在CPU平台上CNN模型上的推理优化

卷积神经网络（CNN）的越来越受欢迎，CPU的广泛使用，提高在CPU上CNN模型的推理性能，可以给使用者带来可观的收益。当前的主要方法（MXNet和Intel OpenVINO）通常将模型视为图，并使用诸如Inter-MKL-DNN之类的高性能库来实现图形操作。尽管这些库在单个操作上实现了不错的性能，但由于本地操作都是预定义的，使得这种解决方案在图级优化上不是很灵活。然而在整体上缺乏端到端的优化，所以本论文提出了一个NeoCPU的方案，它是一个全栈的系统的优化方案。NeoCPU作为一个模板化的优化操作，它不依赖第三方库并且可以将操作级（operation-level）和图形级（graph-level）结合，实现更好的优化。

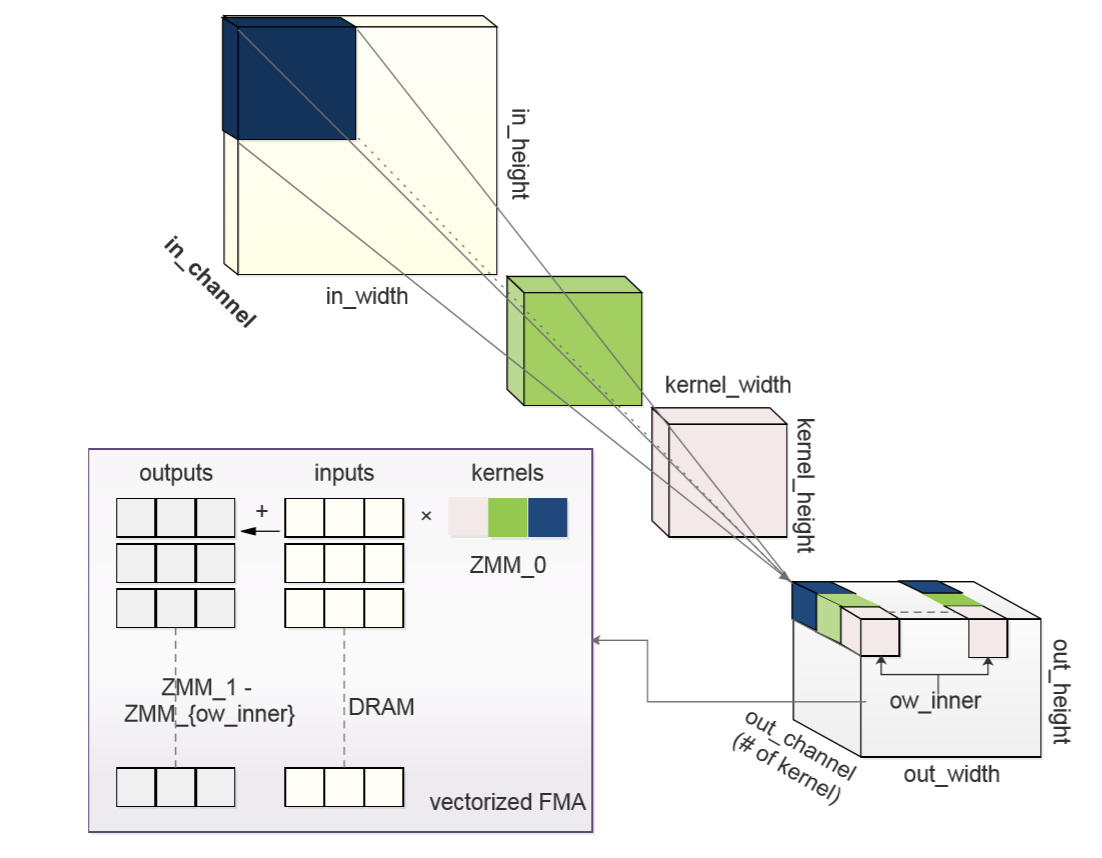
论文主要分为一下几个部分来介绍，前两节介绍了背景，第三节提出了优化思路和实现办法，第四节通过实验，对NeoCPU方案进行评估，以及最后一些的总结工作。

