AGO: Boosting Mobile AI Inference Performance by Removing Constraints on Graph Optimization

INFOCOM 2023 CCFA

万嘉诚 2024/1/30

背景

复杂算子间的融合往往不被主流的深度学习编译器所考虑，这里论文举例称下图中轻量的Op3与Op4因为复杂算子Op1和Op2的并行结构被迫单独执行。但这个例子不好，只要Op1最先执行，Op2，Op3以及OP4便可以融合。

另一个方面，如Op5与Op6所示，只有极少数的后继融合被考虑

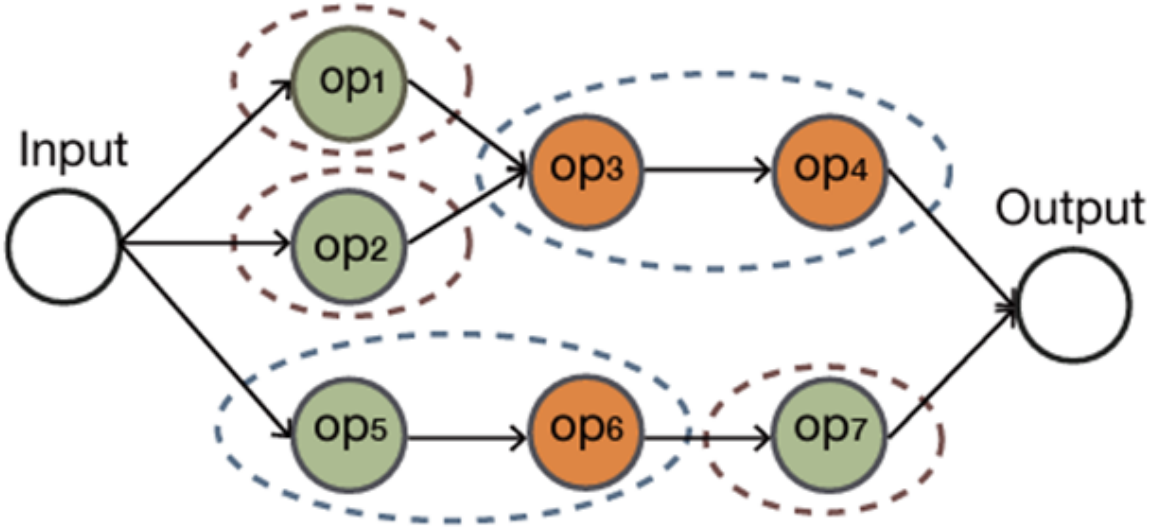


Fig. 1. An illustrative computational graph.

总览

作者从上述观察出发提出了基于迭代空间对复杂算子做两阶段调优方案，并使用分而治之的策略实现融合子图的构建

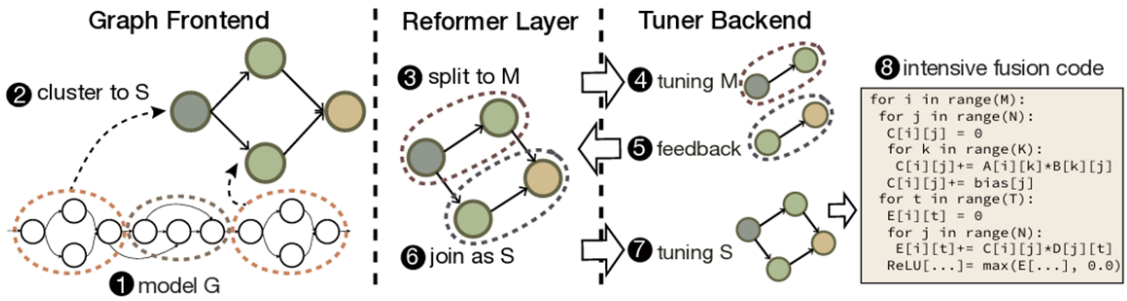


Fig. 2. System overview of AGO.

