AI编译器-系列之前端优化

公共子表达式消除

ZOMI

Talk Overview of Frontend Optimizer

I. AI 编译器前端优化

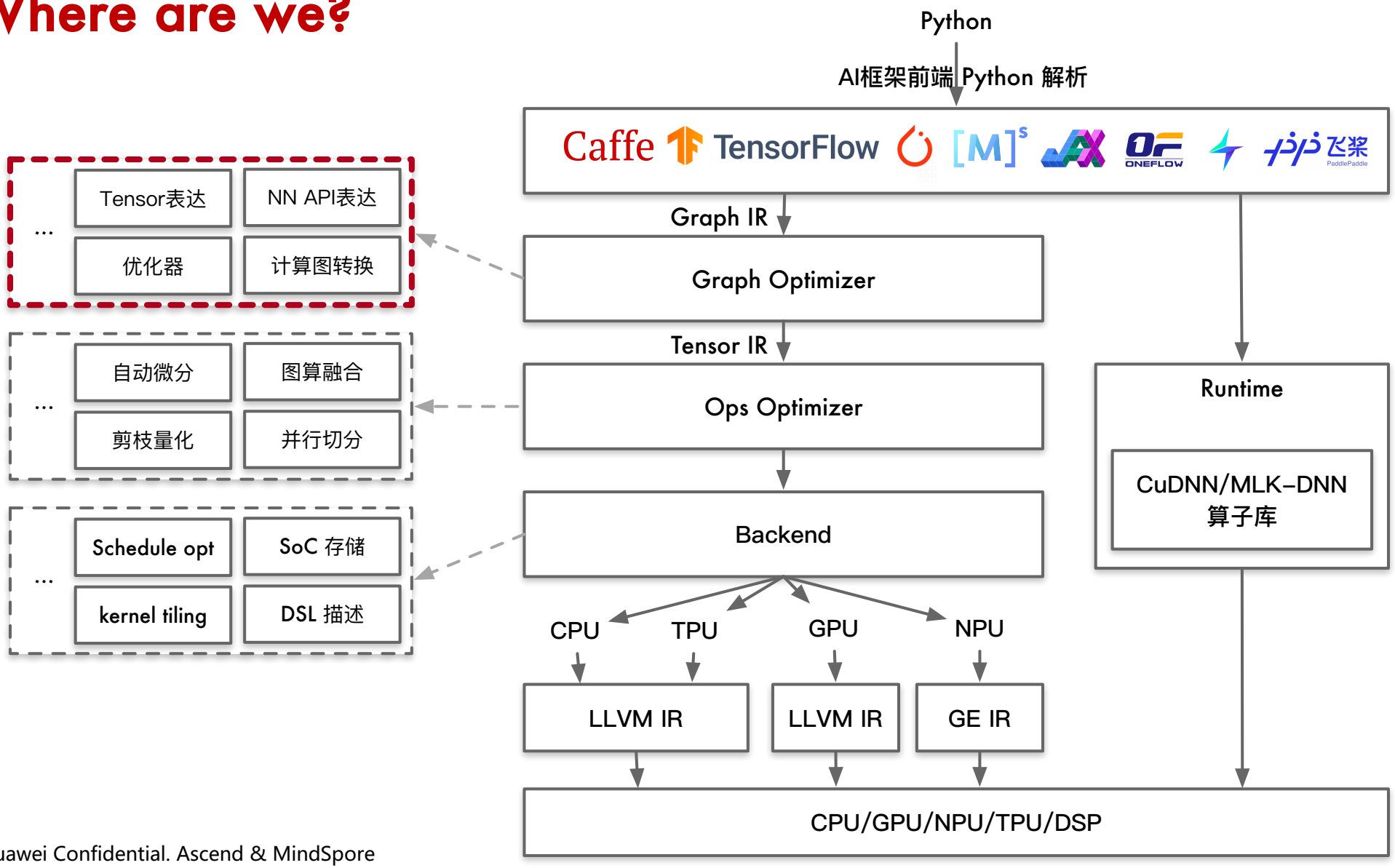
- 图层 - Graph IR
- 算子融合 - OP Fusion
- 布局转换 - Layout Transform
- 内存分配 - Memory Allocation
- 常量折叠 – Constant Fold
- 公共子表达式消除 - CSE
- 死代码消除 - DCE
- 代数简化 - ARM

Talk Overview

Common Subexpression Elimination – 公共子表达式消除

- CSE的概念定义
- AI编译器中的CSE

Where are we?



