Design and Implementation of Deep Learning 2D Convolutions on modern CPUs

JX-Ma

2024/1/11

1 笔记

1.1 摘要

这篇文章提出了一种加快深度神经网络中卷积层执行速度的新方法。所提工作具有普适性，也可应用于其他处理器家族，如 Arm。通过应用编译器优化，如矢量化、寄存器阻塞和循环平铺，以更有效的方式，所提出的工作在最先进的英特尔 oneDNN 库上实现了高加速值。这是通过开发一种寻找优化参数的解析建模方法来实现的。后面的实验在两个英特尔 CPU 平台上，对 DenseNet-121、ResNet-50 和 SqueezeNet(包括 112 个不同的卷积层)，以及 FP32 和 int8 的输入/输出张量 (量化) 进行了全面的实验评估。实验结果表明，上述模型的卷积层执行速度从 x1.1 倍提高到 x7.2 倍。

1.2 介绍

介绍部分:GPU 被认为是训练 dnn 的更好选择，在某些情况下 CPU 可能比 GPU 更适合训练 dnn。如 cpu 在训练小型模型或数据集效率更高,cpu 的大内存容量使大型数据集训练模型更加容易,cpu 已经广泛部署在数据中心和边缘设备中。加速 dnn 卷积层有很多不同的优化策略，我们需要更具硬件架构在不同层中调节很多的参数，如果想要从合适的参数中找到最优化的结果需要测试非常多的优化策略。为了解决这些问题，很多 cpu/gpu 供应商提供了优化库，如 onednn,oneDNN 是一个高度优化的供应商库，由英特尔工程师多年来开发和优化.oneDNN 使用即时编译 (JIT) 在运行时根据输入参数生成最佳代码。
这篇文章的工作，1.. 一项研究工作为基于目标层参数、量化水平和 HW 架构高效设计和实现二维卷积层提供了理论背景，2. 一个促进优化过程的分析模型 3. 一个实验过程，表明所提出的工作在两个 CPU 上实现了比 oneDNN 更高的性能增益。

1.3 优化

1.3.1 向量化

矢量化受到输入张量的布局强烈影响，输入张量流行的布局有 bdyx,byxd,dyxb. 其中，b,d,y,x 代表输入张量的 batch channel height width byxd 布局，代表输入张量中 channel 相邻之间的元素是连续的，也是就相邻元素之间共享 byx 索引。（我们之前使用的布局是 bdyx 布局）为了使用这个布局，我们需

要在第一层卷积中改变输入张量和卷积核的布局，因为第二层的输入，就是第三层的输出，因此我们可以通过将输出张量的元素按照它们需要在下一层读取的确切顺序存储来避免改变输入张量的布局。
矢量化可以应用于一个或两个循环，因此出现了许多矢量化选项。在这项工作中，为了找到要向量化的循环，进行了理论分析。向量化包含在 out 数组下标中的循环是高效的，也就是可以向量化输出张量的宽，第一，输出的元素不是一个一个存储在内存中，而是存储在向量中，第二，累加器向量中的元素直接存储在内存中，它们不是水平相加; 这就导致了更少的存储和算术指令。如果 in 的布局不合适，向量化 b、x 或 y 循环的效率就不高，因为改变 in 的布局的开销是显著的。对于向量化输入张量，向量化不包含在 in 下标中的循环是有利的，因为为在这种情况下，向量化更容易应用于输入张量的不同内存布局。虽然有三个循环没有包含在 in 下标中 (m, k.y 和 k.x)，但只有 m 是一个有效的选项，因为其他两个不能提供足够的并行性。

1.3.2 寄存器阻塞和数据重用

寄存器阻塞可以显著减少执行 L/S 指令的数量，因此，可以实现更高的算术强度 (AI) 值; 因此，程序在 roofline 模型中取得了更好的位置，这就是为什么寄存器阻塞是到目前为止最关键的优化。在哪些循环中应用寄存器阻塞，以及使用哪些因子。探索空间很大，测试所有不同的解决方案非常耗时。方法 1 对输出张量的四个维度和卷积核的 channel 进行寄存器分块方法 2 对卷积核的宽度和高度这两个维度进行分块。方法 1 在大多数情况下可以提供很好的性能，方法 2 则在寄存器级别中实现了更好的数据重用，在方法 2 中输入张量元素都只加载一次，在缓存中没有很好的数据重用，而在卷积时输入张量的各窗口之间存在很多的数据重叠。但是方法 1 需要数据可以刚好放入缓存中，例如如果输出张量分块不均匀时，这个时候就需要使用 padding 填充，有额外的性能损耗。因此当方法 1 的数据无法放入缓存中时，方法 2 优于方法 1

