中科院计算所 崔慧敏

【P1】

09:07  
大家好，非常荣幸。其实因为我们这个团队一直是在做传统编译器的，所以 AI刚出来的时候，我们最早跟寒武纪陈云霁老师我们一块聊，当时他们刚开始做AI的时候，其实大家就是说因为他们做芯片，然后当时就是说我们就有过一次讨论，就是说我们到底需不需要一层类似CUDA这样的一层语言，当时算子还相对蛮固定的，但是当时后来我们讨论的结论还是就像林老师提到，还是让大家不要受硬件约束，要能够想玩什么玩什么，所以我们觉得寒武纪也是需要这样一个一层比较general的语言的，所以后来我们就做了Bang语言，你可以看到不管怎么样，我们当时的思路都是从一层general的角度来出发，因为我们没有AI的背景，我们没有 AI的这些知识，所以我们都是比较平的去看这个代码的，但我们在这个过程中做的这几年过程中也会发现，我们用传统编辑的思路来优化，确实有一些特殊的机会，所以我今天想抛出来一个问题就是说我们到底传统编译和AI编译它们之间结合点在哪里？

【P2】  
所以这是我们团队之前的一直做过的事情，我们最早从五六十年代开始做大型编译器团队一直就存在，然后到从2000年以后，我们最早是给intel做ORC完全就是纯面向一个芯片架构的编译。后来开始我们所做龙芯，我们也开始面向龙芯做编译。然后又当时神威开始启动，我们又面向神威，就帮神威做编译。然后一直到后来现在开始给寒武纪做编译，所以一直是处在一个通用编译器的这样一个层次上。

【P3】  
然后我们这几年有一些思考，我们刚才几位老师都提到了，做AI的话一个就是算子内的，一个就是算子间的对吧？然后算子内的话，这个是我们当时做Bang语言的一个很很主要的一个诉求，就是把当时编译器的基本上的一些像model-driven compiler的analysis和optimization都加进来，然后这些常规的一些优化是可以做的，这像CUDA做的这些事情一样。

然后 auto tune的一些东西，这些传统compiler做auto-tuning的技术要加进来。然后像intelligent optimization是我们缺失的，然后但是现在寒武纪的人在往里做，但这一块因为我们的background比较弱，我们就没有加进来了。

但是后来我们做了很什么的一块，就是我们做了semantic的knowledge，就是做AI的人，其实AI从你们的角度，你们对算子是干什么的，然后他每一行代码干什么其实是非常清楚的，但是反映到compiler，从代码的层次来看，它其实不知道这个事情的，然后这样就miss掉一些东西，然后我们从算子的角度把这个知识再加进来，然后这样会有一些特殊的优化机会，所以这个是在自定义算子里面去做优化的时候我们的一些想法。

然后其实后来我们就发现做AI的人非常care做 operator fusion，就是因为林老师提到的kernel launch的代价实在太大了，然后尤其是现在有非常多的这种长尾算子出来以后，其实一个做可能比如是 operator fusion，然后另外一种就是像Rammer这样的，他可以把好多小的kenel同时的launch起来对吧？然后这样去做。

但是这个东西我们其实一年多前开始做这个问题的时候，我们就说为什么要专门去做operator的fusion，如果把我如果看到的不只是一个算子层面那个代码，然后我从图的层面看的话，我也不是只看到node的，我把图和代码结合到一块儿，我不就把原来大的程序返回出来了吗？我拿到大的程序，对传统的compiler来说就是非常经典的loop fusion对吧？但是在这里面当然有非常多的问题，你怎么把这个东西fuse到一块，然后你怎么把这个东西又reverse回去，把这些所有的往CUDA上丢的，我们拿CUDA来说哈，CUDA上丢的这些所有的隐式的循环，你再把它打开，然后你要把所有的memory给它搞好，其实是一个繁琐的过程。

【P4】

所以我们现在大概两年这两年的时间，我们build的一个就是能把host和kernel fuse到一块去做analysis的这样一个框架。这个框架其实当时我们publish出去的paper是去用OpenCL来做的，然后因为我们当时是其实因为关起门来说，寒武纪的Bang是很像CUDA，model是差不多的，然后OpenCL我们也用在了神威上，然后当然它support的平台非常多，当然OpenCL不是一个很好的推广的语言，但是按照我们做这个一些前期的验证是可以的。

就是说我们是把host代码和kernel代码拿到一块，然后把他们的AST层面合到一块，然后在这上面去一块去做lowering，lowering到LLVM IR上面去。然后这样带来的一个问题，你fuse到一块以后control flow analysis怎么做对吧？然后在我们的paper里面解决了。然后data flow的analysis怎么做对吧？我们也解决了，然后inter procedure的analysis怎么做我们也解决了，所以我们把把基础的base的analysis都解决了，然后现在我们在做在这上面在做怎么做fusion ？

【P5】

简单抛一个例子来看一下，我们这样子fusion和传统在operator层次来做fusion有什么区别？这个是 LSTM的例子，Rammer也是拿它来做一个很好的demonstration，然后你可以看到它的大量的模式，你如果把它循环展开以后，然后你去做传统的dependence analysis，你会发到发现它的循环的依赖是这种wavefront的模式，我们自然就可以用传统的 loop skewing这些东西来找到这样一个parallel的模式，然后这样的话我就知道我沿着wavefront的方式是可以去做并行的，对吧？这种情况下我可以去做fusion，我知道每个kernel都比较小，我从kernel的层次，我从每一个算子的层次，我知道他可以去做fusion，我可以把他们fuse到一块，fuse到一块的好处是什么？然后你可以做data reuse，这个其实是传统以前大家做两种fusion就是不会搞定的，那Rammer这种方式，它可以把多个kernel一块launch起来，但是它不会解决多个kernel之间data reuse的问题。

传统的operator fusion是在解决data reduce的问题，但是他不会考虑inter kernel的parallelism的问题。所以但这个东西因为赵捷老师也在，赵捷老师就更擅长了对吧？就这个东西从compiler的角度，它是天然能够解决的。因为它有很多的model，像polyhedral的，然后传统的LNO的model都是可以同时在兼顾parallelism和locality的，所以这就把原来大家做的一个比较面向AI做的一个model，然后其实从compiler的角度你只要model好，其实它是一个传统compiler可以解决很好的问题。

所以我们的角度是从，当然我觉得这样也有问题，因为你非常费事。很多AI model的信息其实从就是语义啊，然后analysis的角度啊， parallel的dependency啊，然后data race啊，这些东西其实都丢掉了，对我来说也是不好的，所以我们现在在写paper就遇到一个问题，就是说我们到底这个东西这个界限到底该怎么定，所以这也是我们一直 paper没有organize好，就觉得这个界限我们自己都还没有搞清楚。所以今天在座的各位专家在这也是希望能够我们在这个问题上帮我们往前走，进一步的一个回答。好，然后我的share就是这样子，谢谢大家。

蒋力13 16:56  
谢谢崔老师的分享，其实是个非常好的大的问题，编译器和AI编译器到底边界在哪里？好，我们紧接着到再下面一层，编译器在下面就是芯片架构了，我们这边走着两位非常厉害的芯片架构师，我们先有请清华大学的尹首一教授，他们 Thinker芯片系列也是名噪中外对吧？

所以我很早就跟尹老师交流的时候，他就给我吐槽过，当时用TVM做，因为基础去做编译的时候遇到了很多坑，反映了很多问题，所以今天我们就从芯片架构的角度来看看编译器到底趟过什么坑，或者我们对于对我们现在在场这些做编译的专家有什么样的一个一个建议或者诉求，很欢迎。