AI编译器-系列之前端优化

算子融合

ZOMI

Talk Overview of Frontend Optimizer

I. AI 编译器前端优化

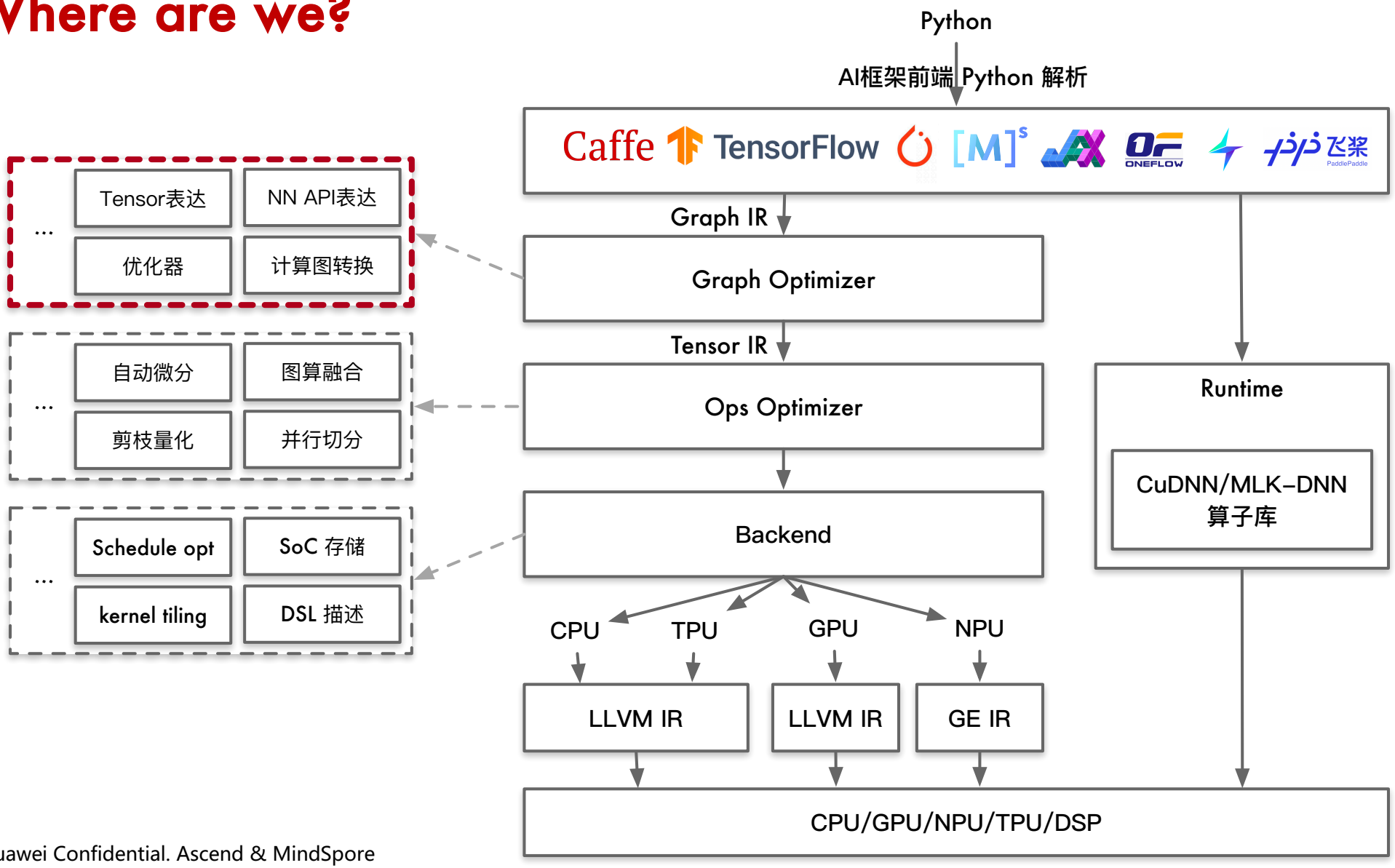
- 图层 - Graph IR
- 算子融合 - OP Fusion
- 布局转换 - Layout Transform
- 内存分配 - Memory Allocation
- 常量折叠 – Constant Fold
- 公共子表达式消除 - CSE
- 死代码消除 - DCE
- 代数简化 - ARM

