讨论部分 上午

（47：55）

张荣开：

大家好，我是鑫服计算的张荣开，是鑫服计算的联合创始人之一，主要负责软件还有芯片架构这个方面的工作，今天也很荣幸来参加这样一个编译的论坛，谢谢大家。

编译的问题，就像各位专家所说的，很多都是传统体系结构里面的问题，只是现在大家又找到一个新的ai计算的项目，又把它翻出来了，这是最大的一个感受。但是怎么样做取舍，然后随着时间的推移，什么样的一种架构能够在这种市场竞争中去存活下来，更有生命力，能够构建起一个软件的生态，这是我觉得大家更值得探讨的一个问题。

从我们实际跟一些客户交流的这种反馈来看，很多时候我感觉我们的硬件架构设计其实是走入了一个误区，就是去拼命的拼这种极致的性能。但是其实在实际的应用中没有什么意义，相反我觉得如何从芯片架构角度让软件变得更友好，更易用，这个才会让你的产品在实际的客户中更有生命力，从商业上成功的这种可能性会更大。这是我看到的一些实际的问题供大家各位参考，好，谢谢大家。

（50:02）

刘燕：

大家好，我是来自燧原科技的刘燕，我负责芯片的架构设计，然后那边的顾伟是我同事，是软件层架构师，跟大家比较熟悉一点。今天我是带几个问题过来的，今天前面几位像林老师还有尹老师的发言非常有启发，其实我们在平时工作时，在做架构设计的时候，一个核心的问题就是我们怎么样对存储优化。其实现在换到ai这个系统中，本身算力的提升是非常快速的，从晶体管的发展到摩尔定律的延伸，这块是非常快速的，但是存储墙的问题以及存储优化问题依然存在。最后到现在看到的大模型，刚才也提到了大模型的问题，这样的话最终会回到我们怎么样去feed计算引擎，怎么样去高效的提升数据可复用性、可编程性以及怎么样可控。其实大家不太喜欢可控数据这件事，因为传统来自于CPU编译器的体系架构，像刚刚唐博士提到 Facebook那边，认为对软件编程里面最优的,像cache这样，可能很多的programmer不希望去管理memory，希望能够通过common的统一编程方式来使得我们只关注运算和运算的负载，以及希望能够自动的通过或者是编辑器，或者说硬件上的一些体系结构的一些加速支持，能够把这些东西全部隐藏掉，提供一个对我们用户更有优势的一个结构。其实这块也是我们现在看到的一个问题，然后也想把这个问题带到会场，看看各位专家有没有什么好的解决方案。

林达华：

这里面的话，我觉得任何复杂系统的设计肯定都有一个非常关键的词叫trade off，我们得舍弃一些东西，才能够获得一些更有价值的收益。在商汤的话，其实我见过很多很复杂，小到写算子，大到现在在临港建的几十亿的集群，每一个过程都要去做trade off。那么对于AI来说的话，其实有两个点，第一个点就是ai的算法演化非常快，我们回头去看两年前AI都是用什么方法，到现在用什么方法。当然我也没办法预测两年之后会不会有很不一样的新算子会出来，所以现在有一个fundamental gap，就是AI算法演进的速度和体系结构的演进速度和周期完成不在同一个数量级上。一个芯片流片了到最后能够投入到实际应用，再经过了可能一年的时间写算子去优化，就是三年之后的事了，那三年之后究竟会有什么样的算子来代表我们主流的模型，这个实际上是没有办法去很好判断的。

这里面其实很重要的建议就是芯片也好，系统也好，它要有一个相对简单一点的设计，这样使得我们的软件栈能够更容易去跟上它。事实上像我刚才所说的，计算所占的时间在整个AI开发流程里面也就是2~3成，在这2~3成里面扣10%的性能提升价值并不是有那么大。真正我们希望看到的是一个新的芯片出来，它能够迅速的形成生产力，这个过程中需要相对简化，比如这个过程中简化了它的内存墙，简化了它的整个architecture，这可能损失了10%的性能，但是整个软件栈的开发可能因此缩短了50%的时间，能够更早让芯片发挥它的生产力，我觉得这个可能是更为重要的一个事情。

其实我并不建议内存墙做4级5级6级，这样也许还能再榨出5%的性能出来，但是我们做框架的很多同事非常痛苦，到最后等他做完了，这个算法就变了。

蒋力:

林老师，我紧接着一个观点，你是不是反对像MLPerf打榜的这种行为。

林达华：

我是看到了这种打榜行为，但我觉得这里面是这样的，我们商汤也是靠打榜起家，我们最早的话打了ImageNet，在2014年的时候说performance比Google差多少。首先，大家关注的所有打榜里的模式，最后都没有一个在生产环境用起来，最大的意义坦率的说就是有PR，拿投资。但在这个过程中，我们其实真正有了一个东西去锻造我们的基础的编译栈，实际上打榜本身的成果未必是最重要的，在这个过程中带动什么事情，比如说这个基础上，我们整个软件栈在哪个层面，它能够有一些革新，有一些创新。一般来说打榜的模型做的特别大，对各方面的要求特别有挑战，这个过程中的话它会带动一些其他东西的进步，这里面其实也是它的价值。

