AI编译器的趋势和挑战 金雪锋

【P3】

第一个我想介绍一下AI编译器的一个概念，就是讲一下我对ai编译器的一个理解。我认为ai编译器的驱动力主要有三块，第一块就是python，我们整个的ai的框架都是在Python里面，所以说实际很大程度上ai编译器干的事情其实在加速python。第二个就是dsa，如果底下的硬件都是CPU，那ai编译器就跟普通编译器差不多了。第三个是深度学习，包括ai领域的业务怎么驱动。

【P4】

然后我们看一下Python，因为我是大概在零几年的时候就开始接触Python，但是我们那个时候还没有什么深度学习，python就是做一个测试工具。但是我看了一下现在的ai的框架，我发现其实AI框架做的事情我们以前都做过。对Python的加速总共有三种范式，第一个是CPython，是最主流的方式。Python很灵活，但是有些地方性能是不够的，怎么办？就加一个C++的库把它替换注册进去。整个的CPython这条路线，其实已经发展很多年了，它的方式非常的粗暴，就把Python解释器的内部的数据结构完全的开放给开发者，大概有1000多个，这种形式我觉得支撑了整个Python生态的发展。但这也有一个问题，就是这样也阻止了Python向JIT编译器的演进。

第二条路线是pay-pay，它希望提供一个Python的类似于Java的这种接口的编译器去提高Python语言执行的性能。但它很难成功，因为 CPython把内部的东西全暴露出去以后，大量的兼容性问题没办法解决，到现在为止它也只兼容了20%不到的结果。

第三条路线就是Numba的路线，这是一种对Python JIT编译器的一种妥协，做了一个旁路，既然没办法全部解决，就摘一些出来解决。

然后我们可以看一下整个的 AI框架的范式。早期，我觉得它就是典型的CPython，很简单，它的执行逻辑，很多语法就是依托Python，对于性能不行的算子就用 C。然后Tensorflow 1.0，我感觉是一种CPython和 Numba的一种结合。到后期我们都看到所有的框架，包括TorchScript，包括Tensorflow其实走的都是CPython和 Numba的一种结合。Numba它的目标没有c扩展，所有的Python的语句都可以lower到二进制去，但实际上这种情况在深度学习里面可能是不可行的，因为他的很多算子，比如说矩阵乘，用Python直接lower到二进制很难搞定。所以它最终就变成Python可以用JIT修饰符的方式来加速，然后性能高的算子，还是用C的方式去解决。

【P5】

第二个驱动力就是DSA的硬件加速这一块。这方面我觉得讨论的很多，我中午也跟唐博在讨论，就是说整个的我们的硬件，所有的都在做DSA的DSA化。传统的NPU，像华为的昇腾，里面有ai的CPU单元，也有vector，也有标量，也有矩阵。但是我们看到现在的CPU，包括苹果的，包括英特尔的出来以后，现在也加了矩阵单元了，所以说这两个架构其实都是类似的，无非就是说在NPU里面矩阵单元要多一点，在CPU里要少一点。所以说我们在NPU上遇到的问题，可能未来在CPU上也会遇到。对于整个编译要解决的问题，最早的像pytorch这种框架，他的算子跟图层是分开的。然后算子的管算子的，图层的管图层的，但其实这种方式是没法充分发挥dsa架构性能的。一方面它的算子性能不是最优的，我们还可以通过tuning，找出比厂家提供的算子是更优的，第二个就是算法工程师划分的算子粒度可能和硬件能够发挥最高性能的算子粒度是不一样的，所以说我们可以做融合。所以现在几个主流的编译器，像XLA就是最早开始做图算融合的，然后TVM在XLA的基础上可以做自定义，因为XLA基本上规则是确定的，后面像TC包括mindspore的akg，还希望schedule，包括tiling是自动化的，这是一系列的发展路径。

【P6】

最后一个其实上午很多专家也提到了，驱动ai编译器还有一个巨大的因素，就是整个深度学习的负载。第一个是自动微分，还有刚才讲的大模型带来的并行切分，包括一些量化，推理和常用的一些东西，这个是我对整个的ai编译器驱动力的一个理解。

【P7】

然后对整个ai编译器的范畴，ai编译器是一个领域特定的一个多层的编译器，主要是分三层，分别是图编译器，算子编译器和codegen。这里面我觉得大家有很多不同的定义，有的人把前端的转换其实没放在编译器的范畴里，模型从Python拿过来以后，怎么把它转到图层的IR里面，很多人就把它当做一个前端，不算在编译器的范畴里。我实际上在做的过程中发现这个转换其实是非常难的，有可能比后端还复杂，因为 Python太灵活了。所以我把前端的转换也放到了编译器里面，包括一些通用的优化，像nn相关的优化，这个是图编译器里面做的，然后算子编译器主要做循环优化、访存优化、内存Layout、并行优化。最后就是codegen。

