The Deep Learning Compiler- A Comprehensive Survey 深度学习编译器综述 （三）

The Deep Learning Compiler- A Comprehensive Survey 深度学习编译器综述 （一）

The Deep Learning Compiler- A Comprehensive Survey 深度学习编译器综述 （二）

4.3 Frontend Optimizations

在构建计算图后，前端会应用图级别的优化。许多优化在图级别上更容易识别和执行，因为图提供了计算的全局视图。**这些优化仅应用于计算图**，而不是后端的实现。因此，**它们与硬件无关**，并可以应用于各种后端目标。

前端优化通常被称为passes，通过遍历计算图的节点并执行图变换。前端提供了以下方法：1）从计算图中捕获特定特征；2）对图进行优化重写。除了预定义的passes外，开发人员还可以在前端中定义自定义的passes。大多数深度学习编译器可以在导入和转换DL模型为计算图后，确定每个操作的输入张量和输出张量的形状。这个特性允许深度学习编译器根据形状信息进行优化。图3展示了使用TensorFlow XLA进行计算图优化的示例。

在本节中，我们将前端优化分为三个类别：**1）节点级优化，2）块级（peephole, local）优化，以及3）数据流级（global）优化。**

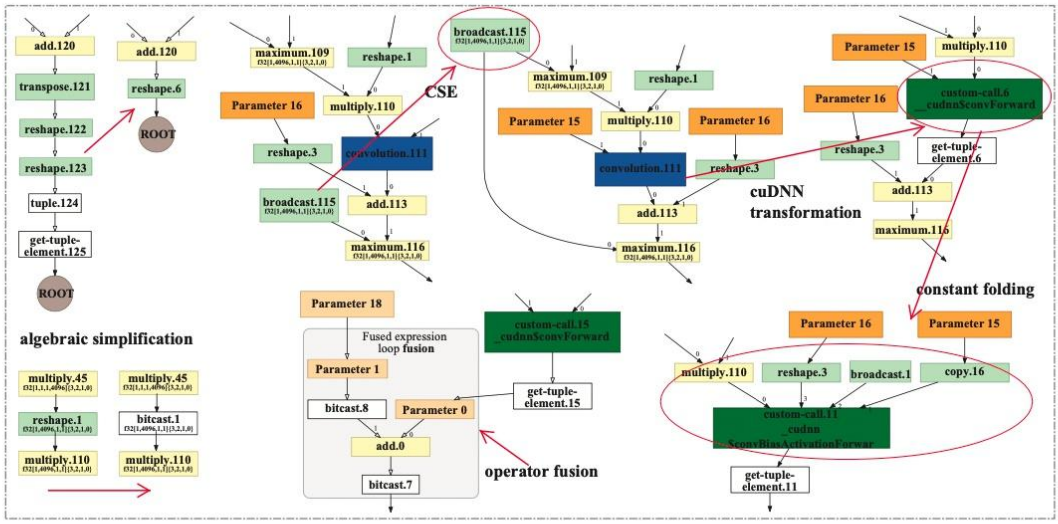
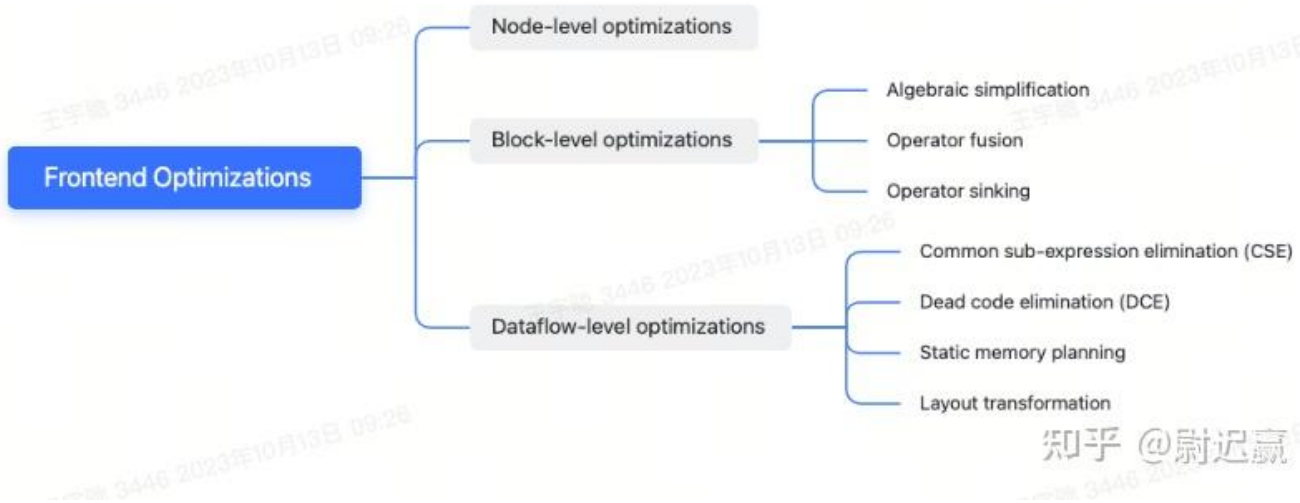