具体而言本文做的主要工作：

1. 提出了操作级和图像级的联合优化方案。
2. 构造了一个模板，来提高卷积的性能，可应用与多个CPU构架而无需依赖高性能的内核库。
3. 设计了一个全局方案以在CNN模型的不同操作中寻找最佳的布局组合，最大程度的减少布局转换带来的开销。

以下为作者的优化思路和方法，其基本思想是端到端的优化并寻求全局最优性能。按照操作优化（单线程优化和多线程优化）到布局转换消除再到优化方案检索步骤来实现端到端的优化方案。

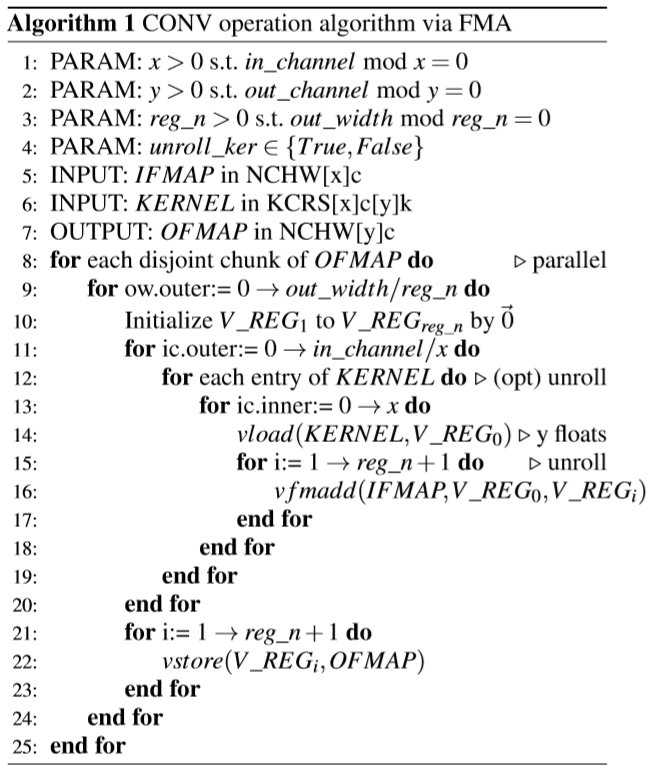
首先是操作优化。由于计算量大，卷积的优化对CNN模型的性能很重要，以往都是通过优化汇编代码来提高性能。本文作者展示使用最新的CPU功能（SIMI，FMA，并行化）来优化COVN的卷积运算。首先是单线程的优化。CONV是计算密集型的需多次遍历操作。CNN中2DCONV使用3D特征图（高度x宽度x通道）和多个卷积kernel（通常高度和宽度较小，但通道相同）进行卷积以输出另一个3D的张量（tensor）。计算如图1。每个kernel都沿高和宽在要素图上滑动进行逐元素乘积和累加以在输出要素图上生成相应的元素。这里使用NCHW来描述默认的数据布局，这意味着输入和输出是批处理大小为N的4维tensor，通道数为C，要素map高度为H，要素map宽度是W，N是数据最外部的数量。Kernel的布局是KCRS，K是输出信道，C是输入信道，R是kernel的高度，S是kernel的宽度。作者将特征图布局组织为NCHW[x]c,c表示通道C的分割子维，x表示子维度的分割大小（channels=sizeof（c）×sizeof（x),其中sizeof（c）=x）以获得最佳的内存访问模式（即获得更好的缓存局部性）。输出具有与输入相同的布局NCHW[y]c，卷积kernel布局为KCRS[x]c[y]k，具有分割大小的x的c和具有分割大小y的k分别是输入通道c输出通道k的子维度。