Talk Overview

Operator Fusion - 算子融合

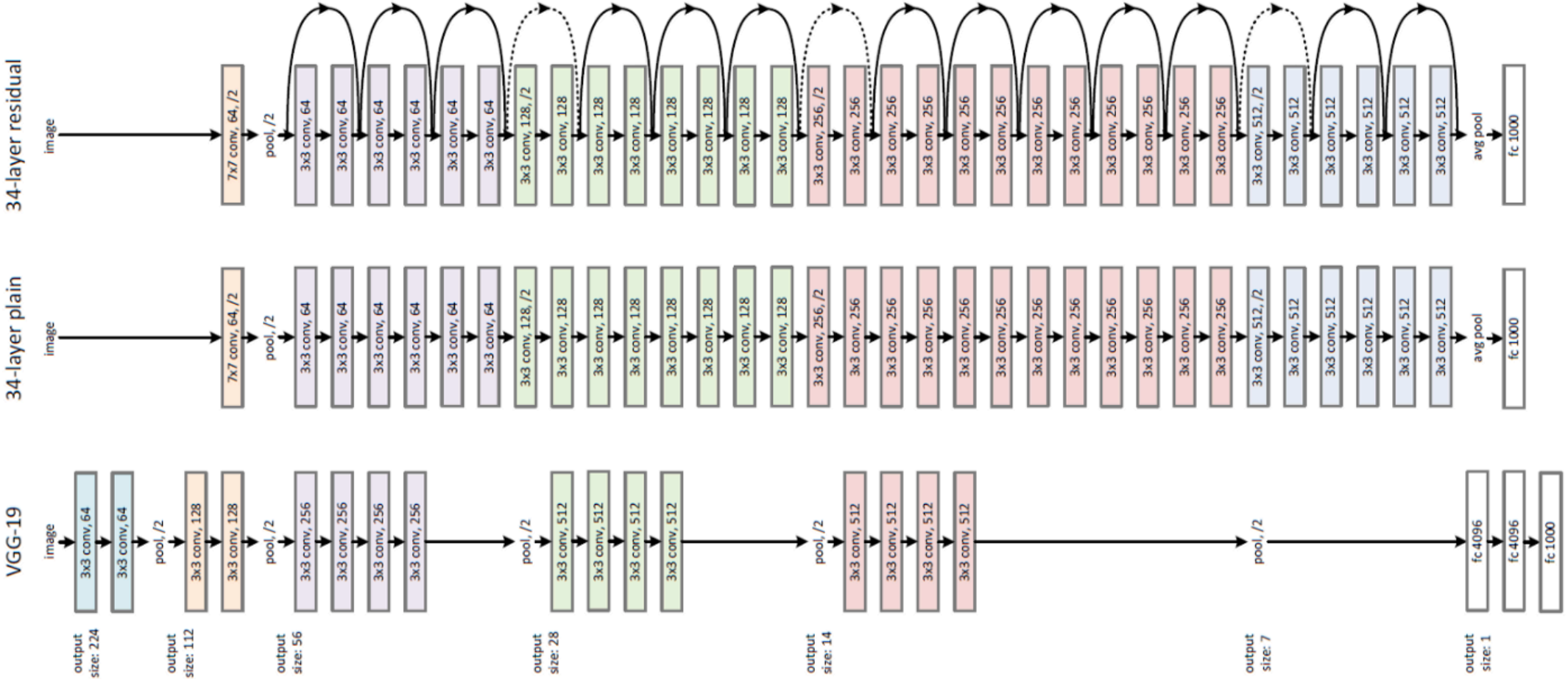
- 算子融合方式
- 算子融合栗子
- 融合的规则和算法

Where are we?

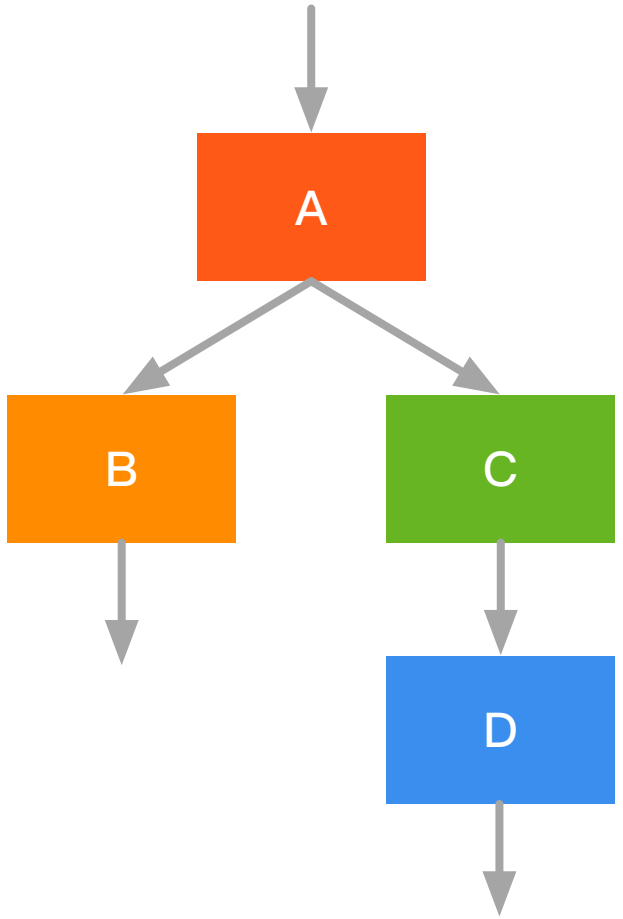


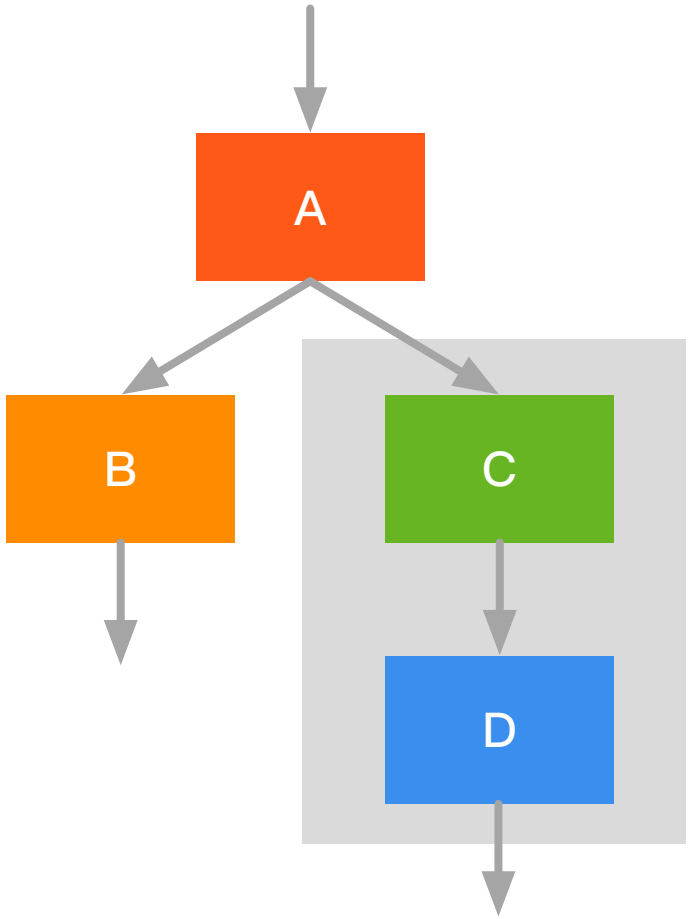
算子融合方式

网络模型



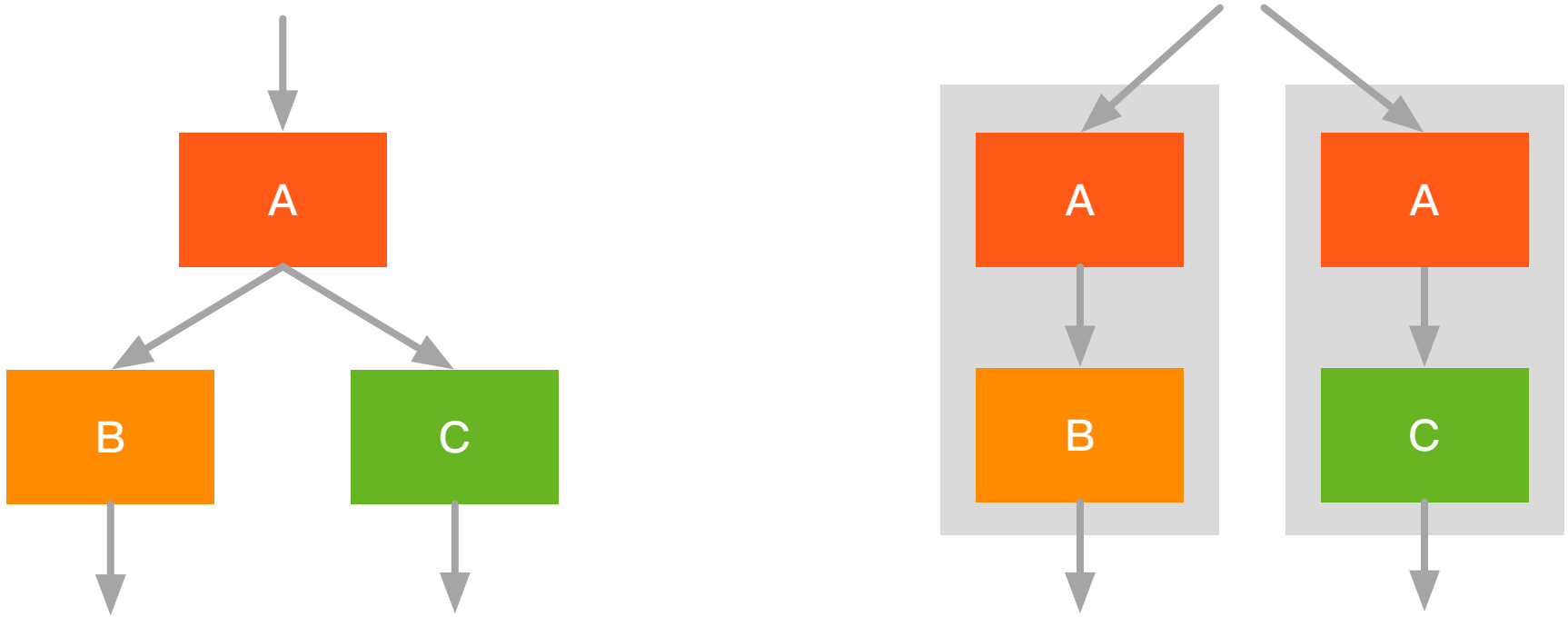
融合方式





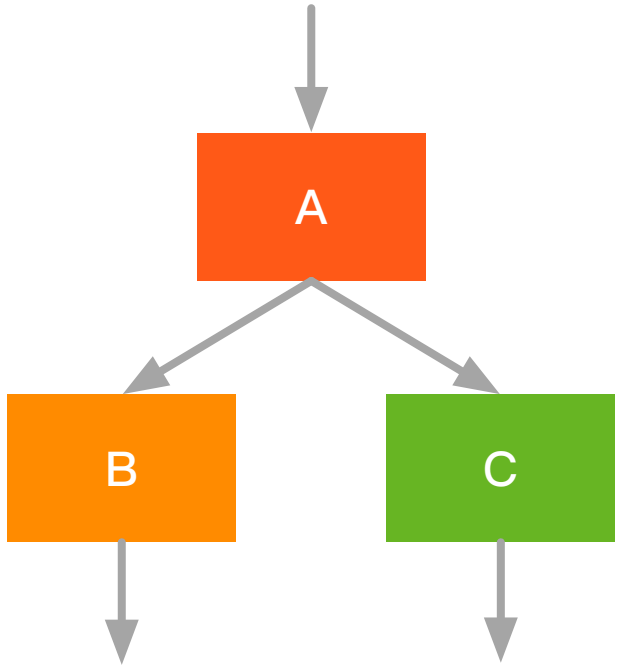
融合方式

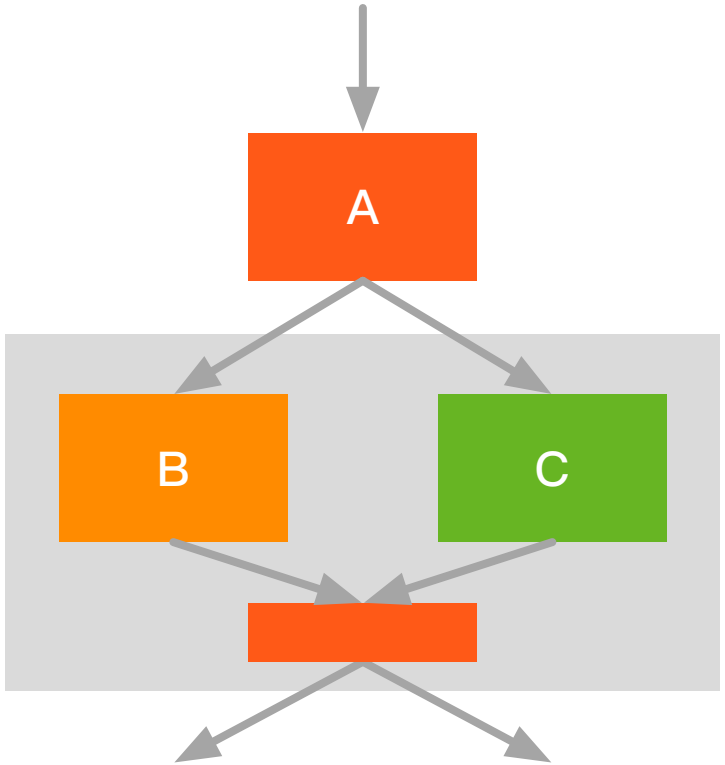
- 左侧：B/C并发执行，A一个来回，B/C一个来回
- 右侧：A+B/A+C并发一个来回



融合方式

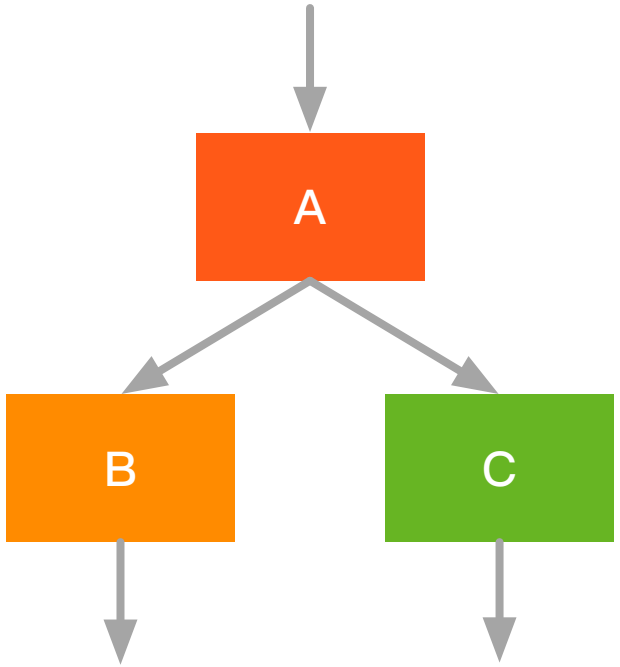
- 左侧：B/C并发，要两次kernel调用开销
- 右侧：B/C融合，减少一次kernel调用，两者计算都依赖A的结果放入内存，缓存效率会更高