Fig. 3. Example of computation graph optimizations, taken from the HLO graph of AlexNet on NVIDIA GPU using Tensorflow XLA.



4.3.1 Node-level optimizations

计算图的节点足够粗粒度，可以在单个节点内进行优化。节点级优化包括节点消除，即消除不必要的节点，以及节点替换，即用其他成本更低的节点替换节点。

在通用编译器中，Nop消除（Nop Elimination）会移除占用很小空间但没有指定任何操作的无操作（no-op）指令。在深度学习编译器中，Nop消除负责消除缺乏足够输入的操作。例如，只有一个输入张量的sum节点可以被消除，零填充宽度的padding节点可以被消除。

零维张量消除（Zero-dim-tensor elimination）负责删除输入为零维张量的不必要操作。假设A是一个零维张量，B是一个常量张量，那么A和B的sum操作节点可以被替换为已有的常量节点B，而不会影响正确性。假设C是一个3维张量，但其中一个维度的形状为零，例如{0,2,3}，因此C没有任何元素，argmin/argmax操作节点可以被消除。

4.3.2 Block-level optimizations

Algebraic simplification

代数简化优化包括以下几种操作：1）代数识别（algebraic identification），2）强度降低（strength reduction），以便我们可以将昂贵的运算符替换为廉价的运算符。3）常量折叠（constant folding），以便我们可以将常数表达式替换为其值。这些优化方法考虑一个节点序列，然后利用不同类型节点的可交换性、结合性和分配性等特性，简化计算过程。

除了常见的运算符（+、×等），代数简化优化还可以应用于DL特定的运算符（例如reshape、transpose和pooling）。这些运算符可以重新排序，有时还可以消除，从而减少冗余并提高效率。下面我们列举了可以应用代数简化的常见情况：

1）计算顺序优化：在这种情况下，优化器根据特定特性查找并移除reshape/transpose操作。以矩阵乘法（GEMM）为例，有两个矩阵（例如A和B），两个矩阵都进行了转置（分别产生A转置和B转置），然后将A转置和B转置相乘。然而，更高效的实现GEMM的方法是交换参数A和B的顺序，相乘后再对GEMM的输出进行转置，这样可以将两个转置(transpose)操作简化为一个操作。

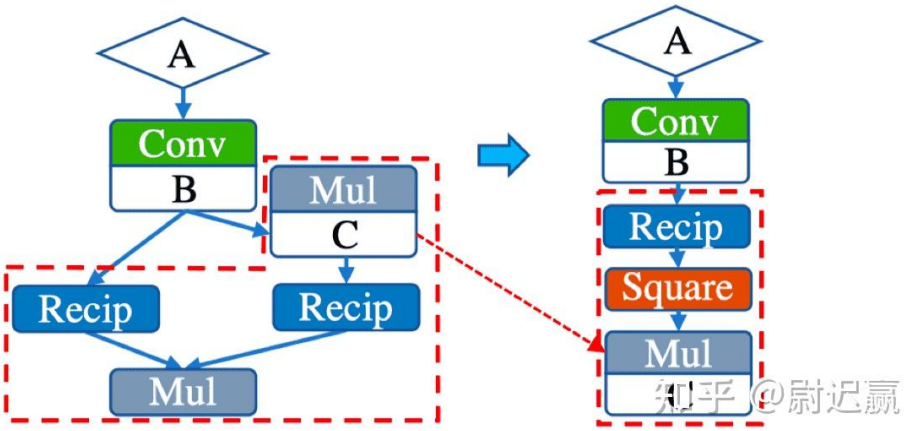
2）节点组合优化：在这种情况下，优化器将连续的多个transpose节点合并为一个节点，消

除恒等转置节点，并在实际没有移动数据时将transpose节点优化为reshape节点。

3）ReduceMean节点优化：在这种情况下，优化器会将ReduceMean节点替换为AvgPool节点（例如在Glow中），如果reduce操作的输入是4维且需要对最后两个维度进行reduce。