崔慧敏：

我觉得其实刚才几位老师也都讲了，存储优化确实是一个现在特别重要的issue，我们之前就针对好几款AI芯片，做了相应的memory的上面的优化，当然我们也会把它抽象一下，这个东西其实我们来重新看的话，可能和好多年前比如说随机分配差不多的思想，当然你不能直接搬过来用，但可以extend。当然之前大家也做过像 STM的管理什么的，用graph coloring是可以搞定的，但是问题在于不同的架构会有不同的设计，比如说SRAM的组织方式，它的大小，它bank上面的约束。所以我们觉得有很好的一种思路，当然每个芯片有自己的工具链，不太可能像以前弄一个LLVM，大家都用你的backend，每个公司有自己的backend的。我们现在在想一个新的proposal，就是在做 optimization as a service，比如说你需要做memory了，你可以把你的架构输入给一个模块，一个service，然后这个service就把分配以后的结果给你。Anyway，关于memory的优化确实是一个特别有意思的事情，包括刚才像唐博讲到的dataflow的定制，routine的定制，现在其实好多人都在做。我们也觉得这个东西应该是一个抽象的，因为它还是有很多共通的技术在里面，当时LLVM其实把很多东西抽象出来了，包括传统的CAC,backend是每个芯片有一个，但是大部分的算法其实是公用的，但是现在我们的AI里面基本上就是每家一个，我们觉得尤其是像memory，这一块其实是比较容易抽象出来的，它的层次化的架构，然后它互相之间的互联是什么样子的，这个东西还是有机会抽象出来做一个比较通用的模块或者service。这样一些东西还是可以抽象的，所以我觉得可能有这样一个topic也是个好的事情。

王敏捷：

先自我介绍一下，我叫王敏捷，现在在亚马逊上海人工智能研究院，做了一个系统叫DGL（Deep Graph Library）,专门做图神经网络的框架。因为我现在做软件上层，和很多的researcher打交道，然后去看他们在开发过程当中遇到一些问题。

比如说我经常发现的一个鸡和蛋的问题，因为图神经网络现在还非常的前期，所以很多的情况下，我发现我们schedule到大图上，会不知道是算法的问题，还是是硬件的问题或者是系统的问题，也就是说究竟现在能不能schedule到底是算法制约了还是系统制约，这些我们不是很清楚。我就是在思考这个问题该怎么解决，这是我的第一个问题。

第二个问题可能更加的controversial一点,就是说我们是不是需要一个新的language。目前来看，我们现在主流的编程框架，是Python或者说一个比较好用的 program language，加上tensor这样一个抽象作为一个编程语言，这个模式是不是一个正确的道路？我们未来是不是应该有新的方向能够把一些硬件的opportunity expose给user，这是我的两个问题。

唐杉：

第一个问题的意思是说你希望知道问题在哪，对吧？

王敏捷:

我想知道的解决方案是什么？比如说我现在有一个research，遇到一个比较新的deep learning问题的时候，我有的时候感觉是系统有制约，有的时候感觉到算法有制约，然后在这种情况下，应该往哪个方向去走，因为我可以出力量去做系统，我把这个系统做的比较很大，然后出来效果会很好，或者也有可能算法有问题。这个可能和编译相关一点，如果编译做的很好的话，就可以帮助一个idea很快的实现出来，但是目前我至少没看到。

唐杉：

我不知道我们一些现在的工具，不管是从profiling的角度，不管是编译能给你一些case的角度是不是能解决，但是你的问题也确实可以讨论一下。然后你第二个说法我其实是有点疑问的，现在是硬件可以给你暴露很多东西，有的硬件直接所有都暴露给你，data interfence全暴露给你，你memory怎么用，control怎么用，你自己全去编，但所有编程的人都愿意这么用吗？这不是给编程人增加太大的难度吗？

我也发现有些人可能想控制硬件，有些上层，或者说做算法的，或者说做编程的，其实挺想控制硬件的，但从我们的角度来讲，做硬件我们还是希望给上层提供好用一点的抽象。我不知道这个到时候怎么去解决，或者说你想控制是因为下层给你的东西不好用，还是一个什么情况？还是大家碰在一个什么地方比较好？我给你最想控制的东西，其他的我尽量帮你去抽象，我觉得这样子效率高一点，我不知道这个边界在哪，其实我们之前也讨论很久底层给上层到底暴露什么，暴露到哪个层次，这是我的感觉。

王敏捷：

我举两个例子。一个例子就是说我们在做算法的时候，有的时候会去开发一些新的算子，这些算子，其实也不能叫算子，我可以叫做Micro program，比如说我设计一个transformer，我想要把transformer中的attention结构给换掉，我其实希望做很高效，它不是一个普通的矩阵运算。还有就是是图神经网络，就是说它为什么是个算子，是因为粒度非常细，我要表达的运算逻辑非常细，我希望运算逻辑直接lower到kernel上去，我在program level我没有办法表达这件事情，现在至少给我的这些program都是非常粗粒度的，我没有看到现在有tool来给我们解决问题。第二个说我还想提到一个例子就是Alex，他其实是个非常好的hacker，当时我在Google实习的时候有幸跟他共事了一段时间，当时google的人在用TensorFlow，然后我问你用TensorFlow吗？他说我不用，我用CUDA comet。他很自豪地说我用自己编的东西，我说你为什么要用自己编东西？他说我可以去control everything，我可以写自己的东西，我可以非常好的去hacking我想要做的新的东西出来。我现在看到越来越多的，比如说在现在pytorch这种框架里面，它能够暴露出可定制化算子，但是能不能更进一步，或者说应该提出怎样的抽象给developer，来让我们能够很方便的去实现我们想要的idea。

