)->(m,n)；





CANN

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann