针对神经网络的编译器和传统编译器的区别和联系是什么？ - 知乎

https://www.zhihu.com/question/396105855/answer/1868408680

金雪锋关注AI和基础软件产业，负责AI框架MindSpore的设计 Mon Jun, 28 21:23

1）首先聊聊神经网络编译器出现的背景和历史。

1、早期深度学习框架，重点是框架和库，与编译器关系相对较弱

比如Tensorflow早期版本，在神经网络/深度学习的编程模型上，主要进行了graph/图和op/算子两层抽象

- 图层通过声明式的编程方式，然后通过静态图的方式进行执行，这里其实也做了一些编译器的事情，这里包括硬件无关和硬件相关的优化：硬件无关的优化包括编译器通用的优化，如表达式化简、常量折叠，也包括与深度学习/神经网络强相关的，如自动微分等；硬件相关的优化包括简单的算子融合、内存分配优化等。
- 算子层通常采用手写的方式，比如GPU上基于CUDA/cuDNN。

这种方式遇到几个问题：

- 表达上，语法不是Python原生的，算法工程师使用的易用性不够好
- 更多的Transform出现，比如并行、量化、混合精度等
- 算子粒度和边界提前确定后，无法充分发挥硬件的性能
- 硬件厂商提供的算子库也不一定是性能最优的，在SIMT和SIMD的架构中，scheduling、tilling都是有很大的空间，在具体到一个模型，shape确定的情况下，开发者还有可能开发出性能更高的算子。
- AI专用芯片出现（Google TPU、华为Ascend等），3与4的情况加剧。

2、后期引入大量编译器的技术进行改进

- **表达上的改进(Pytorch/TorchScript、JAX)**

Pytorch的Eager Model是一种解决易用性的方案，虽然基本上还是图层和算子两层的抽象，但是整个语法基本上是Python Native的，让算法工程师比较容易上手；不过这个方案在运行的时候基于Python解释器的能力，不是一种高性能的解决方案，本身与神经网络的编译器关系不大；但是其表达的方式成为后面框架参考的标杆，图层的神经网络编译器主要就是考虑如何把这样表达转换到图层的IR进行优化，目前主要有两种方式：

AST-Based：以Pytorch TorchScript为例，主要通过Python的修饰符，把Python代码的AST拿到，然后变换成图层的IR，进行编译优化。

Tracing-Based：以JAX为例，主要把Python代码假执行一遍，保存执行序列，基于执行序列变换到图层IR进行编译优化。

两种方案各有优缺点，第一种方案实现复杂，第二种方案在一些处理上有限制(比如控制流的处理)。

- **性能上的优化(XLA/TVM/TC)**

性能上的优化思路其实比较统一，就是打开图和算子的边界，进行重新组合优化。

XLA：基本上的思路是把图层下发的子图中的算子全部打开成小算子，然后基于这张小算子组成的子图进行编译优化，包括buffer fusion、水平融合等，这里的关键是大算子怎样打开、小算子如何重新融合、新的大的算子(kernel)怎样生成，整体设计主要通过HLO/LLO/LLVM层层lowering实现，所有规则都是手工提前指定。

TVM：分为Relay和TVM两层，Relay主要关注图层，TVM主要关注算子层，总体思路与XLA是类似的，也是拿到前端给一张子图进行优化，Relay关注算子间的融合、TVM关注新的算子和kernel的生成，区别在于TVM是一个开放的架构，Relay目标是可以接入各种前端，TVM也是一个可以独立使用的算子开发和编译的工具（基于Halide IR，最新演进到自己定义的TIR），TVM在算子实现方面采用了compute和schedule分离的方案，开发人员通过compute来设计计算的逻辑，通过schedule来指定调度优化的逻辑。

TC(Tensor Comprehensions)：开发者发现算子的计算逻辑的开发是比较容易的，但是schedule的开发非常困难，既要了解算法的逻辑又要熟悉硬件的体系架构，更重要的是，前面提到图算边界打开后，小算子融合后，会生成新的算子和kernel，这些新的算子compute是容易确定的（小算子compute的组合），但是schedule却很难生成，所以传统的方法就是事先定义一大堆schedule模板，万一组合的新算子不在模板之内，性能就可能比较差，甚至出错；那TC则希望通过Polyhedra model实现auto schedule，降低开发门槛，当然这个项目基本已经停更了，但是类似的工作在MLIR、MindSpore上还在不停发展。

- **图层和算子层的IR表达**

在神经网络编译器发展过程中，有多种IR的出现，各有特点：

图层IR：朴素的DataflowIR、函数式IR、函数式图IR、SSA风格IR

算子层IR：HalideIR、LLVM等

图算融合表达：MLIR

以前分析过图层IR，供参考：

https://zhuanlan.zhihu.com/p/263420069zhuanlan.zhihu.com

2）回到题目本身

1、神经网络编译器与传统编译器的相同点

神经网络编译器和传统编译器一样，也是有前端表达、硬件无关优化和硬件相关优化、最后的codegen等，整体结构是类似的，这一块就不多展开了。

2、神经网络编译器与传统编译器的区别

主要体现在神经网络编译器像数据库的SQL引擎/向量化引擎一样是一个特定领域的编译器，这些领域特征包括：以Python为主的动态解释器语言的前端、多层IR设计（图层/算子层/codegen）、面向神经网络的特定优化（自动微分、量化/混合精度、大规模并行、张量运算/循环优化等）。