唐杉：

我大概明白了，就是说其实developer的需求有时候不一样。你的需求或者你的创新点可能不在性能这个地方，所以你希望直接控制性能，就是说你对别人给你提供的性能可能还是不太满意，对吧？

蒋力：

其实你刚才作为developer的观点,就是像林老师说的，搞研究的develop和搞生产的develop对于硬件暴露的事情观点是不一样的，那我们内部的大论战就放在下午。

说话人（1:06：22）：

我补充一下我们在这一块现在正在尝试的做法。我们发现一个很复杂的神经网络编译器，从产品设计的角度来讲，它其实是存在不同方面的需求的，最重要的就是易用性，极致的可编程性跟优化粒度之间的怎么样能够取得一个很好的平衡。

对于一般的用户来讲，其实我是很希望能够非常易用，最好整个网络编译的过程全部无感，几乎不需要去介入，这个时候我可以接受Performance的比较一般的沟通，但是当你度过了这样一个新手期之后，进入到高级的用户阶段，我希望一个模型能够在你一个给定的硬件上把性能调到最好，这个时候其实又希望神经网络编译器能够去开很多口子，把很多硬件的细节给暴露出来，能够让高级的用户去把控很多东西，从而把performance调到最好。

这是我看到的两类不同的需求，那么怎么样去在神经网络编译器设计过程中，去很好的满足这两方面看似矛盾这个需求，其实是一个比较trick的事情。

薛继龙：

我先介绍一下，我是微软亚洲研究院的薛继龙，我也是负责做ai编译的，我先回答一下敏捷的一个问题，然后我再问一下嘉宾老师一个问题。事实上我觉得敏捷是观察到了一个非常有意思的需求，并且我们也观察到这个需求是一个很大的空间，就是市面上编译器在通用性和性能上面有一个gap的。

举个简单例子，像CUDA为代表算子库是非常高效的，但是它的灵活性会受到一些制约，像 TVM这种compiler比较灵活，基本上面写什么代码，它都能翻译到最下面kernel，但是它的编译时间比较长，那么我们现在正在做一个工作，在中间会做一个比较好的trade off，我现在不细说了，如果下午有时间我可以稍微简短的分享一下。接下来我想问一下三位老师一个问题，我们现在也观察到另外一个需求，就是神经网络里面会有一个很强的sparsity的现象，或者说pattern。我们不知道这个会不会成为一个非常common的pattern，因为人脑的结构这样的。那么首先我想问一下林老师你们在做工业的实践上面，对sparsity的应用或者落地有没有一些挑战，或者你认为以后有机会有多大？然后同时想问一下尹老师关于在硬件上面，如果想支持这样的pattern的话，您觉得有哪些挑战，或者是不是一个将来AI芯片里面比较重要的技术模块，然后最后问一下崔老师关于在编译优化上面，您觉得有哪些机会或者是挑战。

林达华：

首先回应算法方面的问题，就是 sparsity这个事儿。这个by nature的话，其实在所有的模型包括 cv的模型,在NLP可能更普遍，sparsity pattern是普遍存在的，但是对 pattern的充分利用，是不是高优先级的事情，我觉得也是另外一个story。现有的看上去的话，我非常赞同刚才尹老师的一个意见，其实现在很多的AI的应用，它的performance是访存这边bound的，在计算上面充分利用sparsity带来收益有多大，目前来说还没有特别的清楚。我们内部也有在这方面的探索，可能后面可以去尝试一下，把sparsity的leverage做得更深度一些，但这个事情在我们内部优先级并不高。因为这里面的诉求并不是特别的强烈，就是说这里面是普遍存在一个pattern，但它的利用是不是一个最痛点的问题，我觉得可能还没有到那么痛的一个程度。