当然大家也比较关心编译器里面IR的定义。现在大家普遍看到的比如图IR就有很多种，包括最早的像Tensorflow，我们可以认为它是一个Graph的方案，就是一个非常朴素的图的定义，后来像TorchScript做的是ssa的一种风格，然后像我们的mindspore是一个图的函数，这个主要就是为了更方便的做自动微分，Tensor里面用的比较多的是Halide IR。大家现在比较关心的是说我们有没有一种统一的IR的基础设施像MLIR，这个是我对整个ai编译器范畴的一个理解。

【P8】

最后我介绍一下对ai编译器未来的趋势和挑战的一些思考。整个ai编译器的驱动力无非就是三块，一个是应用的变化，应用会怎么样的发展，第二个就是开发者的体验怎么提升，第三个就是硬件的变化，最后就是一个开放的问题。总结下来，我觉得大概有4个大的课题。

在应用这一块，其实上午大家都提了大模型这块，我就不赘述了，但是这里面大家可能关注的点是不一样的，我关注的是从框架来说，怎么样能够把这种模型的切分做到自动化，完全的自动化。因为我原来是做数据库背景的，我做一个query从来不会要用户切来切去，我肯定是通过cost model，然后拿到一个plan就直接就执行了，但是我发现现在深度学习做的大模型太痛苦了，这是一个点。所以说我认为超大模型自动并行，不管它能不能做到真正的这种并行，还是做到中间的某一个程度，是未来非常大的挑战。

第二个挑战就是在深度学习里面CV跟NLP的模型，其实结构我觉得相对还是比较固定的。虽然早上林总也讲了有那么多的变化，但是这个变化变来变去就是transformer和CNN那几种。但是未来我觉得很大的一个变化在于，我们要跟科学计算进行结合，就是说你在一个模型里面，除了做深度学习的模型以外，你可能还要做一些科学计算的负载，这个是我们目前在做很多项目时候碰到的。在科学计算里面有什么特点？总结起来就三个，第一个叫高维，第二个叫高阶，第三个叫不规则。高维就是说数据的维度特别高，这里面一定要做大量的稀疏。第二个就是高阶，我们碰到了好多二阶甚至三阶微分的场景，这个在我原来的深度学习里面遇到的比较少，其实对我们框架的挑战是非常大的。第三个是不规则，深度学习有一个非常好的抽象Tensor，所有的操作都是在Tensor\*Tensor，Tensor+Tensor的范畴里，但是在hpc里面，它不是这种范式，它其实是Tensor与index，一旦有index，我们现在所有的优化其实是不work的。上面就是我觉得未来对我们编译器非常大的三个挑战。

第二块就是开发者体验，其实我们现在的框架偏向于一种方式，叫动静统一的执行方式，实际上比较简单，就是说我要静态图的时候就在加个@修饰符。这种方式我个人觉得它不一定是开发者最乐意接受的，因为没有一个开发者会知道我加了修饰符，这段代码能不能加速，究竟加速的怎么样，所以最好的方式其实还是不需要加修饰符，这是一个。第二个就是许多芯片，其实它最好的执行方式就是把整个图扔到芯片里面去，那静态图怎么实现。我觉得有三个趋势可以考虑，第一个就是我们能不能做一个跟动态图基本一致，比如说它的大量的语法，都能够翻译到我们的静态图上。这里比较难的就是控制流怎么解决。控制流怎么样翻译到静态，然后动态类型怎么解决，包括还有很多Python复杂的数据类型。第二个就是在动态图的模式上能不能做一个自动的JIT，不需要你去加这些修饰符，当然现在有些探索像LazyTensor，在底下自己来决定哪些热点的代码进行加速，但这个很难，目前我们实验的结果就是编译的时间比执行时间还长，因为很难系统的知道哪些代码是能够做到自动的JIT的，但这是一个比较大的方向。第三个方向其实大家上面也提到了，能不能做一个比Python更好的前端，但这是一个非常开放的问题，可能需要一个大厂去搞这样的一个生态。

第三块就是硬件dsa这一块，其实总的来说跟我上午说的目标是一样的，我们能不能做一个图灵完备的，然后性能还可以的这样的一个编译器，就是你提供一些原子的算子，然后用户可以在Python进行组合，然后编译器还能确保这些用户自定义的算子能够生成，而且性能还不错。如果这个目标能达到，我们就彻底的解放了开发者，但这个事情很难，这里面比如说像auto-schedule，auto-tiling这些事情都非常难，而且你们可以看到这里面一个非常通用的挑战就是cost model，无论是在自动的图切分，包括底下的auto-schedule，auto-tiling里面，都需要一个非常好的cost model，我觉得现在也还看不到有非常好的解决方案。

然后最后一个开放的话题就是这种MLIR ai编译器基础设施的必要性。我们公司也做了好多探索，确实也发现很多问题，比如说MLIR的编译器编译的时长要比普通的编译器慢个10倍左右。这一块是一个开放可以讨论的地方，还有其他的问题，大概内容就这么多，谢谢。