如图算法1总结了在单线程中对CONV的优化，本质是：

1. 友好内存布局性的维度排序
2. 寄存器块化以实现良好的矢量化指令利用率。

使用循环展开的策略，使块的大小（x，y）和寄存器可以配置，因此可根据不同的CPU体系结构和工作量调整计算逻辑。



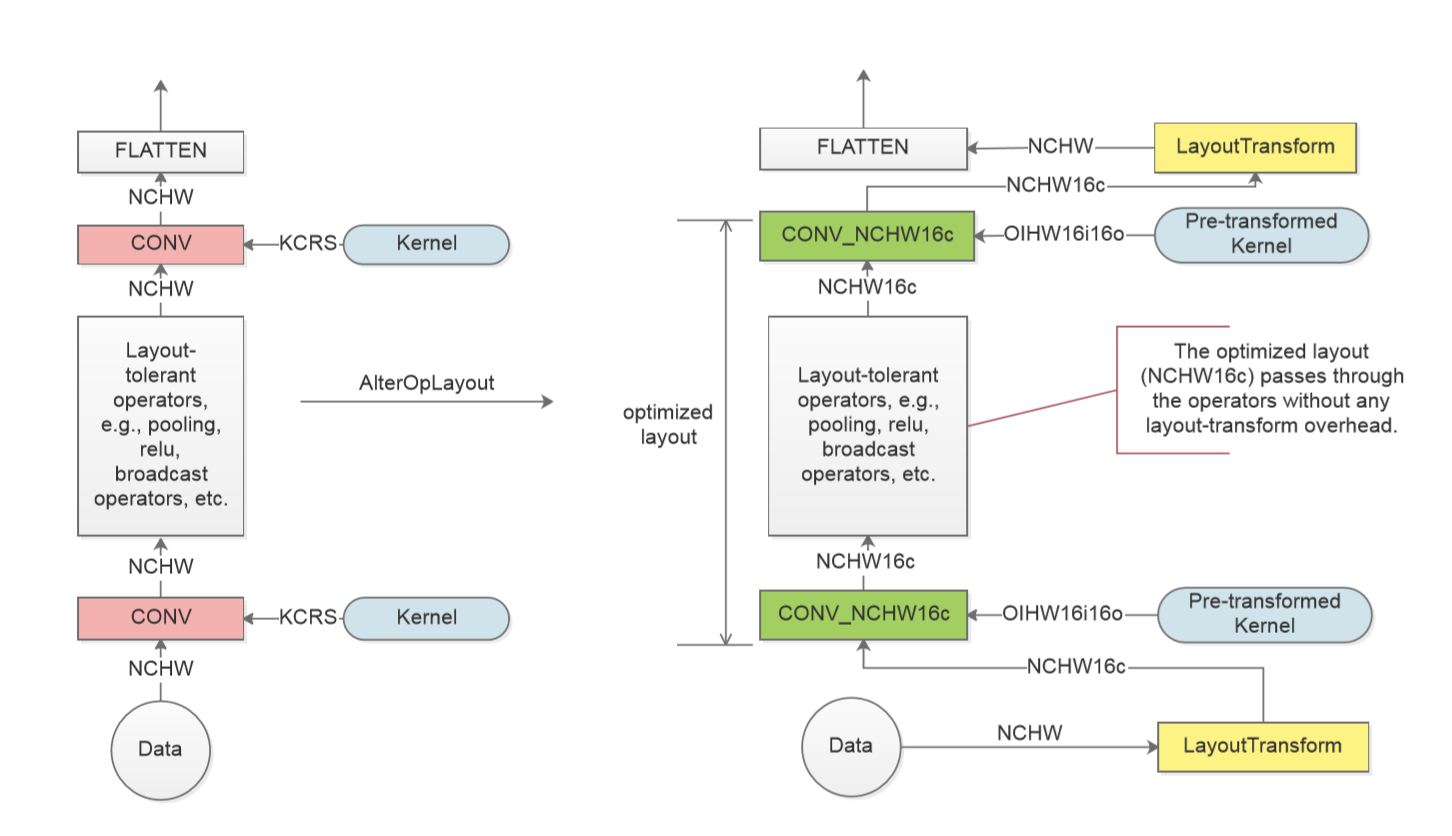
本文还提到的实现了一个自定义的线程池，处理并行化，采用有意机制防止资源的争用并减少线程启动的开销，其性能是优于openMP。

其次是布局转化的消除，其将优化范围从单个操作扩展到CNN模型的整个计算图。主要思想在图级别上提出通用的解决方案，已最大程度减少上述优化引入数据转换的开销。将CONV之外的布局转换作为独立的节点，仅在必要时插入它，消除了CONV操作中发生的转换。