然后又回应一下敏捷那边的问题，他提到就是说开放一个接口，能不能做很深的specialized的优化，在pytorch也能做，我们直接写到汇编层面，然后把算子拿出来去用，实际上中间的是非常packable的，这个事情是可以做的。然后刚才你提到了就是说要不要自己写一套语言。语言这个事情的话，我在回到香港进入商汤之前，其实在MIT毕业之后有一段时间做了一个语言叫Julia，这里面的话我常常。参与创建类的生态。（1:11：42）这个语言从我毕业那会开始做了，但是到现在七八年的这么一个时间，到现在还是一个比较小众的语言。我们找一个真正懂编译的同学或者说一个小的team，做一套语言出来并不难，这些东西都写在教科书上，这个事情并不难，难的是你这个语言究竟别人为什么要用。Python绝对不是一个最优秀的语言，很多东西表达不出来，比如说很多这种类型的信息，这种编译的语言，比如说Julia，它能够把每一个类，什么类型的，什么形状的全都写到静态上面去，这样编译的时候就很好用了，你知道它大小数据类型，编译的时候就可以这样做了。像Python的话啥都没有，实际上中间会做很多这种type nek把信息给提取出来，然后去指导编译，但Python为什么还是大家去用，这个原理跟pytorch为什么能在深度学习里发展起来是一样的，它特别容易用。编程语言有个很重要的一个点，它一定是interactive的，我们写程序，有人会以为是大家写了一大段，好几千行的编译，然后就出来一个东西，研究员不是这样写程序的，实际上是写两行就调一下，然后不对了，又改一改，所以这个时候这种脚本式的语言，它是会更受欢迎的。这里面反映了整个领域的trade off，易用性在这里面占了一个非常重要的位置，这也是我刚才一直在我的分享里面去强调的事情。如果说语言你要去做，有两点我觉得是需要注意的。你真正要投入进去的是多长的时间，多大的资源，然后它的价值能不能justify investment。以前我在学校里面不会从这个角度想事情，但是现在天天跟公司去做proposal，都要考虑这种事情，投入多少东西，做多长时间，最后return就成了justify。

我觉得做这个也是一样的，如果纯粹是很有兴趣想做这个事，我觉得没关系，晚上下班了把它写出来，上传到open source的committee，但是如果作为一个在公司里面的formal的investment的话，你总是report到上面给钱的那个人，你为什么要去投入做这样的一个事情？所以我觉得更大的是这个点。如果是graph neural level这个事的话，我是觉得最终还是得有一个落地的地方。第一现有的工具和你落地的 desire的status是不是有比较大的gap？第二就是落地最终影响的受众的范围是不是足够大?这两个乘到一起去，那可能就是你最终的目标有多大。

尹首一：

从芯片角度看，我觉得可以毫不夸张的讲，过去5年里面所有学术界做出来的 AI芯片，能够有比较好的publication的，其实都是在搞sparsity。因为坦白讲，除了sparsity以外，在计算角度来看，其实没有太多新东西可以做。当然sparsity从底层看，它的贡献还是很大的，因为sparsity这里其实有两种，一种是weight model里边产生的，像pooling之类的，还有一种就是activation里边，因为有relu这些东西，它自然就产生了很多，而且activation里边的sparsity还是动态，你也不知道它在哪。所以这时候从做 research的角度来讲，这就是很好的一个事。它很dynamic，然后量又很大，由于软件上编译的时候完全不知道，你就可以在硬件层次上设计各种的电路结构或者各种方法，你去check它。你甚至都可以改底层的乘法器，让它在遇到这么大的sparsity的时候能够效率更高。

所以过去我们看到很多非常顶级的工作，其实都是在搞这个，如果要是统计一下的话，从底层计算角度来讲，性能提升还是挺明显的。但反过来讲实现sparsity的检测代价挺大的，因为实际上就是在一个tensor里边去检测随机的0，检测了以后还不能影响计算的吞吐。因为你检测本身是有个复杂的逻辑在那里，这个逻辑如果处理不好的话，check本身的延时就比计算的延时还高了，所以那就不值得了。这样最后的结果就是检测部分的面积会很大，所以其实从商业的产品看，好像目前就 A100是在model上面支持了sparsity，但是它的sparsity还是一个regular的，random的甚至包括activation里的sparsity好像大家现在也都没怎么考虑。从研究的角度看，这个东西我觉得还是很值得做的，因为从性能提升来讲，效果还是会很明显的，但是从产品角度来讲，去平衡一下面积功耗各个方面的，总的来讲，我觉得还是很值得在底层处理的。

崔慧敏：

我觉得这编译方面sparsity也是一个非常热的问题。我刚才想了一下，我觉得值得研究的有三个地方。第一个针对ai这种model，它可能有一些sparsity的特殊的格式，像以前提的CSR的这种格式，更多的是从比如说图或者是HPC这些数据来的，对于ai来说，它的数据特征有一些自己特殊的地方。我们之前有一个博士生做过一个工作，这是很典型的，图片过来它可能有些维度是稀疏的，但是比如channel它是不稀疏的，这种可能就有一些机会，你可以去设计一些新的数据格式来处理它。

另外一个就是往硬件上mapping的时候，像尹老师提到的，现在本身就有很多针对sparsity的硬件，你怎么把这个 mapping上去也会有一定的格式问题在里面。

另外一种现在做的更多的，可能还是说把sparse的应用mapping到像GPU这样，更多的是为dense来设计的这种上面，这种时候你怎么去把shared memory用好，尤其现在像vector，在vector对sparsity技术有一定的支持以后。今年MIT好像有一个paper，它也是有一些这种方面的target，当然可能不是特别的全。另外我们现在考虑的是sparse和precision的结合，硬件上本身有好多不同精度的数据类型，我们在sparse的时候是不是可以跟这些数据类型交到一块去做，我觉得本身这个东西蛮有意思的。