复杂算子融合

实现复杂算子的融合首先便要规避冗余读取与冗余计算，如下图所示，其对两个连续的卷积Conv1与Conv2的融合导致Conv1出现了冗余计算

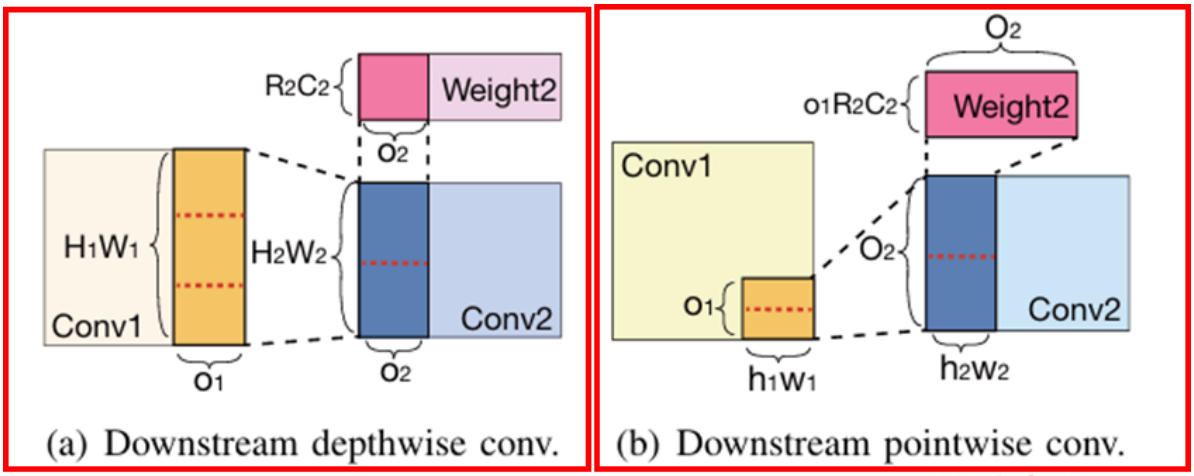
```
for n, o2, h, ow in range(N, O2, H2, W2 // 16):
 # intra-tile loops, compute a tile of Conv1
 for o1, ih1, iw1 in range(O1, 1 + R2 - 1, 16 + C2 - 1):
  Conv1[n, o1, h+ih1, ow*16+iw1] = 0.0
  for ri, rr1, rc1 in range(I, R1, C1): # reduction
   Conv1[n, o1, h+ih1, ow*16+iw1] += ...
 # intra-tile loops, compute a tile of Conv2
 for io2, ih2, iw2 in range(1, 1, 16):
  Conv2[..., ow*16+iw2] = 0.0
  for ro, rr2, rc2 in range(O1, R2, C2): # reduction
   Conv2[...] += \
    Conv1[..., ow*16+iw2+rc2] * Weight2[...]
```

每计算Conv2的一个tile，Conv1的相关数据都会被冗余计算一次

Fig. 5. Intensive fusion program of two convolutions.

我们对这一现象做形式化定义，记前后两个算子的迭代空间为GS_1与GS_2，对应的计算量为$|GS_1|$与$|GS_2|$，记经过tiling后，内部tile的迭代空间为$|TS_1$与$TS_2|$，则经过融合后前一个算子的计算量为$|GS_2/TS_2 \times (GS_1/TS_1 - GS_2/TS_2)| * |TS_1|$，其中$\times$为笛卡尔积，$/$为其逆运算。

当$GS_1/TS_1 - GS_2/TS_2 \neq \emptyset$或$|TS_2| < |TS_1|$便会产生冗余计算，而前者受到算子类型影响，无法改变，因此我们降低冗余计算所努力的目标应该为尽量使$|TS_2| \geq |TS_1|$，本质上是一次性计算/取出一次归约所需数据