公共子表达式消除概念定义

公共子表达式消除

- CSE（Common subexpression elimination），是一个编译器优化技术。在执行这项优化的过程中，编译器会视情况将多个相同的表达式替换成一个变量，这个变量存储着计算该表达式后所得到的值。

```
1
2    a = b * c + g
3    d = b * c + e
4
```

公共子表达式消除

```
1
2    a = b * c + g
3    d = b * c + e
4
```

- 可以观察到 $b * c$ 是两项表达式中的公共子表达式。如果计算这个子表达式并将其计算结果存储起来的开销，低于重复计算这个子表达式的开销，则能够将以上代码转换成以下代码：

```
4
5    temp = b * c
6    a = temp + g
7    d = temp + e
8
```

Principle

执行这项优化的可能性基于表达式的定义可达性。当以下条件成立，则一个表达式 $b * c$ 在程序的某个点 p 被定义为是可达的：

- 从初始节点到点 p 的每条路径在到达 p 之前计算过 $b * c$ ；
- $b * c$ 被计算后，无论 b 或者 c 到达 p 以前都没有被重新赋值过；

Principle

- 由编译器计算的成本效益分析可以判断出，重复计算该表达式的开销是否大于存储该表达式的计算结果，并且这个分析也要将寄存器等因素考虑在内。

Principle

编译器开发者将公共子表达式消除分成两种：

- **本地公共子表达式消除**：这项优化技术工作于基本块之内。
- **全局公共子表达式消除**：这项优化技术工作于整个过程之中。

Algorithm

对于语句 $\%s:\ z\ =\ x\ op\ y$，若 $x\ op\ y$ 在语句 s 之前可用，则：

1. 从 s 开始逆向搜索 IR，找到距离 s 最近执行 $\mathrm{w} = x\ op\ y$ 的语句
2. 建立临时变量 $\%u$
3. 把步骤 I 中找到的语句 $\mathrm{w} = x\ op\ y$ 进行替换：