蒋力：

我要补充一下，我们组也做了不少sparse的工作，像尹老师说的学术界都喜欢做这个，也有很多顶会的文章，我们现在也是逐渐考虑往编译器做，主要原因是什么呢？这种paper大家还是基于仿真，像不管是MICRO的paper也好还是DAC，仿真都是不准的，哪怕你做的再严谨，参数调的再严谨，其实还是不准的，因为这还是软件方面。所以这是为什么我们开始去往编译器去考虑稀疏的问题，然后一旦开始做，刚才崔老师提到的像CSR这个问题，我们是首先要tackle的，包括怎么去把它transfer，因为底层是dense，你肯定得在前面把transfer做好。所以我们发现一旦开始做这个事情，工作量就急剧的上升，你基于TVM也好，还是基于mindspore也好，可能一篇文章要做一年多时间才能够有一个实验结果，所以我觉得还是比较看好这个方向，还有其他问题吗？

许欣然：

好，大家好，我是许欣然，是现在负责MegEngine开发的这一块的，然后我这块有两个问题。一个问题是我们最近做的比较多的国内的各种训练芯片的对接工作，然后这块我们就会有疑惑，目前对接界面一个比较合适的点在哪里？像金雪锋金老师这边的mindspore我相信应该是一个全方面的对接，像算子库语义层这样的各种level应该都是足够开放的。假设是一个比较通用的芯片，我们对接是应该在推理空间把一个图交给你就彻底不管了，还是说像XLA这种稍微细一点点的，还是说语言级别，还是说提供一个算子库，这个层面其实很多，但是我们对接的一直很痛苦。不管是XLA还是说算子库，问题都会比较多，毕竟使用者可能特别喜欢像在Python里开一个PDB，随时都可以打个断点把 conv改一改接着往后跑，这种事儿其实是非常困难的，所以想问问大家的看法，尤其是在语言这一块。比如说寒武纪BANG C，我相信应该没有太多人真的写过BANG C，包括我这边设计的要不要太复杂这件事，可能不会有太多人有意愿去学。我学了一个非常复杂精巧的芯片设计语言，最后我下份工作好像用不上，可能很多人就会不想去学，那这个生态就会是这种情况，这是第一个问题。

第二个问题是我们在实际业务中可能会问嘉宾老师的一个点，像一些长尾算子，还有一些日常传统用的CV算子，实际在模型中用的还是很多的，比如说像我们公司的一些检测模型，可能往往百分之四五十的时间是花在这些地方，真正的标准的网络结构部分其实大家优化的都很好了，而前后那几部分一方面是说工作很少，因为它不是很有意思，没什么可做的，另外一方面是我们尝试了各种现在的编译器，其实在这方面都没什么太多的投入。比如说现在有一些复杂的CV算子操作，怎么fusion起来这事，我们试了TVM，Hailde，还有taichi，都不太work，我想知道大家怎么看这件事，是不是似乎现在搞深度学习编译器的人在躲避这部分的工作，这是我的一个疑问啊。

林达华：

我们观察到长尾算子在真实的CV模型里面占的比重相当的大，如果能够很好的解决这个问题的话，发paper我觉得应该也是非常有机会的，但是要解决好。刚才那个图，我们有个同事是在商汤里面做编译器的，为什么说我们商汤可以建那么多做系统的，做计算的，做编译的，做框架的，做这种加速器的，旷视应该也是类似这种情况。回到刚才说的适配国产芯片的事情，其实商汤内部这也是一个非常重要的事，有可能过了11点的话我还得去另外一个厂，一堆国产芯片厂商在里面。这里面很痛苦的一点，就是我们对接任何一款芯片，都得从头做不一样的事情。这还谈不上是有一套标准的设备跟界面，每个芯片的做法都不一样，两个月变一次API，这个事情会非常充裕，代表你几个月干的事情，所以在实际的工作强度下适配是非常重要的。从我们的角度来说，适配某款芯片，首先判断的是这个公司要不要跟我合作，我认为真正成功的适配不是我们定一个方向，interface我交给你，后面的我不管，这样绝对做不成。我觉得真正要把东西适配好做出竞争力的话，肯定这两个团队需要真正有一个共同的common goal，就是把这个东西做出来做好。大家要建立信任，真正work as a team才能把这个事情做好。但你当work as a team的时候，你的interface在哪？我觉得这个并不是需要well define的事情，在摸索的过程中基于destination（1：27：00）就会形成一个interface，这个其实是更重要的。现在有很多国内做芯片的厂商也找到我们，我们首先第一个要判断的就是你是不是真正有诚意去合作，然后基于什么样一个场景去打磨这个事，最终真正把这个事情给做成了。说实话，实际上我们去跟芯片厂商谈，这个过程叫找对象，但真正谈好了要适配了那就结婚之后的事情。

张行程：（1:27：44）

先自我介绍一下，我是张行程，在商汤负责训练框架。我echo一下第二个问题，我这边在公司做各种模型支持的时候，也有类似的现象。拿大家都能够看得到的例子来说，我们开源了我们的检测框架MM Detection，现在影响力应该是相当大了，这个仓库其实已经优化过很多轮了，即便是在这样一个优化的很好的检测库，我们在做训练的时候，他的时间有30%是花在conv以外的地方，都是那些所谓后处理，小算子，这还是一个优化很好的情况。事实上，有更多的算法研究员在想自己的idea的时候，设计出来的这些自定义代码，它的时间会更长，可能百分之四十几以上，这是一个客观存在的现象。