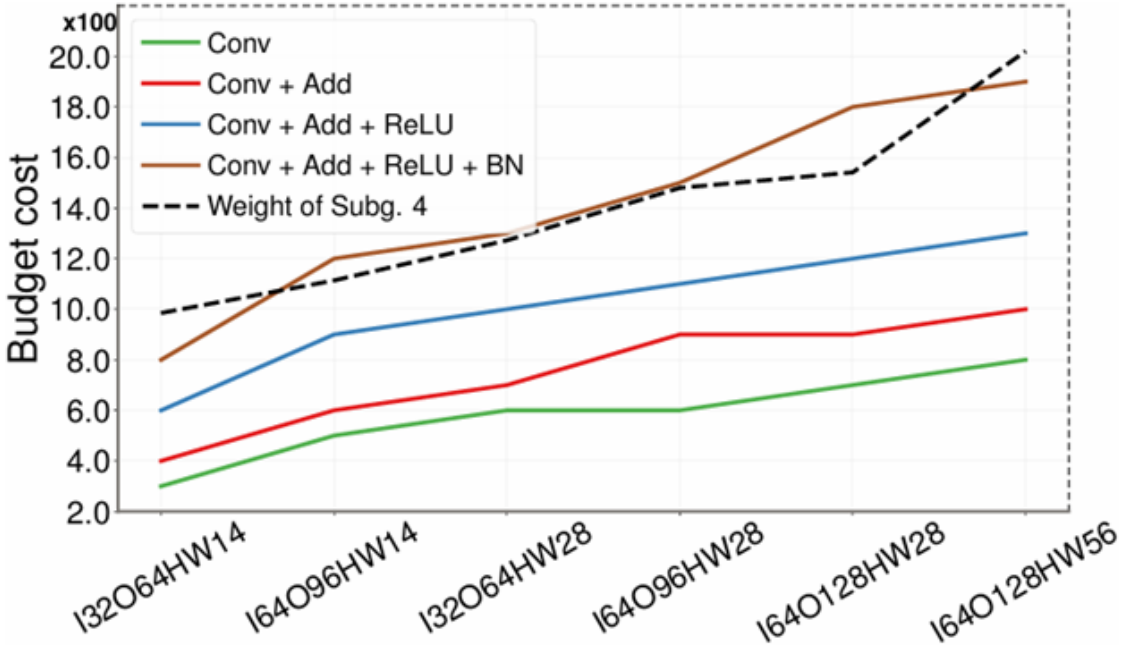


$|TS_2| = |TS_1|$，Conv1数据为一次规约所需

$|TS_2| > |TS_1|$，Weight2数据为一次规约所需

子图划分

通过算子赋权与分而治之两种手段实现高效的子图划分



算子赋权：观察到算子复杂度主要受以下两个因素影响：算子的Loop数与每个Loop的范围，则给出如下公式为算子赋权，其中L为算子Loop集，s为Loop范围

$$w_v = c \times \prod_{l \in L_v} \log(s_l) + b\,,$$

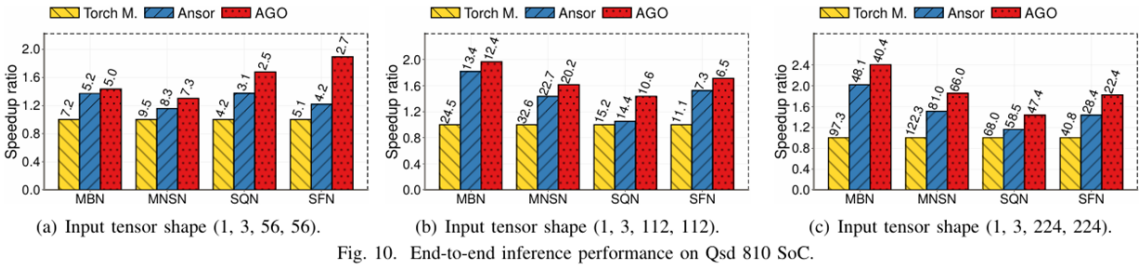
相关的子图划分算法为：

1. 根据输入模型生成候选节点集与阈值T
2. 从候选节点集合中取出权值最大的候选节点
3. 贪婪地为候选节点从其相邻节点中选择权值最小的节点融合
4. 若权值之和小于阈值，则融合成功，融合子图成为新的候选节点

分而治之：为上文中所得子图进行调优仍是一个艰巨的任务，为此作者实现了一个进一步划分子图的方案，也即通过Split函数将当前子图切割为若干只包含一个复杂算子的迷你子图，根据反馈逐步融合

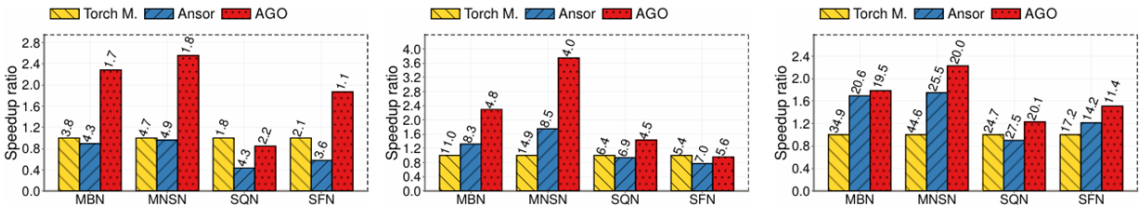
实验结果

端到端时延：加速效果与输入规模无明显关系



(a) Input tensor shape (1, 3, 56, 56). (b) Input tensor shape (1, 3, 112, 112). (c) Input tensor shape (1, 3, 224, 224).