$$\%u = x\ op\ y$$

$$\%w = u$$

4. 使用 $z = u$ 替换s，重复步骤 I 直到遍历完 IR

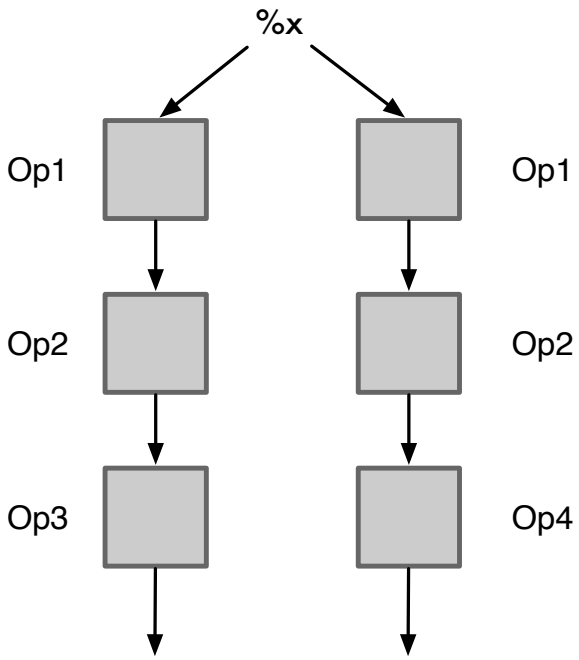
AI编译器的公共子表达式消除

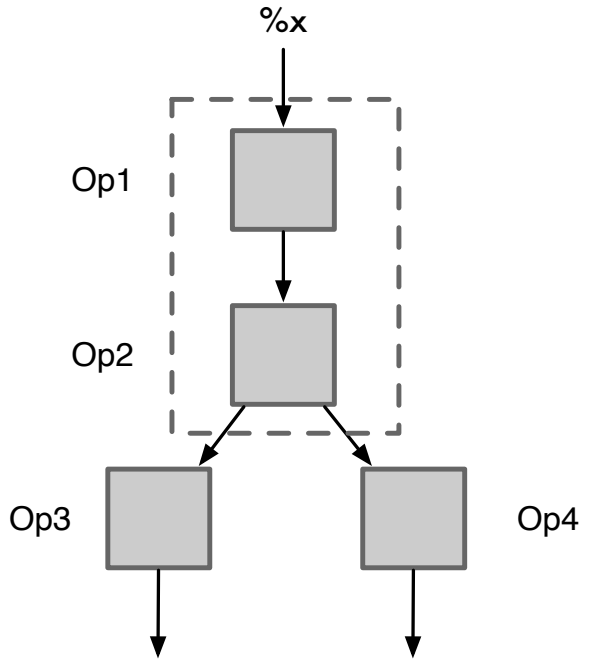
What is ?

- 对于公共子表达式，只需要计算其中一个表达式的值，其他表达式的值可以通过赋值得到。
- 这个过程就称作公共子表达式消除，它是一种传统编译器中常用的优化手段，经过迁移也可以应用到深度学习编译器中。

What is ?

AI 编译器中公共子表达式消除采取相同的思路，区别在于AI编译器中子表达式是基于计算图或者图层 IR。通过在计算图中搜索相同结构的子图，简化计算图的结构，从而减少计算开销。





Principle

- 通过建立候选哈希表 MAP, 记录已处理过的同一种类型的 OP。对于当前正在处理的 OP，先查找该 MAP 表，如果能找到其他和正在处理的 OP 类型相同的 OP，则对其进行遍历，如果其中某个 OP 的输入和参数与当前正在处理的 OP 相同，则它们为公共子表达式，结果可以互相替代；如果所有 OP 都不能与当前正在处理的 OP 匹配，则将当前 OP 复制一份回。

Algorithm

输入：计算图 Graph IR，找到公共子表达式并优化计算图：

1. 获取逆后续节点集 Set(Reverse)：对计算图 Graph IR 进行深度优先遍历 DFS，将结果逆转得到逆后续节点集。逆后续得到的结果就是拓扑排序，即访问到某一个节点时，该节点的依赖节点都已经被访问。
2. 创建 Map(MSE)：存储公共子表达式候选集，遍历 Set(Reverse) 时，可以从候选集 Map(MSE) 中查找是否有可以使用的表达式。
3. 遍历计算图的所有节点，判断是否有公共子表达式：获取节点 hash 值，hash的 key 由输出个数 + 输出类型s + 输入节点 id，key 可以保证输入相同及输出个数和类型相同时，得到的hash值相同，达到检索公共表达式的目的。
4. 记录入候选集：根据 hash 值 h 从 Map(MSE) 中得到候选集，当候选集为空，第一次遇到这样的表达式，将节点 Node 存入 Map(MSE) 中。

Algorithm

5. 可复用公共子表达式判断：Candidate 非空，则说明之前遍历到了相似的表达式，进一步判断是否可复用已保存节点表达式的结果。即节点的输入都是来自相同的Const节点，可以保证输入数据完全相同；输出个数和类型相同，可以保证输出的结果相同；满足条件下，可复用之前保存节点的结果，不用再次进行计算。
6. 删除重复节点：判断表达式可以复用后，最后一步就是删除重复的节点，将 Candidate 的输出连接到当前节点的输出节点对应的输入，最后删除当前节点。

Reference

1. Common Subexpression Elimination – Code optimization Technique in Compiler Design https://www.geeksforgeeks.org/common-subexpression-elimination-code-optimization-technique-in-compiler-design/
2. Steven Muchnick; Muchnick and Associates. . Morgan Kaufmann. 15 August 1997. ISBN 978-1-55860-320-2. Common subexpression elimination.
3. https://www.tensorflow.org/jvm/api_docs/java/org/tensorflow/proto/framework/OptimizerOptions.Builder
4. https://en.wikipedia.org/wiki/Common_subexpression_elimination