我们还有一个观察，现在的芯片厂商都花大量的力气去堆tensor core，Tensor的算力一代比一代都恨不得多一0出来。刚才前面的几位嘉宾的分享，关于编译的架构分享，我看到硬件和编译的注意力也都在 tensor或者是conv的计算上。包括像NV发布的新一代a100，它比v100的性能可能翻了一两番，但我们在实际网络上它能表现出来的性能提升其实只有不到20%，这个现象其实是我们看到非常痛的地方。

也有一些工作，比如TVM做的代码生成，看起来好像能把这些长尾的算子能生成好去做，我们也有试TVM，包括我们自己也有开发类似TVM codegen的工作，但实际的表现就是这些工作到现在还不能够被算法研究员使用。就是说现在还不存在这样一个工具，能够让算法研究员自己就把30%的 cost降到实际算力所应该占有的比重，或者说稍微大一点。这个我觉得是需要讲出来的，在座其实有很多做编译的，做架构的专家，我觉得大家要重视这个问题，接下来很可能我们的训练要有进一步的性能提升，其实不是在 tensor core的提升上，是要解决后面的长尾，怎么去把它给优化掉。

还有一个点，我们其实做过关于小算子编译的工作，在一些有限的case上，我们的一套工具可以做到把一个用户原来写的Python代码的加速10倍或者20倍，这个实现是有可能的。其实我觉得如果大家把注意力放到这边来的话，应该是能够做出很多很好成果的。

金雪锋：

框架和芯片的边界问题，我谈谈我的理解。其实跟一个产业的发展是有关，任何一个新的产业，它肯定会经过一个垂直整合然后到水平分层，我觉得ai的产业还处于一个垂直整合的阶段，它其实还很难分层出来。可能过了若干年以后，整个产业成熟了以后，它的结构才能慢慢的稳定下来。目前看的话就两类接口，一类算是个接口吧，这是CUDA原来给我们的接口。第二个是在大量的dsa芯片出现以后，因为每个dsa芯片它的目标肯定是要超越 GPU的性能，所以他不得不做大量的tensor core包括大量的图层的封装，但是我觉得未来这两层接口应该都会在。所以说另外一个维度，我也觉得这个是框架的一个新机会，因为最早的时候TensorFlow是静态图，然后pytorch搞了一个动态图，但是未来大量的dsa芯片的架构出现以后，纯的静态图其实在性能上又发挥不出dsa架构的一些效能，这个也是为什么mindspore要去做的原因之一。因为芯片用不起来，你得自己搞一套框架去把它发挥出来，我觉得未来可能整个框架的架构是非常灵活的，既能对接我们企业底下的像CUDA的这种算子的接口，也能去对接NPU的接口，这是第一个问题。

第二个是刚才提到的那种所谓的长尾算子，其实跟前面敏捷提的问题有点相似。我们整个AI编译器面临一个问题，就是图灵完备的问题，我们怎么做一个图灵完备的且性能还是可以的一个编译器，实际上我们都没有做到。整个ai编译器最颗粒就是一些kernel，上面我们的应用就是一些model，怎么样有一个基础的原子的kernel，然后model通过编译器都能自动生成。如果我们做到这一点，我觉得整个算法工程师的效率就高了，他如果需要一个新的算子，他利用这种原子组合一下就行了。但确实这个挑战非常大，我觉得下午我们可以好好讨论一下。

张洪斌：

大家好，我是中科院软件所PLCG实验室的张洪斌。我平时是做MLIR这块的工作的，大家刚才提到的比如说DSA或者DSL，我觉得MLIR可以帮助大家把中间打开。我这边有一个问题，因为从我们做端到端的一些编译的任务来看，需要很多工具链来协同配合，工具链要很长，要有翻译的过程，先是lowering，然后再翻译到LLVM之类的。当我们在debug的时候，我们可能很难定位到具体是哪条工具链出了问题，比如说我推理的结果不是很理想，我应该从哪里开始debug，是我平时比较头疼的一个问题，然后也想请教一下各位老师有没有好的方法？

有时候推理结果不是很理想或者分类不对，不知道具体比如说是训练的参数问题，还是具体编译的过程中哪个算子出了问题，就很难定位了。

林达华：

我在商汤的话，大量的时间就花在了所谓换了一套东西，换了一个什么东西，又差两个点，又要调几个月，当然我们形成了一些方法。

张行程：

这个问题确实很广泛存在，我这边讲一下我的经验。我主要是做训练，所以主要面临的问题是说，如果硬件或者是编译链某一个地方发生了变化之后，整个训练的结果发生了偏差，而且是最难的是发生了微小的偏差，这种情况该怎么处理？

首先训练这边的计算过程是良定义的，它的训练是由算子拼成网络，然后网络再做SGD或者是前向的迭代，然后再进行多轮迭代之后得到结果。其实每一步分阶段分得是非常清晰的，我们在做问题定位的时候，基本上就是顺着这个层次从底向上，先看它的每一个算子是不是行为跟我们的一个能复现的结果对标，比如说在NV上是不是一样。接着再看它的图像出来是不是一样的，再看它的梯度是不是一样，再然后看它一轮迭代之后是不是一样，这个过程中还得干掉一切的随机性，进行这样的比较之后，就可以一层一层的去发现它的问题，最后如果能够整个训下来的进度都能够对齐，应该就能齐了。这个过程的话，我们在实践中形成了一套工具体系，可以去在每一个层次去看它的计算的结果是不是能够跟前面对齐来做到快速的去发现问题。

