唐杉

【P1】

现在市面上有各式各样的架构,是一个架构的黄金时代，虽然新东西不多，大部分都是老东西，或者说可能全部都是之前有过的。这么多各式各样的架构或者架构上的选择，对编译器或者上层的框架，提出了很多的问题。

不一定是纯ai的，现在各式各样的加速芯片，或者大家叫的bsa或者是传统的架构的dsa的组合可能有很多种变化。从一个基本的 core的来讲，传统的Scalar，到CIMD、vector，不管是数据并行、指令并行的各种模式现在都有，然后加上tensor core这种三维的计算，multi-core从一个大core到1000多个小core，还有many-core和Chiplet。Chiplet之上就是multi-package，一个板子上有多个chip，再往上就是多个板卡，甚至是多个reg。这种不同的层次之间一个重要的区分，就是不同层次相应的communication的代价是不太一样的，比如说on chip的communication，chip外的 communication，再往上就是 package之间的communication，然后再往上就是board之间的。对于memory，以前大家用cache这些硬件来控制或者调度的memory，其实对compiler或者对上面编程者是比较友好，大家看不到它具体的调度。但现在把 memory system暴露给compiler，或者暴露给编程的人来用是很常见的。现在很多ai加速里面的SRAM直接是静态分布或者需要软件来分配的一个memory，这个问题复杂性又增大了。

现在的异构的组合也是各种各样，传统的CPU编程可能比较容易，但是现在CPU的extension也很多。除了传统CPU，现在res-5也在讲vector的extension，也在讲含有DSP的extension等等这些东西，这个可能还是耦合最紧的，相对来说简单一点。像SIMT+Tensor core和CPU+NPU的组合排列起来就非常多了，大家希望compiler能把它都用好，或者编程者都把他用好，这是大家的期待。

后面还有各种各样特殊的硬件架构，比如 CGRA，李老师他们做了很多年，现在也有一些关注度很高的stuff来做这种CGRA类的架构，未来可能还有 In-memory的computing，processing，还有会和传统的处理器结合在一起的一个异构的架构，这些问题的组合就会就会非常多。

还有一个问题就是比如说你拿了一个 IP化的加速器，其实它是有很多配置的，你是放几个core，每个SRAM配多少，memory hierarchy是几层，我们有时候会简化这个问题，可能memory hierarchy会比较简单，但实际上memory hierarchy复杂的芯片是非常多的。现在的问题在于normal people怎么去编程这个东西，特别聪明的人可能这种复杂度还能写出几个好的kernel，但是一般人就很困难，现在有各式各样新的这种芯片，这个现象是很普遍的。现在还有一些想法，干脆把所有硬件里去control的部分都拿掉，都给计算的部分，然后把 control的难度都给软件，当然我们不评价好坏，反正也有这个说法。

【P2】

这一页主要讲了什么硬件是编程友好的，什么硬件是编程不友好的。左边一列是编程友好的，右边不友好的，比如说 Memory，像cache这种可控制的肯定是编程友好的，SRAM这种你必须自己去分配的，这肯定不友好.包括通讯和处理器的核心这块。

【P3】

这是一个例子，这个wafer-scale的芯片是非常大的一个芯片，大家都很佩服作者的工程能力，但是其实从各个层面来讲，它是编程很不友好的一个芯片。它每个最小的tile都是图上这样的一个东西，整体来看非常规则，但每个tile的memory和control要自己控制，而且很重要的是每个tile之间有router，包含东南西北4个方向，而dataflow流程必须事先排好。这里面有这么多个tile，many-core的编程也很难，把一个workload mapping到上面，你要排得非常好，你不仅要把 SRAM用好，你不仅要把core用好，你还要把 router编程编好，然后你才能达到一定的效果。下面还有个paper讲了怎么去mapping一个科学计算的应用在上面，它可能能实现一个大概百分之四五十的这种效率提升，我觉得他能拿出来的肯定是已经很好了，但是这种编程我们可以想象一下，有这么多的核，然后把每一个核配合好，这个是非常难的。

【P4】

Cerebras面对这么多硬件的这种问题，他也给了答案，就是 MLIR加circuit这两个项目。对这个我们不评价，我也没有答案，我只是把这个问题摆出来。我作为硬件架构师我很关心什么呢？还是软硬件划分的问题。我们做硬件架构一般来说第一部分就是考虑软硬件划分的问题，就是谁在硬件做，谁在软件做，什么东西在硬件做合适，什么东西在软件做合适，可以得到整体的效率最高的。大家现在做很多架构选择，有很多trade off，但是我还是希望大家能讨论出一个相对来说好一点的划分。有些东西在硬件做可能很简单，那硬件可以增加一些cost来去做，可能就减少了很多编译或者program的难度，但是有的可能确实在硬件做的话代价很大，当然这个跟workload有很大关系。像前面那个例子，他有的可能追求的不是单个core运行的效率，而是一个很大的scale的这种运算，他把所有的model所有东西都放在芯片上，SRAM可以做的很大，反正能放下的，他就认为是适合他的运算，这也是domain-specific的特征，也许某种硬件架构就适合很窄的东西，我们就把它限定在很窄的范围内。

我很想知道硬件架构和编译器在某种workload上更合理的分工的一个点在什么地方？现在大家有的从通用到专用，现在专用开始到通用，一直在循环，这个问题不一定很快有答案，但讨论一下也是很有意思的，或者说很有价值的。