为了确定是否需要进行数据布局转换，将相关操作分为三类：

1. 布局无关操作，可处理任意布局数据
2. 布局允许操作，可处理多种布局数据
3. 布局依赖操作，可处理特定布局数据

前两种操作无需进行布局转换，仅当布局依赖操作（数据从NCHW[x]c转换回NCHW）需要布局的转换。在实践中，首先遍历计算图以推断每个节点的布局，将CONV的布局从默认转换成NCHW[x]c以获得更好的性能。如图2，在左侧显示了具有默认数据布局的网络，粉色的每个CONV节点都要付出额外的开销，才能转换成高性能的布局以实现更好的性能，然后在转换成默认的布局；右图将LayoutTransform节点插入，该网络任具有NCHW输入输出，但CONV层之间的布局处于NCHW[x]c，并且模型的布局参数不变，因此可在编译期间进行预转换。

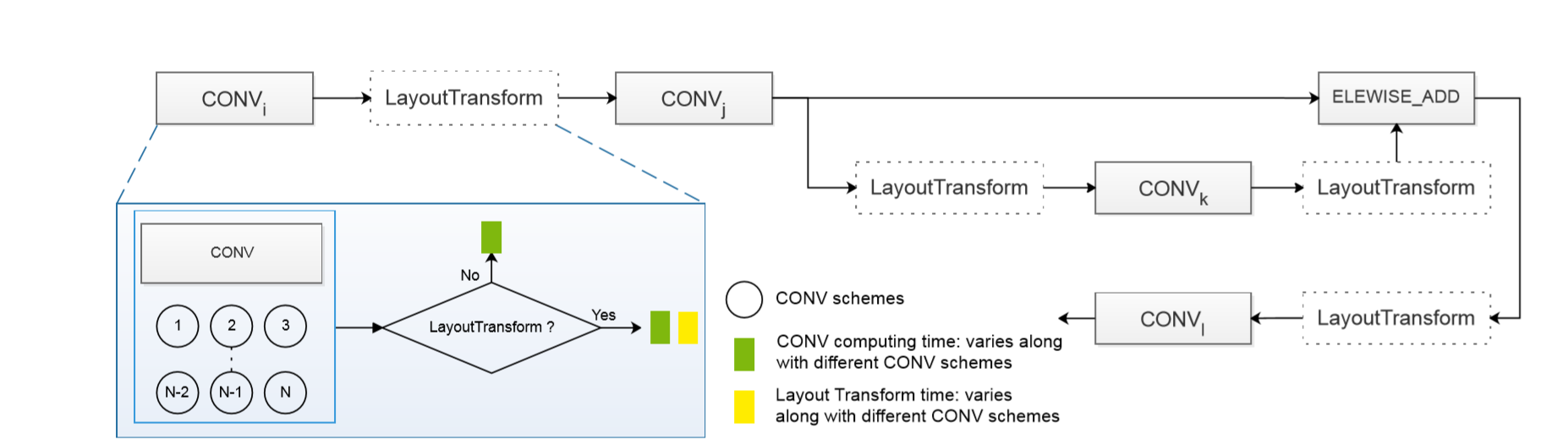


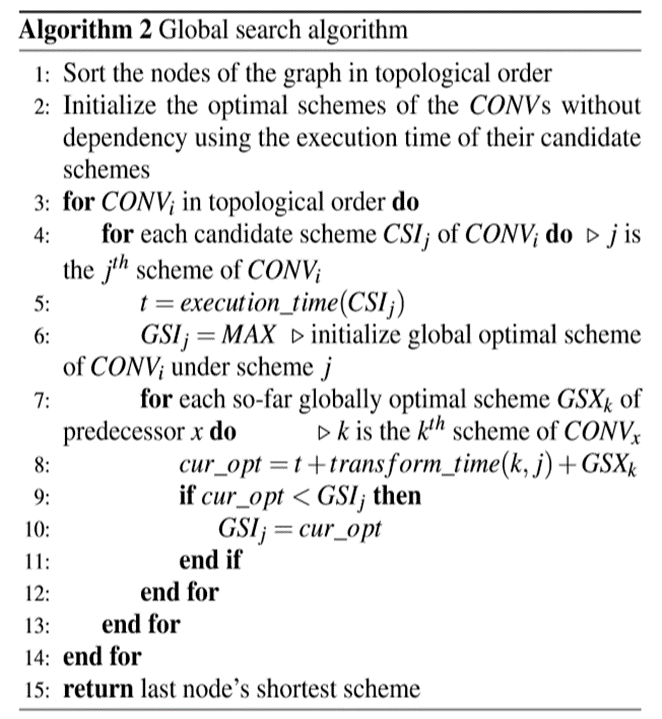
最后是优化方案检索。由于不同的CONV中的x之可能有更好的性能，另外输出宽度的分割因子（reg\_n）支队不同的指令集要相应的调整，所以文章使用了自动检索的方案进行提高性能。检索分为两个阶段，局部检索和全局检索。