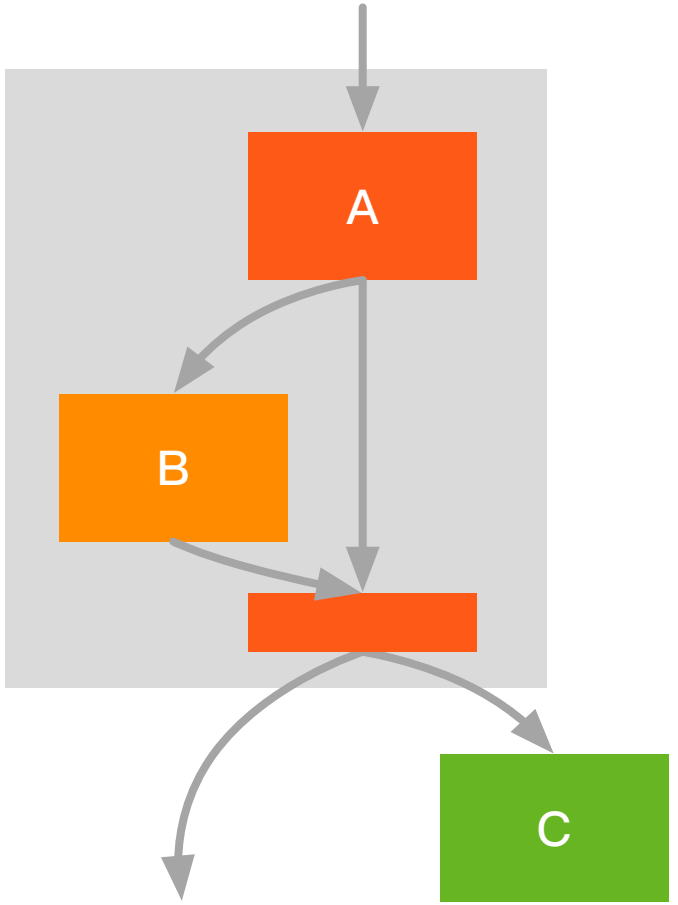




融合方式

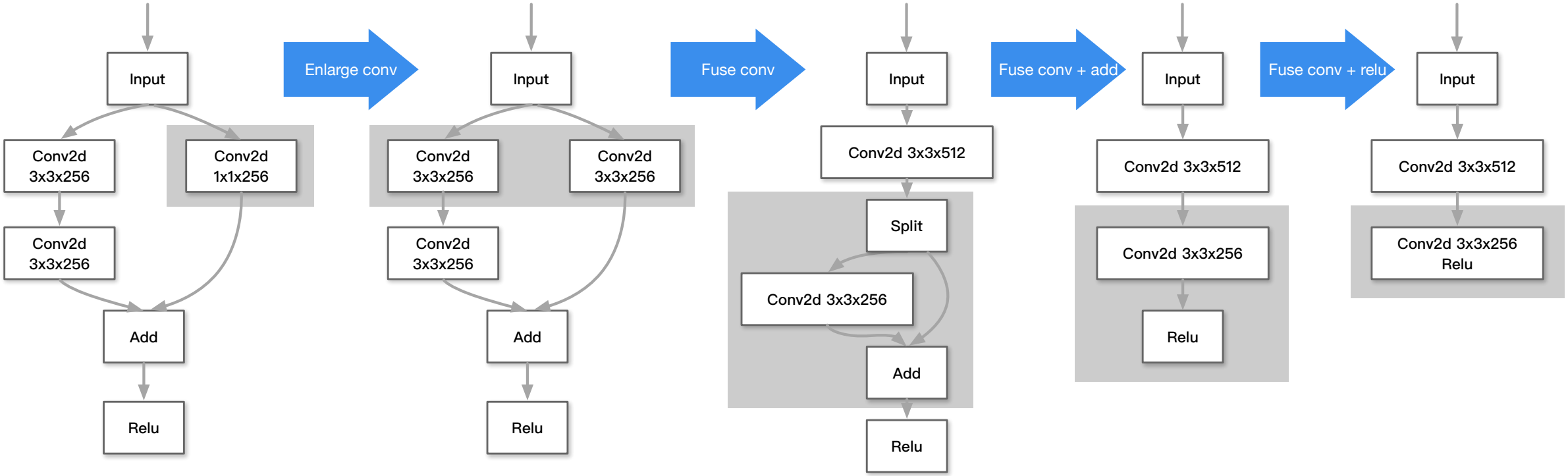
- 左侧：三次kernel开销
- 右侧：同时A/B间合并，提升内存使用效率





融合方式 Review

- 融合算子出现主要解决模型训练过程中的读入数据量，同时，减少中间结果的写回操作，降低访存操作。



Pros and Cons

- 消除不必要的中间结果实例化
- 减少不必要的输入扫描
- 发现其他优化机会

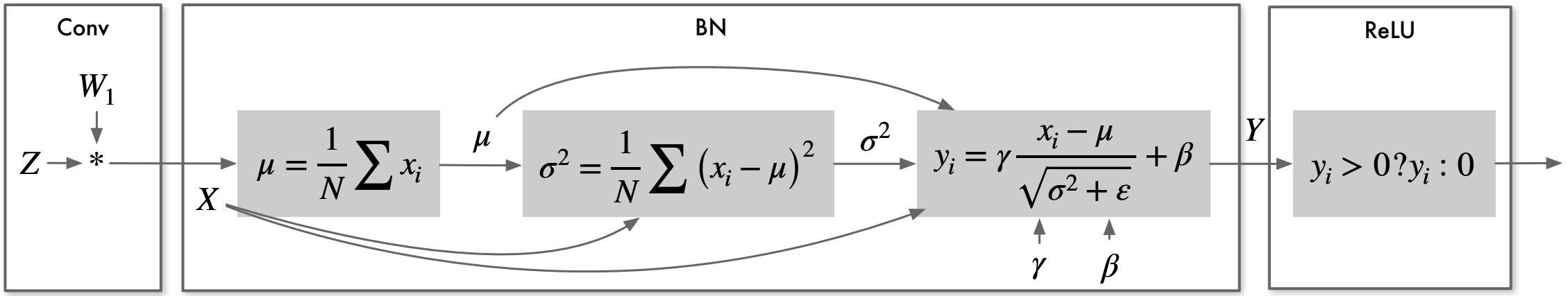
Pros and Cons

- **内存墙**：主要是访存瓶颈引起。算子融合主要通过对计算图上存在数据依赖的“生产者-消费者”算子进行融合，从而提升中间Tensor数据的访存局部性，以此来解决内存墙问题。这种融合技术也统称为“Buffer融合”。在很长一段时间，Buffer融合一直是算子融合的主流技术。早期的AI框架，主要通过手工方式实现固定Pattern的Buffer融合。
- **并行墙**：主要是由于芯片多核增加与单算子多核并行度不匹配引起。可以将计算图中的算子节点进行并行编排，从而提升整体计算并行度。特别是对于网络中存在可并行的分支节点，这种方式可以获得较好的并行加速效果。