Fig. 10. End-to-end inference performance on Qsd 810 SoC.

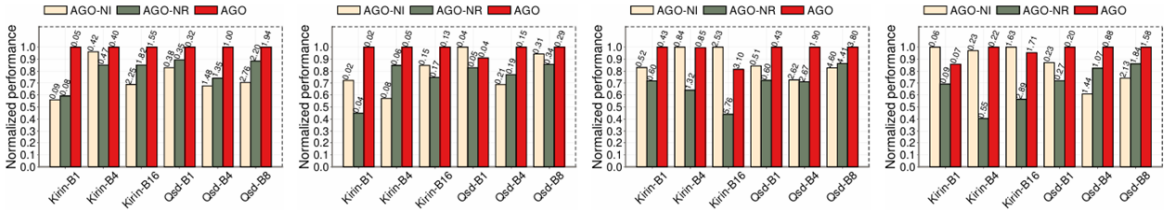


(a) Input tensor shape (1, 3, 56, 56). (b) Input tensor shape (1, 3, 112, 112). (c) Input tensor shape (1, 3, 224, 224).

Fig. 11. End-to-end inference performance on Kirin 990 SoC.

消融实验：

1. NI为无复杂算子融合的版本，NR为无Reformer Layer的版本
2. NR版本效果最差，这是因为难以找到有效的融合子图
3. D中存在NI优于AGO的情况，这是两个Pointwise Conv的内禀冗余计算过多



(a) Two depthwise convolutions. (b) Depthwise conv & pointwise conv. (c) Pointwise conv & depthwise conv. (d) Two pointwise convolutions.

Fig. 13. Subgraph performance with different variants of AGO.

融合力度提升明显：融合子图的规模主要集中在含7-11个算子的范围

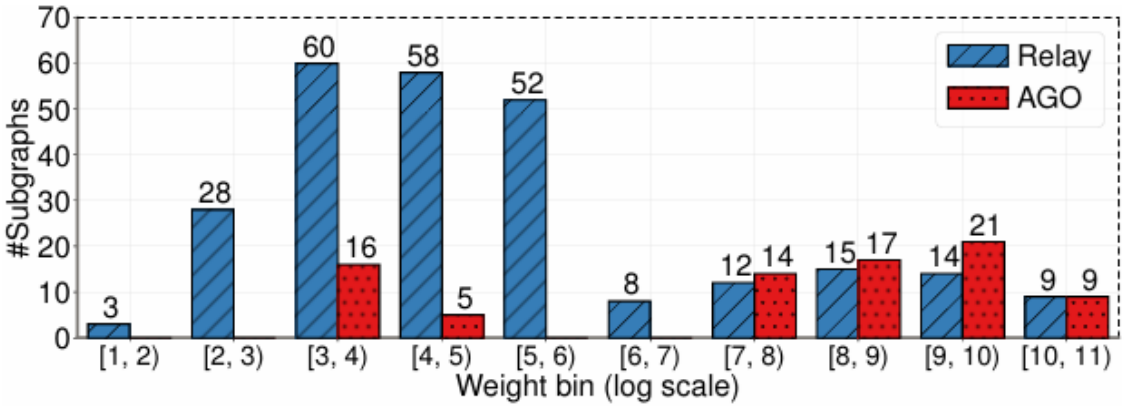


Fig. 14. Subgraph weight distribution for MVT.

讨论

算子融合：没有新意，采用工业界广泛使用的思路

子图划分：两阶段子图划分，基于迭代空间为算子赋权，提出贪婪地第一阶段子图划分策略；采用分而治之的思路提出第二阶段子图划分策略，然而第二阶段关于如何join语焉不详