要找到局部的最佳性能，使用元组（ic\_bn，oc\_bn，reg\_n，unroll\_ker）表示卷积方案。ic\_bn和oc\_bn表示输入和输出通道的分割因子（NCHW[x]c）中的x），与不同CPU的缓存有关;reg\_n是在内部循环中使用SIMD寄存器的数量；unroll\_ker是一个布尔值，决定是否展开涉及卷积kernel计算的for循环。具体步骤如下：

1. 定义ic\_bn和oc\_bn的候选列表，为了穷尽所有的可能因素都包括在内，例如通道数为64，则[32,16,8,4,2,1]候选。
2. 定义reg\_n的候选列表，[32,16,8,4,2,1]候选。
3. 定义unroll\_ker的候选列表，[true,flase]候选。
4. 遍历定义的空间，记录所有组合的执行空间，多次取平均值减少误差，最终生成排序表。

局部检索到最佳性能后，文章将优化检索扩展到整个计算图，即全局检索。目的是使每个CONV自由的选择分割因子x（ic\_bn和oc\_bn）。如图3，图中每个CONV都有许多不同的ic\_bn和oc\_bn对候选，在局部检索中可获得最短的执行时间，可以根据全局检索决策调用或者不调用LayoutTransform，如果调用，则要支付黄色表示的数据开销。