Conv-BN-ReLU
算子融合

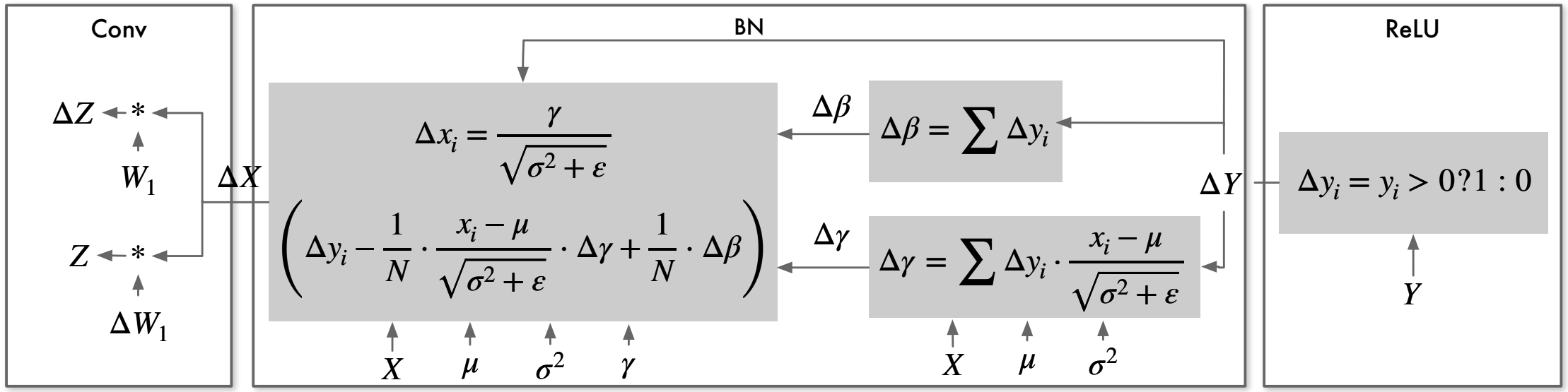
BN 计算流程

- 在 BN 前向计算过程中，首先求输入数据的均值与方差，然后使用均值、方差对每个输入数据进行归一化及缩放操作。其中均值及方差依赖于输入数据；归一化及缩放计算的输入则依赖于输入数据、均值、方差以及两个超参数。下图为前向计算过程中 BN 的数据依赖关系：



BN计算流程

- BN反向计算过程中，首先求参数误差；然后使用参数误差$\Delta\gamma$、$\Delta\beta$计算输入误差ΔX。参数误差导数依赖于输出结果误差ΔY以及输入X；输入误差ΔX依赖于参数误差导数及输入X、输出误差ΔY。反向过程包括求参数误差以及输入误差两部分，BN反向计算的关键访存特征是两次使用输入特征X及输出误差ΔY，分别用于计算参数误差$\Delta\gamma$、$\Delta\beta$及输入数据误差ΔX。

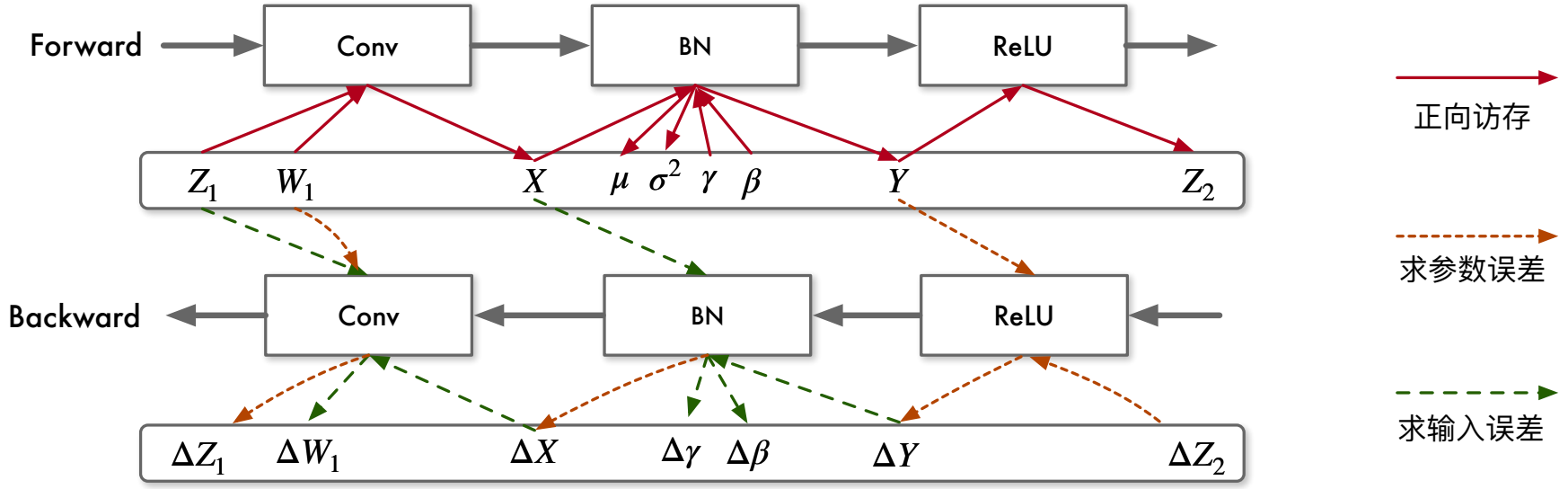


计算访存分析

- 网络模型训练时，需要保存每层前向计算时输出结果，用于反向计算过程参数误差、输入误差的计算。但是随着深度学习模型的加深，需要保存的中间参数逐渐增加，需要消耗较大的内存资源。由于加速器片上缓存容量十分有限，无法保存大量数据。因此需将中间结果及参数写出到加速器的主存中，并在反向计算时依次从主存读入使用。

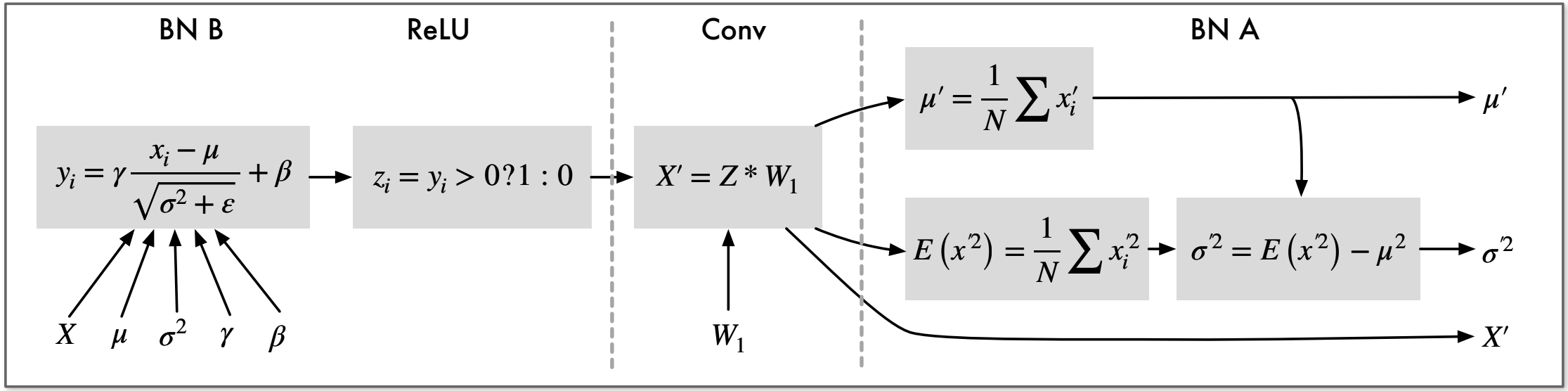
计算访存分析

- 前向计算过程中，每层的计算结果需写出主存，用于反向计算过程中计算输入误差；反向计算过程中，每层的结果误差也需写出到主存，原因是反向计算时BN层及卷积层都需要进行两次计算，分别求参数误差及输入数据误差，图X、 ΔY加载两次来计算参数误差$\Delta\gamma$、$\Delta\beta$及输入误差ΔX。ReLU输入Y不需要保存，直接依据结果 Z即可计算出其输入数据误差。