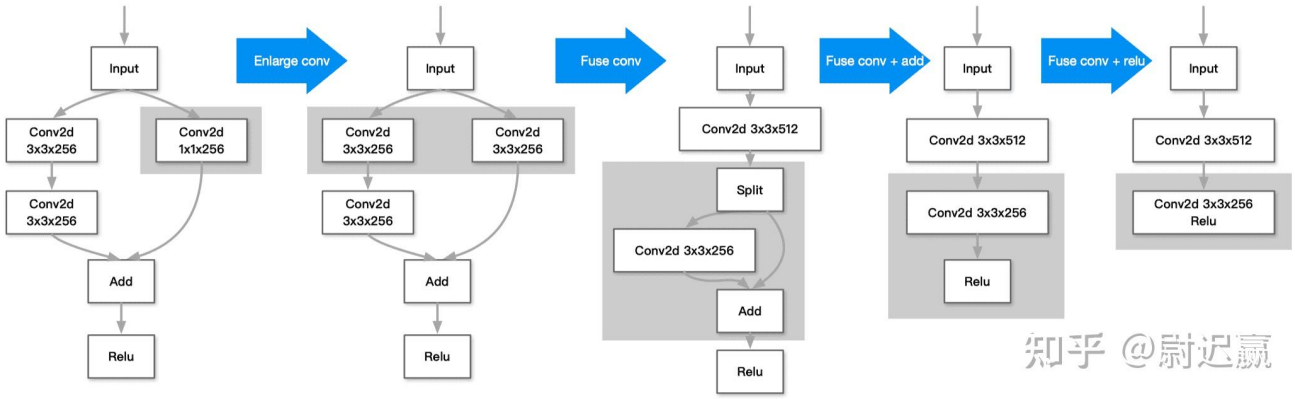
- 结合律化简：

$$(A \circledast B)^{-1} \odot ((A \circledast B)C)^{-1} \rightarrow (A \circledast B)^{-2} \odot C$$



Operator fusion

运算符融合是DL编译器中不可或缺的优化方法。它可以更好地共享计算，消除中间分配，通过组合循环嵌套来促进进一步优化 [78]，同时减少启动和同步开销 [91]。在TVM中，运算符被分为四个类别：**injective（可注入）、reduction（归约）、complex-out-fusible（复杂输出可融合）和opaque（不透明）**。当定义运算符时，其对应的类别也确定了。TVM针对上述类别设计了跨运算符的融合规则。在TC中，融合根据自动多面体变换的不同方式执行。然而，如何识别和融合更复杂的图模式，例如具有多个广播和归约节点的块，仍然存在问题。最近的研究 [61, 62] 尝试解决这个问题，并提出了一个框架来探索和优化激进的融合计划。该框架不仅支持逐元素和归约节点，还支持具有复杂依赖关系的其他计算/内存密集节点。