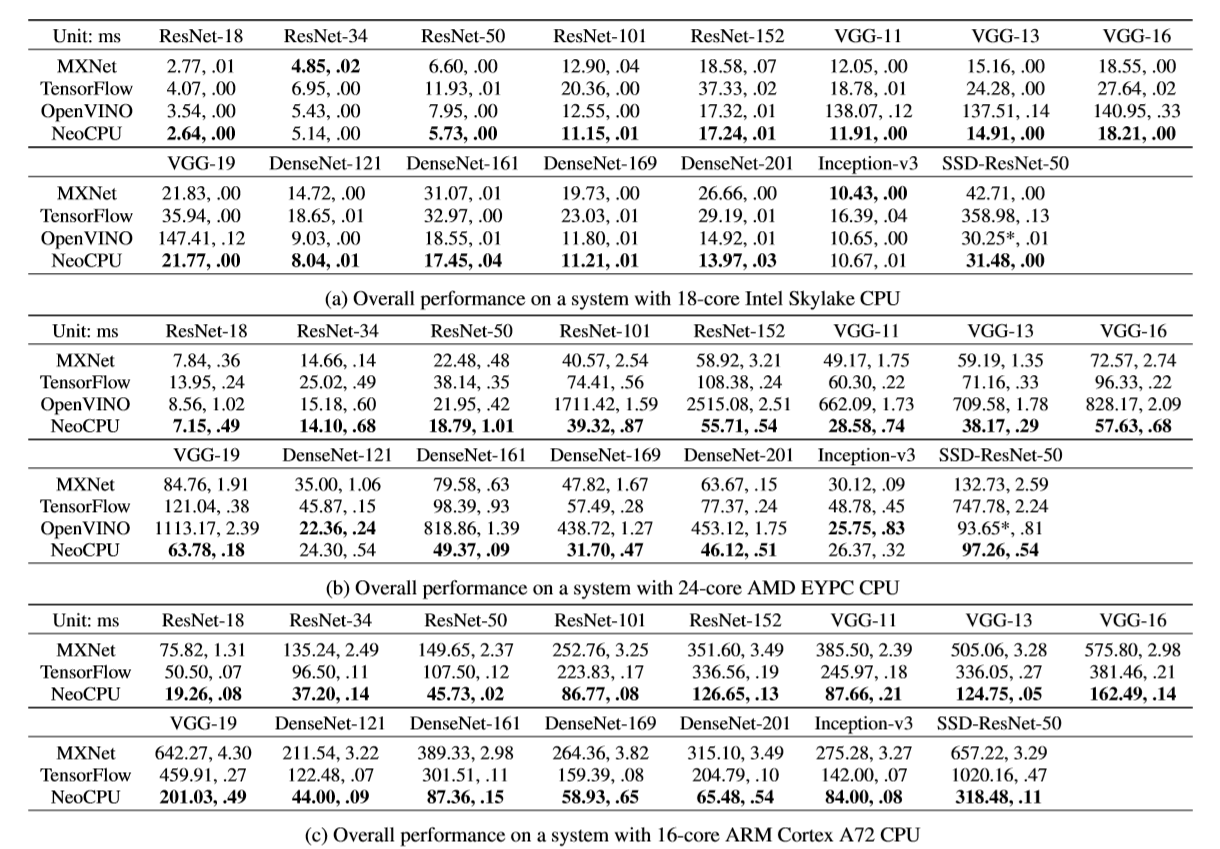


在提出了NeoCPU的思路和实现方案后，作者通过实验来验证，通过多种 CPU的数据对比来评估NeoCPU方案。

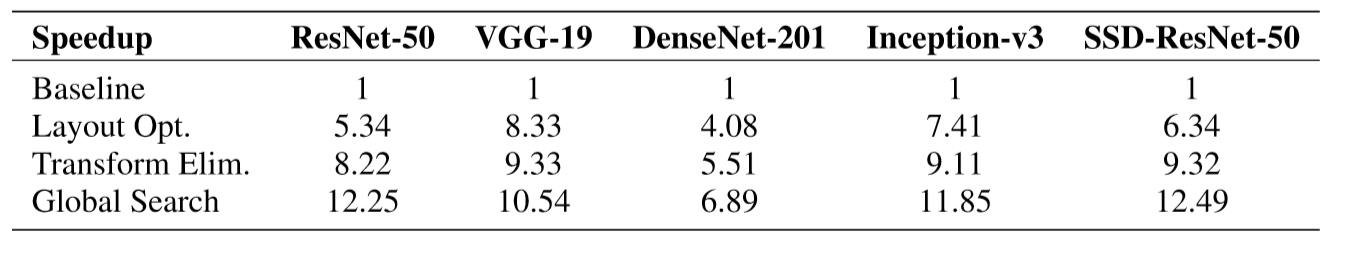
所有实验均在Amazon EC2的平台上完成。共在三个厂商（Intel、AMD和ARM）的CPU十五种常用的CNN模型上评估了NeoCPU方案。

首先是整体的评估结果，具体如图4。

可以看出，在所有的优化中NeoCPU的性能提升最为优秀。



其次是上述三个优化步骤单独对性能的优化，具体如图5。



可以看出每一个优化步骤可以对前一步骤有1.1-1.5的性能提升。

综合全文，文章提出了关于在CPU平台上端到端的CNN模型上的推理优化，并且通过实验表明在多种CPU上实现了１５种主流的CNN模型上该方案实现了不俗的性能表现。

关于本篇文章，整体的优化思路，划分为几个部分：

• 计算密集型的conv的模板（template）设计。

TVM的工作里很吸引人一点的是，直接去打击GEMM/Conv这类传统意义上高度依赖于人工精细手调的compute-intensive的计算kernel，特定硬件平台上GEMM/Conv手工优化并不容易。

而这里针对conv算子的codegen，并不是传统意义的codegen那么直接（比如直接codegen出对应的LLVM IR或是C代码），而是采取人工撰写了conv算子的执行模板，将直接影响性能的相关参数抽取出来作为template的特例化design choice来进行枚举优化，从而以比较聪明的方式来应对了conv算子的codegen性能优化问题。