1.3.3 减少微内核的内存占用

当微内核的数据占用无法装入缓存时，实现 L/S 指令最少数量的寄存器阻塞解决方案可能不是最快的; 有一种更优雅的方法可以在不牺牲 L/S 指令数量的情况下减少微内核的数据占用。为此，我们保持寄存器阻塞因子不变，并根据需要保存的内存大小减小 m0 值。当 m0 减少我们适当需要增加 d0 的值。并行化，循环平铺和循环排列并行化：采用 OpenMP 框架实现并行化。所用线程的数量等于物理 CPU 内核的数量 (设为 c)。这里并行化的循环是 b 或 y，取决于目标 DNN 和 HW 参数。被并行化的循环总是被设置为最外层。循环排列: 最里面三层循环依次为卷积核高，宽，channel; 这样，out 数组不加载，只存入内存一次，减少了算术指令和 L/S 指令的数量。B 循环被并行化 (输入张量 batch)，因此被设置为最外层。剩下的循环是 (m, y, x)，它们可以被置换。然而，通过使用推理可以减少不同排列的数量，因此只有两个候选循环排列是传播的，即 (m, y, x) 和 (y, x, m)。

1.4 实验部分

实验在两个不同的 Intel 平台上执行，HW1 和 HW2
HW1: 由一个双插槽 10 核 NUMA Intel Xeon Silver 4210CPU(总共 20 个物理核) 组成，频率为 2.20GHz, AVX-512，每核 32KB dL1 和 1MB L2, 27.5MB L3(总共)，128GBDDR4 3200MHz，运行 Ubuntu 20.04.4 LTS, oneDNN v2.6.0 和 icc 版本 2021.6.0。

HW2 由一个四核 Intel i5-7500 CPU 组成，频率为 3.40GHz, AVX-256，每核 32KB dL1 和 256KB L2, 6MB L3, 16GB DDR4 2666MHz，运行 Ubuntu20.04, oneDNN v2.3.0 和 icc 版本 2021.5.0。在 Intel 的最佳实践和另一个论文中提供的融合实现来做对比。后面在三个流行的 cnn，即 DenseNet-121, ResNet-50,SqueezeNet。进行性能评估。得到在第一种评估方式中获得了较高的加速比值，在具有相同输入参数的一层多次运行的特殊情况下获得了显著的加速比值，寄存器分块是目前为止性能最关键的优化，选用合适的分块因子是非常重要的。

2 总结

我前几周所做的优化包括 Index hoist, output element hoist, Register block, simd, 其中 index hoist,output element hoist, 都是只有一种，而对于 Register block 就存在多种组合，首先对于分块因子的选择，我可以选择按照 4 分块，8 分块或者 16 分块，而分块的对象我可以选择输出张量的 width,height,channel,batch, 这也是这篇论文的分块的方法一，但是对于这些分块排列组合的选择有很多种，我可以按照输出张量宽度分块 4,8,16，高度 4,8,16，然后这两种策略排列组合就有 9 种，而方法 2 中也就是我之前说的对卷积核进行 unroll，这也包含了很多种选择。因此我觉的之前学到的 roofline model 可以应用在分块的地方，分块的选择有很多种，不可能一个一个的去测，这个时候可以借助 roofline model 去选择最优的策略，因为输入张量卷积的窗口之间重复元素很多，对卷积核进行 unroll 可以很好的利用这些重复的元素。之前做的实验，当步长为 4 时，我们按照 8 对输出张量的宽进行分块，这个时候输入张量在第一次循环每个窗口第一行的第一个元素相隔为 4，如果此时我们选择对卷积核进行 unroll 4，这个时候就利用到了输入张量每个窗口第一行相隔之间的三个元素，这样做的好处可以减少 cache 不命中的次数，因此对于寄存器分块这一步优化是对整体性能优化很重要的一步。