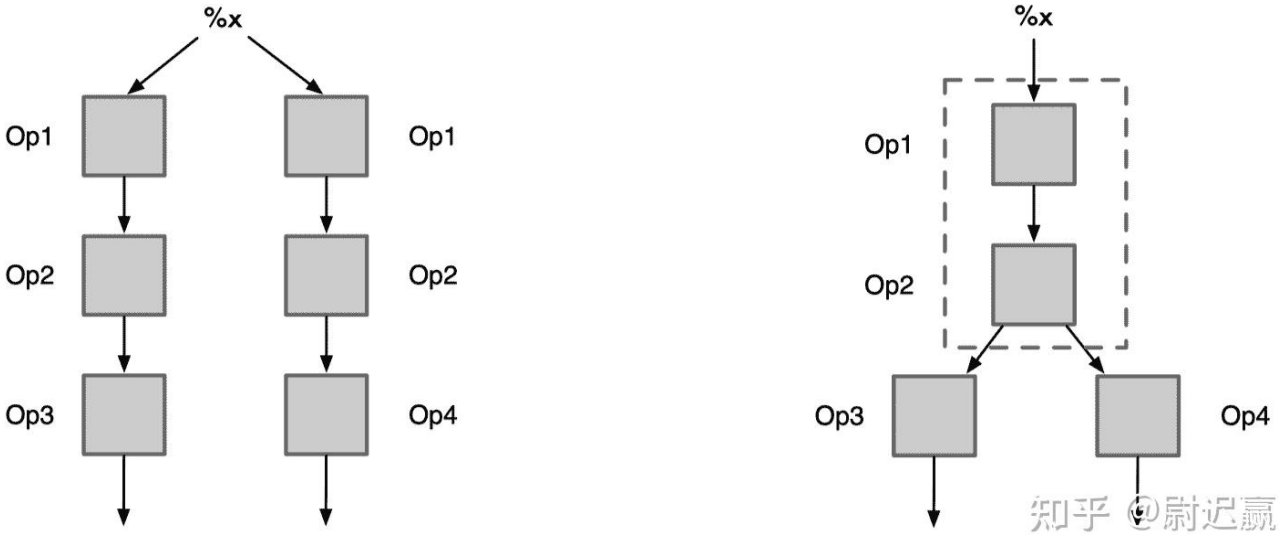
Operator sinking

这种优化将transpose等操作下沉到batch normalization、ReLU、sigmoid和channel shuffle等操作的下方。通过这种优化，许多类似的操作被移动到彼此更近的位置，为代数简化提供了更多的机会。

4.3.3 Dataflow-level optimizations

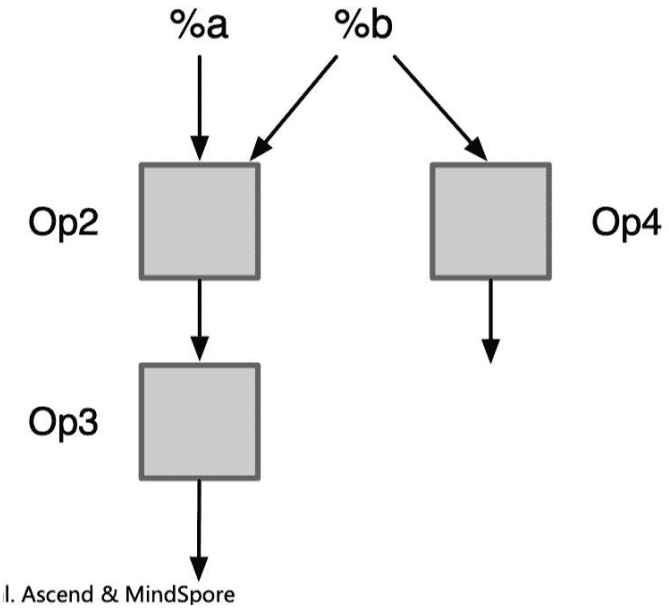
Common sub-expression elimination (CSE) : 公共子表达式消除

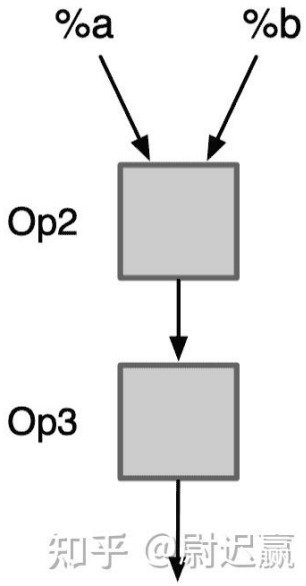
如果表达式E的值在先前计算过，并且自从上次计算以来E的值没有改变，那么表达式E就是一个公共子表达式[6]。在这种情况下，E的值只需计算一次，并且先前计算过的E的值可以用于避免在其他地方重新计算。DL编译器通过整个计算图搜索公共子表达式，并用先前计算的结果替换以下公共子表达式。



Dead code elimination (DCE) : 死代码消除

如果计算结果或副作用没有被使用，一段代码就是无效的(dead code)。DCE优化会删除无效的代码。无效代码通常不是由程序员引起的，而是由其他图优化引起的。因此，无用代码消除 (DCE) 和常量表达式消除 (CSE) 在其他图优化之后应用。其他优化，例如无用存储消除 (DSE)，可以消除永远不会使用的张量的存储，也属于DCE优化的范畴。





Static memory planning

静态内存规划优化旨在尽可能重用内存缓冲区。通常有两种方法：原地内存共享和标准内存共享。原地内存共享为操作的输入和输出使用相同的内存，仅在计算之前分配一份内存副本。标准内存共享则在不重叠的情况下重用先前操作的内存。静态内存规划是离线进行的，可以应用更复杂的规划算法。最近的一项工作[4]首次设计并执行了内存感知调度，以在边缘设备上最小化峰值激活内存占用，这为在内存受限设备上进行内存规划提供了新的研究方向。