- **编译前端解析**

与传统编译器不同，神经网络编译器通常不需要lexer/parser，而是基于前端语言（如Python）的AST将模型解析并构造为计算图IR，侧重于保留shape、layout等Tensor计算特征信息，当然部分编译器还能保留控制流的信息。

这里的难点在于，Python是一种灵活度极高的解释执行的语言，像弱类型、灵活的数据结构等，而神经网络编译器本质上是偏静态，两者之间的完全转化是不大可能的。

- **多层IR设计**

为什么需要多层IR设计，主要是为了同时满足易用性与高性能这两类需求。为了让开发者使用方便，框架前端(图层)会尽量对Tensor计算进行抽象封装，开发者只要关注模型和粗粒度OP；而在后端算子性能优化时，又可以打破算子的边界，从更细粒度的循环调度等维度，结合不同的硬件特点完成优化。因此，多层IR设计无疑是较好的选择。

High-level IR（图层IR），如XLA的HLO，TVM的Relay IR以及MindSpore的MindIR等，重点关注非循环相关的优化。除了传统编译器中常见的常量折叠、代数化简、公共子表达式等优化外，还会完成Layout转换，算子融合等优化，通过分析和优化现有网络计算图逻辑，对原有计算逻辑进行拆分、重组、融合等操作，以减少算子执行间隙的开销并且提升设备计算资源利用率，从而实现网络整体执行时间的优化。

Low-level IR，如TVM的TIR，HalideIR，以及isl schedule tree[7]等。针对Low-level IR主要有循环变换、循环切分等调度相关的优化，与硬件intrinsic映射、内存分配等后端pass优化。其中，当前的自动调度优化主要包含了基于搜索的自动调度优化（如ansor[8]）和基于polyhedral编译技术的自动调度优化（如TC和MindAKG[9]）。

有人可能会问，图层和算子层的表达和编译能否放在一起？也许可以，但是明显看到这样做面临几个挑战：

1、整图展开到原子算子，看上去编译的规模/复杂度指数级上升

2、显然图编译优化的问题和算子编译优化的问题是有明显的区别，一个关注变换和融合，另外一个关注循环优化，放在一起对编译器实现的复杂度是个比较大的挑战

3、要看到硬件供应商和框架供应商目前是分开的，两者总是需要一个边界。

- **面向神经网络的特定优化**

自动微分：BP是深度学习/神经网络最有代表的部分，目前相对已经比较成熟，基于计算图的自动微分、基于Tape和运算符重载的自动微分方案、基于source2source的自动微分都是现在主流的方案。

并行优化：随着深度学习的模型规模越来越大，模型的并行优化也成为编译优化的一部分，包括：数据并行、算子级模型并行、Pipeline模型并行、优化器模型并行和重计算等

张量计算/循环优化：循环优化其实是一个古老的编译器的难题，在高性能计算领域，循环优化已经研究了几十年，一直没有很好的解决，但是看上去，深度学习/神经网络领域的问题要简单一点，原因是这个领域大量的以Dense的矩阵运算为主，不像高性能计算领域那么复杂(大量稀疏/非规则的矩阵和向量运算)，这为循环优化带来了很大的空间，不过即便是这样，自动scheduling、自动tilling、自动向量化这些理想的方案和技术也还远远没有成熟。

量化/....：推理侧常用的一些变换，不展开了

3）神经网络编译器未来的方向探讨

编译器形态：也许需要两类编译器同时存在，一类是面向极致高性能的AOT编译器，同时这类编译器对NPU更加友好；另外一类是JIT编译器，适合与动态图配合；

IR形态：需不需要MLIR这种统一的形态？

自动并行：配合Cost model，提供自动并行优化的能力；

自动Scheduling/Tilling/Tensorizing：可能很难全部做到，能支持大部分也可以。

参考文献：

[1] Tensorflow. https://github.com/tensorflow/tensorflow

[2] MindSpore. https://gitee.com/mindspore/mindspore

[3] Tvm. https://tvm.apache.org/

[4] XLA. https://www.tensorflow.org/xla

[5]TensorComprehensions.https://github.com/facebookresearch/TensorComprehensions

[6] Li, Mingzhen and Liu, Yi, etc. The deep learning compiler: A comprehensive survey. IEEE Transactions on Parallel and Distributed Systems. 2020

[7] Polyhedral Compilation. https://polyhedral.info. Accessed February 4, 2020.

[8] Zheng, Lianmin, etc. Ansor: Generating high-performance tensor programs for deep learning. Symposium on Operating Systems Design and Implementation. 2020

[9] MindAKG. https://gitee.com/mindspore/akg