模型重构及融合算子

- 前向过程中，BN重构为两个子层：BN_A和BN_B。其中BN_A计算均值与方差，BN_B完成归一化与缩放，分别融合于相邻卷积层及激活层。首先从主存读取输入*X*、均值μ、方差 δ2 、参数γ、 β，计算BN_B，完成归一化及缩放计算，将结果*Y*用于激活计算，输出*Z 用于*卷积计算，卷积结果*X'*写出到主存之前，计算BN_A，即求均值 μ'与方差 δ2 '。完成“归一化缩放->激活层->卷积层->计算卷积结果均值与方差”结构模块的前向计算过程只需要读取一次，并写回卷积计算结果*X'*及相关参数。



Conv+BN+ReLU

模型重构及融合算子

- 卷积计算：

$$z = w * x + b$$

- BN 计算：

$$y = (z - mean)/\sqrt{var} * \beta + \gamma$$

模型重构及融合算子

- 融合简化卷积和BN之间的计算：

$$w' = w/\sqrt{var} * \beta$$

$$b' = (b - mean)/\sqrt{var} * \beta + \gamma$$

$$\hat{f}_{i,j} = W_{BN} * \left(W_{conv} * f_{i,j} + b_{conv} \right) + b_{BN}$$

融合规则与算法

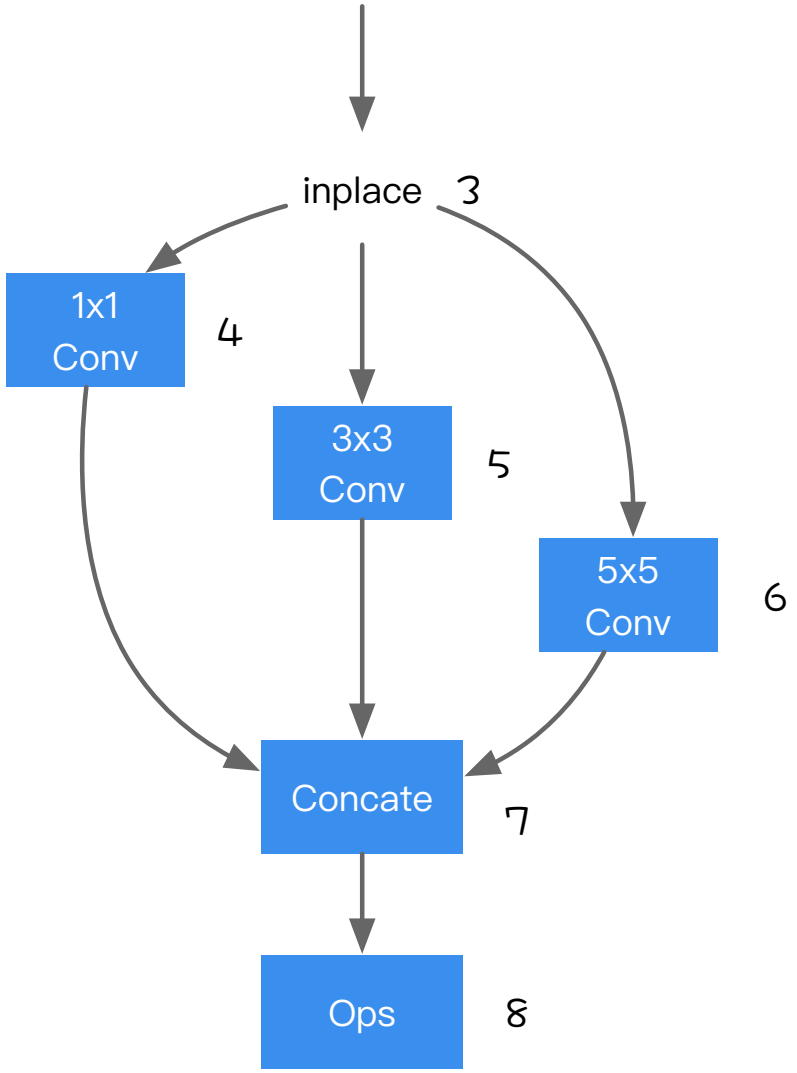
TVM 支配树

- TVM整体的算子融合是基于支配树实现的。那么什么是支配树呢？
- 支配树就是由各个点的支配点构成的树，那么什么又是支配点呢？

- 支配点：所有能够到达当前节点的路径的公共祖先点（ Least Common Ancestors，LCA ）

TVM 支配树

- 支配树作用：
  - 检查每个Node到其支配点的Node是否符合融合条件
  - 融合的基本规则是融合掉的Node节点不会对剩下的节点产生影响
- 支配树生成：
  - 根据DAG生成DFS树
  - 根据DFS树及对应的边生成DOM树
  - 使用Group来描述多个Node是否能被融合



TVM 算子融合流程

1. 通过AST转换为Relay IR，遍历Relay IR；
2. 建立DAG用于后支配树分析；
3. 应用算子融合算法；

TVM 算子融合流程

1. 通过AST转换为Relay IR，遍历Relay IR；
2. 建立DAG用于后支配树分析；
3. 应用算子融合算法

1. 首先根据DAG进行深度优先遍历，生成DFS树，需要注意的是，DFS树是倒序，也就是最后一个节点是节点0，然后依次深度递增；除了单纯的记录每个Node的深度之外，我们还需要为每个节点保存与他相连的边，注意这个边是与的父节点（也就是真正网络中他的输出节点，倒序就变成了他的父节点了）组成的。之所以这里需要保存这个边和其对应的index，就是为了后面找LCA用。

TVM 算子融合流程

1. 通过AST转换为Relay IR，遍历Relay IR；
2. 建立DAG用于后支配树分析；
3. 应用算子融合算法

2. 根据DFS树及对应的边（link）生成DOM树，TVM是使用group这个概念来描述几个Node能否融合，如果一个算子不能和任何其他算子融合，那么这个group就只有他自己，同样如果几个算子能够融合，产生一个新group。