经典方法：Greedy by Size for Offset Calculation（https://arxiv.org/pdf/2001.03288.pdf）

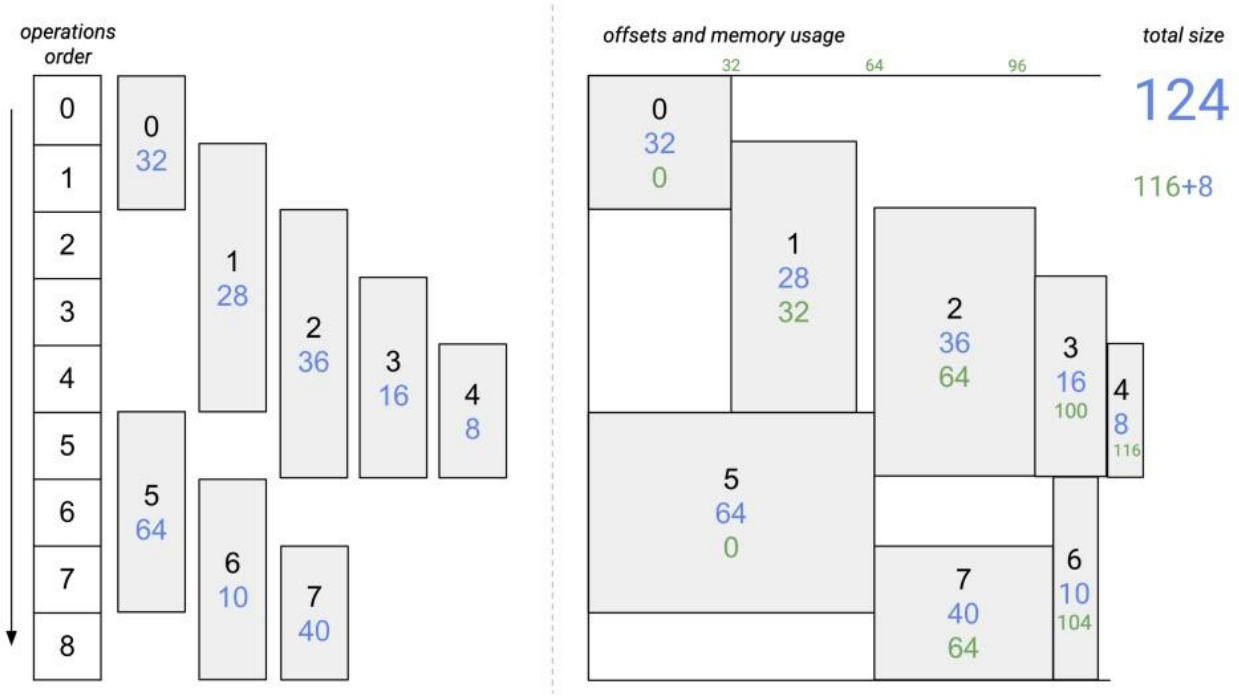


Figure 6. Greedy by Size for Offset Calculation on the neural network in Figure 1.

Layout transformation

布局转换试图找到在计算图中存储张量的最佳数据布局，然后在图中插入布局转换节点。值得注意的是，实际的转换不是在这里执行，而是在编译器后端评估计算图时执行。

实际上，不同数据布局下相同操作的性能是不同的，并且在不同的硬件上最佳布局也不同。例如，在GPU上以NCHW格式执行的操作通常更快，因此在GPU上进行到NCHW格式的转换是高效的（例如TensorFlow）。一些深度学习编译器依赖于硬件特定的库以实现更高的性能，而这些库可能需要特定的布局。此外，一些深度学习加速器偏向于更复杂的布局（例如，tile）。此外，边缘设备通常配备了异构计算单元，不同的计算单元可能需要不同的数据布局以实现更好的利用率，因此布局转换需要仔细考虑。因此，编译器需要提供一种跨各种硬件进行布局转换的方法。

张量的数据布局不仅对最终性能有着重要影响，而且转换操作本身也会带来显著的开销。因为它们还会消耗内存和计算资源。

最近的一项基于TVM针对CPU的工作[58]改变了计算图中所有卷积操作的布局，首先改为NCHW[x]c的格式，其中c表示通道C的分割子维度（sub-dimension），x表示子维度的分割大小。然后，在进行硬件特定的优化时，通过自动调优探索全局的x参数，同时提供硬件细节，如缓存行大小、向量化单元大小和内存访问模式。

NCHW -> NHWC

4.3.4 Discussion

前端是深度学习编译器中最重要的组成部分之一，负责将深度学习模型转换为高级IR（例如计算图），并基于高级IR进行硬件无关优化。尽管不同深度学习编译器的前端在高级IR的数据表示和操作符定义上可能有所差异，但硬件无关优化主要分为三个层次：节点级别、块级别和数据流级别。在每个层次上，优化方法利用深度学习特定和通用的编译优化技术，以减少计算冗余并改善计算图级别上深度学习模型的性能。