我们没有专门花多少人去做这个事情，我们从开始做框架，就开始多多少少会遇到这样的问题，我们其实一直都在解这样的问题，最开始解决的很痛苦，第一个刚来的这个问题可能花了一个多月，才能够找到一个很简单的原因。后来我们逐渐的就把这里面的经验和过程工具化，到现在有一套工具，如果出了问题，我们就按流程一步一步的进行，大多数问题都可以做得很好。

王敏捷：

我就问一个问题，我刚看到你的这些验证方式都是数字化的，有没有比如说编译过程中形式化的方式出现？

说话人x：

我们没有做形式化，但是我们会在优化之后做很多的这方面的测试，一个优化完了之后的等价性的测试会写很多。另外一方面不知道问的主要是训练还是推理，推理我们这边处理的比较多。其实跟刚才张行程说的差不多，很重要的就是整个优化工具可能需要很多其他的工具去支撑它，比如说你的优化是不是能逐级展开，而不是一次就只有0和1。你可以一部分逐级展开，这样通过一系列的工具可以帮助你一层一层去定位这个问题。其实你不知道这个问题哪里会比较多，包括最后怎么才能从一个优化后的图再映射回原始的图，这些都是需要工具去支撑的。拿着一个完全优化后的和优化前的说对不上，那确实没什么办法。

刁岚松：

我是阿里巴巴pi团队的，我叫刁岚松。我这个问题可能是问在座的各位的，并不专指三位嘉宾。刚才提到做编译优化以后，它的正确性还在不在？一方面我们通过数值方法，在工程上是比较现实可行，大部分问题确实能够查出来。但是如果要100%保证正确性的话，可能还是需要一些形式化的验证工具，比如说符号计算这样的一些系统，能够保证说实实在在是对的，就像硬件设计里边形式化验证工具一样。我们现在有没有公司或者说高校有没有在研究这方面的话题。

崔慧敏：

我们做了一点点，但是我们现在也是比较苦恼的，也是想拿出来跟大家一块讨论。这个其实是我们做寒武纪的时候遇到的一个问题，它的芯片只有低精度的，但它支持的不光是DNN的model，它还有一些其他的机器学习的model，然后这样就会遇到比如在归约的时候精度就不够了，然后我们就做了一堆补偿，但是后来又遇到做另外一个算法的时候又不够了，然后又得补偿。后来我们就做了一个工具，你原来拿到的肯定是个float的程序，你转成比如说half或者int\_8以后，哪一个区域的代码会导致你精度影响比较大，这是一个问题。这个东西我们做了框架，也是多次跑啊跑，然后最后定位到一个东西，虽然说是自动的定位，但是问题是它要在simulator里面run好多次，也是个特别麻烦的事。然后我们也是想用symbolic evaluation来做，但这个东西比较难，因为symbolic evaluation本身不支持浮点，这确实是一个特别open的问题。其实精度那个实验我们做了一年了，但是现在我们也没想好怎么从symbolic的角度能把这个东西formulate好。上次也跟软件所另外一个老师讨论过这个问题,本身关于floating point的symbolic问题就是一个没解的问题，所以我们也不知道这个解在哪，我不知道别的团队还有没有在做的。

唐杉：

MLIR其实最后会转换成很多的层次的lowering，每次lowering它也会提供一些验证方法，但我好像没看到形式验证的方法。以前做芯片，做EDA，形式验证是特别重要，你基本不太可能通过仿真或者像软件的这种测试方法来测。我感觉以前编译这块应该是有形式验证的，因为我们做eda的时候，形式验证是继承编译软件的一些东西来的。我不知道这块现在是有什么新的问题？

崔慧敏：

大家问的比较多的是编译前后的代码是不是等价的。华为的冯青云老师一直在做这个，国外的邵东老师也一直在做这个，这个的问题在于它对你输入的语言有一定的限制，不是说你随便写成什么样子都能过的。对于编译器这种里面写的乱七八糟，就不太好处理。

欧洲有一个团队自己重写了编译器，这个编译器他们证明过了，但是那个里面有好多优化都没有开，你要想把，不说MLIR，像 LLVM能转过去就已经比较难了。

唐杉：

确实EDA里的形式验证要求是挺严格的，有写法的问题，还要有很多的点要对上。

金雪锋：

这个问题是硬件换了导致精度不一样了，还是编译器本身的问题?如果是因为硬件变了，我觉得确实形式验证解决不了这个问题，因为这是一个对比标杆的问题，硬件的精度对不对，并不是说你本身对或者不对，而是因为你跟GPU去比是不对的。对于深度学习来，本来就是概率的东西是，没有说绝对正确或者错误的说法。或者说你做了一个软件，你发现跟你原来用的软件精度不一样，你说谁对呢？这不好说，但是人家已经用了，你必须跟他的比，最后把你的软件调的和它一样，你就认为是对了。所以这一块我觉得好像不是形式化能搞定的问题，但是如果说你编译器做出来正确与否或者操作系统做出来正确与否，这个有很多工作都在做。