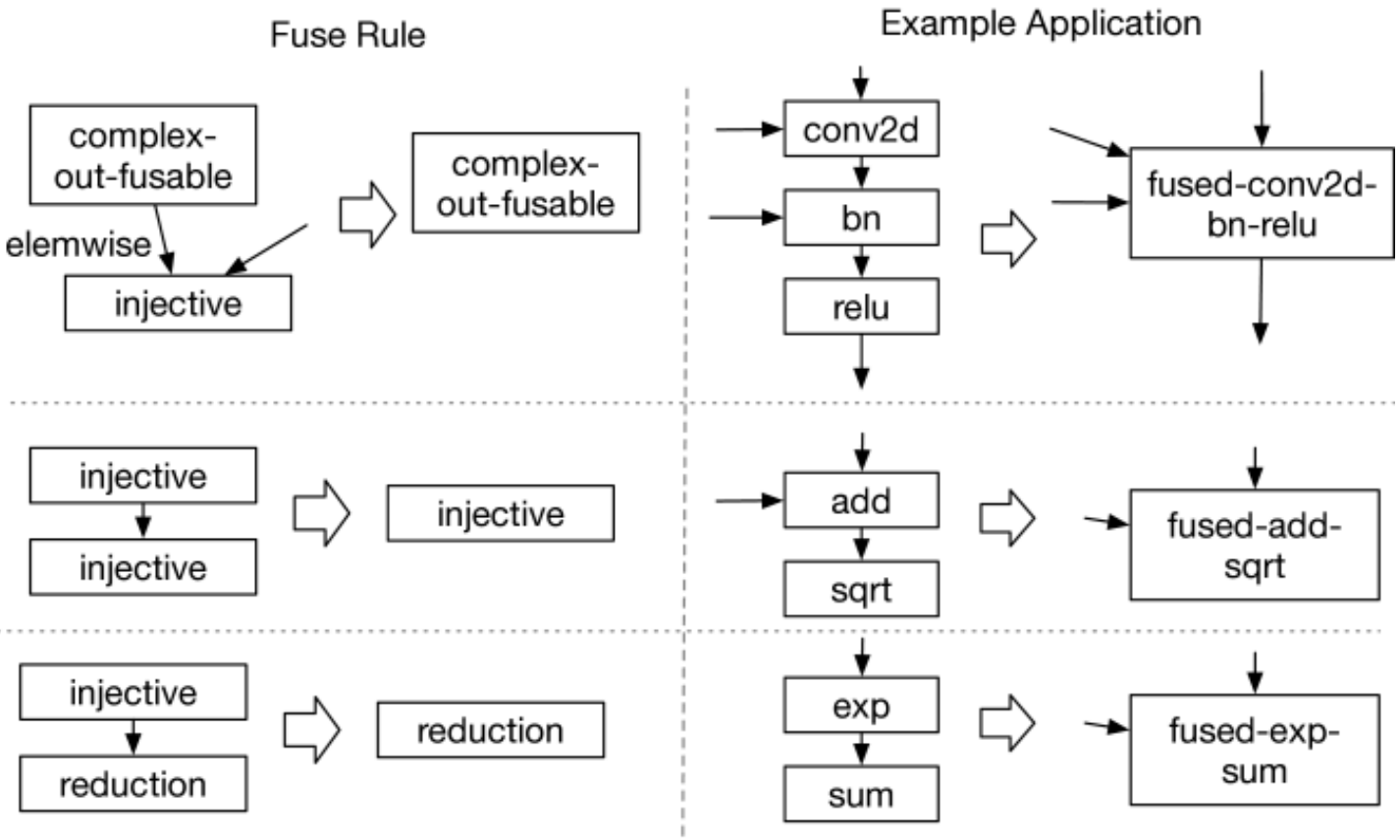
TVM 算子融合流程

1. 通过AST转换为Relay IR，遍历Relay IR；
2. 建立DAG用于后支配树分析；
3. 应用算子融合算法

3. 遍历每个Node到它的支配带你的所有路径是否符合融合规则，完成融合后，遍历节点创新的DAG图

Rules for operator fusion

- Injective(one-to-one map)：映射函数，如Add，Pointwise
- Reduction：约简函数，输入到输出具有降维性质，如sum/max/min
- Complex-out-fusable：an fuse element-wise map to output，计算复杂类型，如conv2d
- Opaque(cannot be fused) ：无法被融合的算子，比如sort