• 配合conv算子的图优化。

解决了就算密集型算子的优化问题以后，就涉及到计算图层面的优化了。计算图层面可以做的优化工作比较多，包括算术化简，计算图的融合等等。有些计算图层面的优化可以认为是跟算子优化正交的，有些则是存在依赖的。比如在这篇文章里针对Conv算子，引入了针对特定数据布局的优化动作，这就会涉及到输入的数据转换操作，为了确保最终执行语义的正确性，也会需要在计算图里合适的位置插入数据转换的操作。这些转换操作结点的插入，在确保单个局部算子性能优势的同时，也会引入潜在的数据转换开销，所以实际上需要从全局视角算一笔账，对各个计算结点的算子要看一下是否需要插入对应的数据转换。

这种布局优化的操作，在其他的主流的深度学习框架里也大体都有涉及，比如TensorFlow grappler的布局优化以及TensorFlow XLA里针对布局进行约束的HLO pass等。

• conv算子粒度+全局图优化范畴的全局优化空间探索。

这是这篇论文里我觉得比较特别的工作的一个体现，因为conv算子的模板表述以及定义空间的枚举是TVM框架里本来就有的功能，配合conv算子优化的计算图优化也算是一个典型的地方。而将二者结合起来，为每个conv算子找到特例化的模板参数定义（如果为每个conv算子使用相同的模板参数定义，那么这里全局优化探索的工作量会小很多），并且结合每个conv算子的特例定义，在计算图全局层面进行优化求解，这是有意思的工作。

文章里提出了两个作法，一个是基于动态编程的确定性算法，另一个是为了避免全局优化探索耗时过长的近似算法。因为当计算图比较复杂以后，确实可能会引入组合爆炸的问题。

• 系统层面的工程实现保障，从而进一步减少系统框架层面的开销。

除了上面的全局优化空间探索的算法。另一个我觉得这篇文章的贡献是将一些比较朴实的系统层面的优化技术引入进来，比较好的确保了整体性能优化的收益 。比如对线程池的实现进行了精化，一方面在数据的切片和线程协调方面引入lock-free的机制避免了线程之间数据共享冲突可能导致的性能损耗，另一方面在实现上也注意了对高速缓存行的填充从而避免了件总线层面的数据访问冲突开销。这些技术手段都属于系统工程优化里常用的技巧，引入到这里，充分利用多核CPU硬件特性来进行硬件加速，还是比较恰如其分。

上面是整篇论文的核心优化思路的一些解析，论文的实验结果，和具体算法细节，我在这里就不详述了，少作知识搬运工，尽可能提供点增益信息。

最后补充一些自己的额外理解：

• 这篇文章比较吸引我的是，这是一篇比较漂亮的针对推理侧任务比较全面的系统全局优化工作，从算子内的性能优化，到计算图层面的优化，自动调整同时引入到算子内和计算图全局优化范畴，并且对于调整算法也做了一定的精细化考量，还在系统工程层面比较扎实地充分发挥了底层硬件的特性以及规避了系统框架层面的开销。如果看每个局部的点，似乎都感觉不算非常优秀，但串接在一起，就构成了一个比较完整的方案。一言以概之，系统性的思维很强。

• 这篇文章针对的主要还是CPU上的优化工作。端到端的性能结果是出色的，不过我也注意到在文章里并没有给出局部conv算子的性能对比情况。在局部conv算子上，全方面超越高性能库还是并不容易，特别是对于Intel这类老牌厂商（论文里更好的性能结果也确实是表现在ARM CPU上）。能够有论文里所展现出来的还不错的性能效果，在non-Intel平台上，有相当一部分来源于算子本身的加速情况，在Intel平台上，则主要还是靠计算图全局层面的优化拉回了局部算子性能可能吃亏的影响。对于NV GPU平台，问题会更复杂一些，因为NV相较于其他硬件厂商，在DL推理场景做了更多的工作，包括算子层面以及全局层面，比如TensorRT、cuBLAS、cuDNN、乃至CUTLASS甚至更全局优化视角的Inference Server，这就使得通过直接打击局部conv算子来压榨更多性能优化空间的挑战更大，不过，随着T4这类新硬件的出现，以及workload的推陈出新，我个人对这方面还是持一个乐观的态度。