刁岚松：

无论是芯片还是编译器，我们有可能会改变计算的顺序，浮点的话，计算顺序稍微变了，比如说a加b加c与ab先加和BC先加，出来可能就不一样了。最后看到结果不一样，有个东西能给个判断，这个不仅仅是计算顺序带来的微小的偏差，而是实实在在编译已经出问题了，像这种东西也是有价值的。我们现在就看到一个数值，我们不好判断到底问题出在哪，如果是由于符号计算，它能够告诉我：你放心，这仅仅是计算顺序的问题。但如果说没有这样一个东西，仅仅是数值计算一样的，我就很难判断。

杨军：

我是杨军，以前是在阿里巴巴。可能有一些tricks的部分能解决一些问题，比如说我们说到浮点计算的话，它具备一定随机性，我在GPU里面可能是a加b加c，我换个新硬件会变成c加b加a，这样的话结果就会不一样的。这种情况一般来说是很难真的去通过比对精度去check的，当我们引入了像IP16的话会更复杂，那我们可以换个角度，就是我们的bug的来源在什么地方？

bug的来源一方面可能是精度的误差，另外是逻辑的bug。如果是逻辑bug的话，我们控制随机性以后，bug就能够稳定的复现出来。那么如果我们对data pattern作为一个fix，控制好输入，枚举不同的test vector，再多次去验证后稳定的在一个点出现，那么可能会是bug，如果在不同点出现的话，可能更多的是一个随机误差。但是这个不是很精细的形式化的方法，但这个方式可能在一定程度上解决岚松那个问题。

另外一点据我自己了解的话，可能在NV刚开始做tensor core时候，他们其实是拿FP64做一个gold(1:48:12),在里面做了很严格的check，也经过了很长的周期，来确保新硬件的精度是符合要求的，这些也是我对这些问题的思路。

说话人：（1：48:40）

我是来自天数智芯做软件库的，然后我回答一下NV是怎么的验证的。NV的调换顺序a加b加c和a加c加b，包括有些算子会调 atomic，比如原子加，有些算法它是分确定性和非确定性的，你设计atomic的时候，一些算法的顺序就会变。NV验证的时候，那些a b c可以由于浮点会有溢出的问题，那它去验证的时候他提出来的时候只有几位（1:49:18），那么他就能保证他加的时候，他那个位置精度是控制好，这样的话这些实验在一定程度上相当于一个形式实验。

我想提的问题就是说我现在做一些kernel的优化，可能对算子的时间上优化比较多，但是其实也有问题，最后kernel联合在一起之后，你跑的可能和单个结果不一样。有些kernel用的带宽比较多，有的kernel cache行为比较多，感觉有些不太可控，因为cache行为其实不好预测。还有一些就是kernel输出的layout，输出形式其实和很多输出形式是有等价性的，比如说数据的layout可能是NCHW，也可能是NHWC，这两个都可以输出，可能这一部分比较慢，下一部分就比较快，这种的话在上市场的框架是怎么处理的？

我的主要问题就是说上层框架怎样处理多个kernel各自不同的特点，像memory或是layout这方面的。

杨军：

这本质上是两个问题，第一个问题是说不同的算子有不同的版本，我应该选哪个版本来去做全局的优化。第二就是说当需要做跨算子fuse的话，我该怎么决定fuse是过于激进，还是说可以做的更接近一些？因为fusion确实不是越多越好，就像尹老师说的，fusion多的话是有代价的。

第一个很早以前NV已经有过类似的做法，比如tensorRT里做得elegant一些，它是把整个组合优化变成了一个热力代数的问题，变成了一个用solver来解决的东西，把它定义成一个类似dynamic programing的依赖链，每个链的每个节点的话提供一些option，可能是NCHW可能是NHWC,甚至可能加了vector（1:52:05）量化的layout转换。那么在整个链的讨论过程中，它通过一些组合优化的求解找到最后的路径，这是他的做法。TVM的话我记得去年发了一篇文章,他们是做了一个dynamic programing的做法，求一个近似解，也是类似的情况。如果我们把它抽象一下，其实这个问题本质上很像是ACM里面一个DP的大问题。

第二，你前一个关于fusion的问题，有些算子对memory用的很多，有些对寄存器用的很多，那么这本质上就是当我fuse完以后，带来收益的同时，kernel的资源消耗增加了，因为没法利用kernel的barrier来保证每个kernel是独立的，可以用所有资源。那么这里面有些不同的方式， 像FC里面他做了一个比较简单的规则，比如说该不该fuse，fuse是否有好的收益，还有一些工作做的可能更为漂亮一些，就是设计一个cost model，决定fuse完以后，我预估的收益会有多少，cost有多少，来决定怎么做一个探索。但这个问题可能比较吃力的点是说，可能在NV的v100之前的架构里面，收益的上界不是那么的高，因为fusion本身的benefit可能会有limitation，这个问题从v100往后变得更有趣，因为算力的增加跟访存launch的gap变得越来越显著，包括像NV新的软件栈，包括现在很多dsa的公司，都在考虑类似的问题，希望有更漂亮的一些query的工作。