5. 深度学习编译器 - 图优化（续）

图优化是深度学习编译器优化中非常重要的一个过程，上次因为时间关系没有写完，本节继续。

5.3 代数化简

代数化简的目的是利用交换率、结合律等规律调整图中算子的执行顺序，或者删除不必要的算子，以提高图整体的计算效率。 比如，在下面的例子中，sub_module_1 在输出前将结果 Reshape 为三维 Tensor；而 sub_module_2 要求输入是二维 Tensor，因此一开始就会对输入 Tensor 进行一次 Reshape 操作。将sub_module_1 和 sub_module_2 级联起来后网络中会有两次连续的 Reshape 操作，但很明显这里只需要做一次 Reshape 即可。

通常有两种场景会导致出现例子所示情况。第一种就是例子中所示的封装，随着网络变的越来越复杂，不可避免的要调用其他已经封装好的子网络，为了适配子网络的接口，就会加入 Reshape、Transpose 等操作。第二种是编译器在图优化过程中插入了新的算子，而插入的新的算子和已有的算则之间存在冗余。

```
def sub_module_1():
    ...
    ret = ...
    return tf.reshape(ret, [-1, n, k])

def sub_module_2(inputs):
    i0 = tf.reshape(inputs[0], [-1, k])
    xxxx

def main():
    s1 = sub_module_1()
    s2 = sub_module_2([s1])
    xxx
```

下表列举了一些 Tensorflow 中支持的代数化简操作：

优化前	优化后
Add(const_1, Add(x, const_2))	Add(x, const_1 + const_2)
Conv2D(const_1 * x, const_2)	Conv2d(x, const_1 * const_2)
Concat([x, const_1, const_2, y]	Concat([x, concat([const_1, const_2]), y]
X + zeros X / ones X * ones	X
Matmul(Transpose(x), y)	Matmul(x, y, transpose_x = true)

Cast(Cast(x, dtype1), dtype2)	Cast(x, dtype2)
Reshape(Reshape(x, shape1), shape2)	Reshape(x, shape2)
AddN(x * a, b * x, c * x)	x * AddN(a + b + c)
(matrix1 + scalar1) + (matrix2 + scalar2)	(matrix1 + matrix2) + (scalar1 + scalar2)

代数化简可以通过子图替换的方式完成。具体实现时，可以先抽象出一套通用的子图替换框架，再对各规则实例化。也可以针对每一个具体的规则实现专门的优化逻辑。实际系统中的实现通常是两者的结合。

5.4 算子融合

算子融合的目的是将几个小 OP 融合为一个大 OP，达到减少从内存/显存中搬移数据的目的。举例来说，假设要计算 Relu(X + Y），X 和 Y 的长度均为 L。在 Tensorflow 中会对应到 2 个 OP：Add 和 Relu。做 Add 计算时，先要将 X 和 Y 从内存/显存中读出，然后再将计算结果写入到内存/显存，因此 Add 计算需要从内存中读写的数据量为 3L。做 Relu 计算时，同样需要将输入数据从内存/显存中读出，然后再将结果写回到内存/显存，因此 Relu 计算需要从内存中读写的数据量为 2L。Add 和 Relu 加起来读写数据的总量为 5L 。如果我们将其融合为一个 OP，将 X 和 Y 从内存/显存中读出后，先做加法，然后做 Relu 计算，再将最后的结果保存到内存/显存，这样读写数据的总量仅为 3L，相比融合前减少 40%。

下表列举了几个 Tensorflow 中支持的算子融合：

Conv2D + BiasAdd + <Activation>
Conv2D + FusedBatchNorm + <Activation>
Conv2d + Squeeze + BiasAdd
Matmul + BiasAdd + <Activation>

注：<Activation> 表示该 OP 是可选项。

从上面的例子可以看出，算子融合有效果需要满足几个前提条件。首先，中间结果只能写回到最后一级存储。如果 Add 的计算结果可以保存到高速缓存中，读取速度非常快，那融合前读写内存/显存的数据量也可以近似认为是 3L，而非 5L。其次，最终结果只依赖于输入数据的部分数据，而非所有数据，否则没有办法在减少访存的情况下实现融合后的计算逻辑。最后，从内存/显存中读写数据的耗时占整个 OP 计算过程中的占比较高，只有满足这个条件，减少数据搬运量才能起到加快图运行的效果。

和代数化简等其他图优化手段相比，算子融合有一个很大的不同：引入了新的融合后的 OP。因此算子融合的难点在与如何为融合后的 OP 实现计算逻辑？目前有两种模式, 一种是将融合后

的算子做为普通算子, 为他们实现定制 Kernel; PaddlePaddle, Tensorflow 等框架本身采用的是这种方式。 但这种模式开发成本较高，相对来说能够支持的融合模式有限。 还有一种模式是和自动代码生成结合, 直接生成可以运行的代码。 TVM 和 XLA 采用的是这种方式。