Experience Results

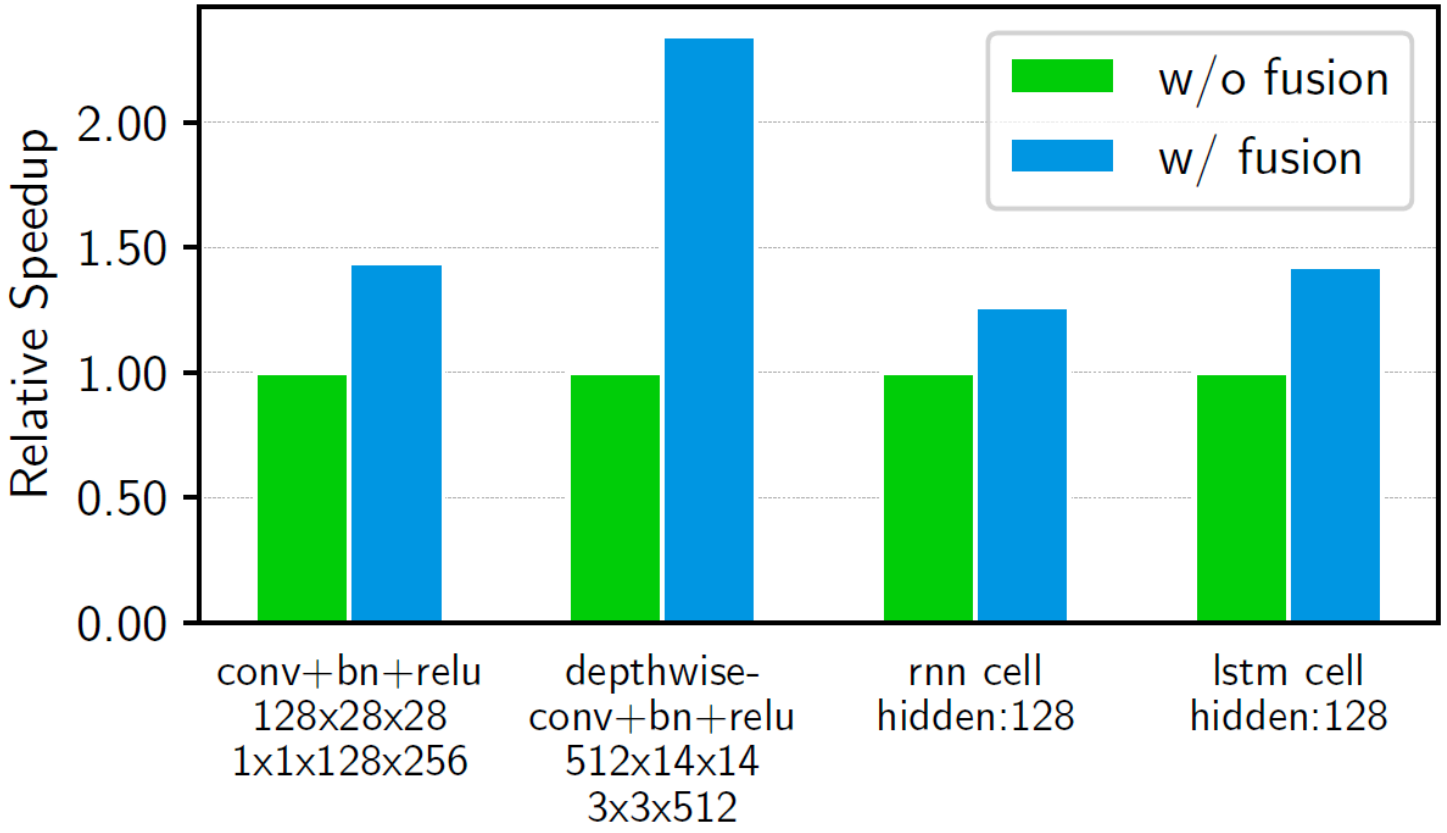
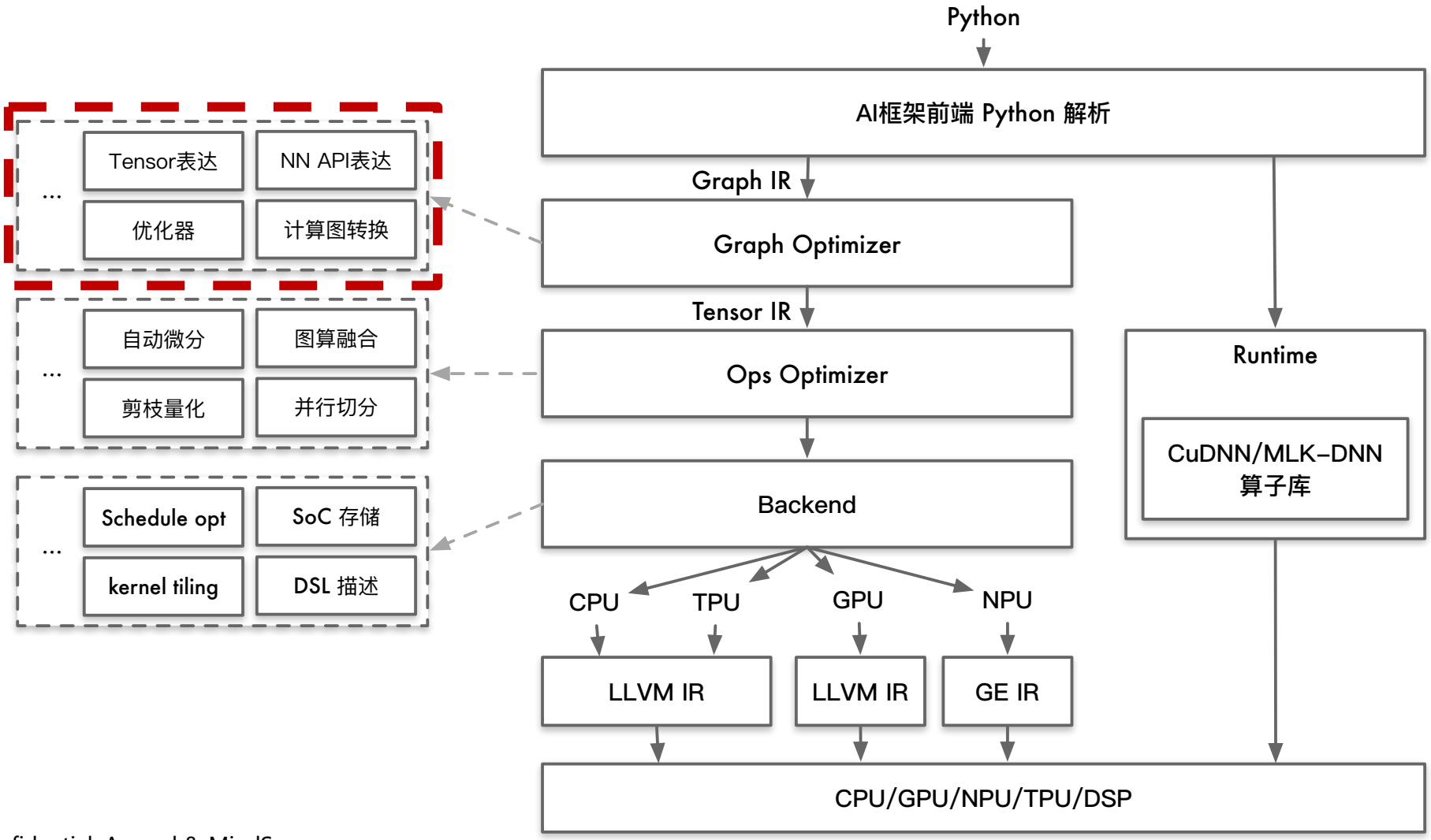


Figure 4: Performance comparison between fused and non-fused operations. TVM generates both operations. Tested on NVIDIA Titan X.

Where are we ?



Summary

- 算子的融合方式有横向融合和纵向融合，但根据AI模型结构和算子的排列，可以衍生出更多不同的融合方式；
- 通过 Conv-BN-ReLU 算子融合栗子，了解到如何对算子进行融合和融合后的计算，可以减少对于对访存的压力；
- 在编译器中，一般融合规则都是通过Pass来承载，不同的Pass处理不同的融合规则，而融合规则主要是人工定义好；

Reference

1. DNNFusion: Accelerating Deep Neural Networks Execution with Advanced Operator Fusion https://arxiv.org/abs/2108.13342
2. Using fused operators to improve performance. https://learn.microsoft.com/en-us/windows/ai/directml/dml-fused-activations
3. TensorFlow operation fusion. https://www.tensorflow.org/lite/models/convert/operation_fusion
4. 计算图替代——一种DNN框架计算图优化方法. https://zhuanlan.zhihu.com/p/393365764
5. 融合算子整理. https://zhuanlan.zhihu.com/p/374851010
6. Fusing batch normalization and convolution in runtime. https://nenadmarkus.com/p/fusing-batchnorm-and-conv/
7. Jeff Dean – TensorFlow w/XLA: TensorFlow, Compiled! Expressiveness with performance. https://www.youtube.com/watch?v=CJdR-0zGTgs
8. Op Fusion（一）：什么是算子融合. https://zhuanlan.zhihu.com/p/581755093