week-19

JX-Ma

2024/1/26

1 本周工作

这周我根据上周的优化，尝试着更改了循环顺序，还有使用了向量指令的水平相加指令，后面对 CONV1 几个优化加上了 openmp, 并统计了他在线程数为 1,2,4,8,10,16 的 gflops.

2 实验部分

2.1 实验环境

- 系统: Ubantu 22.01
- gcc version : 9.5.0
- 优化选项: -O3 -fsse -favx2 -fmadd
- cpu:AMD Ryzen 7 6800H 3.20GHz

2.2 优化策略一

这个优化主要做的是让 simd 寄存器存储的数据没有浪费的部分，主要应用于卷积核和输出张量的宽，但是经过实验发现这样是有问题的，先拿卷积核来说，如下图：
我们本来使用的是 128 位寄存器存储 4 个 FP32 数据，上周的做法是使用 padding 填充，如果我们不使用 padding 填充的话，直接存 4 个数据，这样存的数据对应卷积核第一行所有元素和第二行第一个，这些元素都是连续的没问题，但是放在输入张量的窗口上，第四个元素是第二行的第一个，这个元素与前面第一行并不是连续的。
之后再看看作用在输出张量上面，因为上周给的建议是在我按照 4x20 分块的时候，可以直接使用 10 个 256 位寄存器存储这些数据，同样在第一行没啥问题，但是到第二行卷积时，窗口会出现问题，如下图的例子
输出张量第一行的最后一个元素对应输入张量窗口的第一个元素是橙色部分的元素，而输出张量第二行的第一个元素对应的输入张量窗口第一元素是黑色部分，他们直接也并不是连续的。

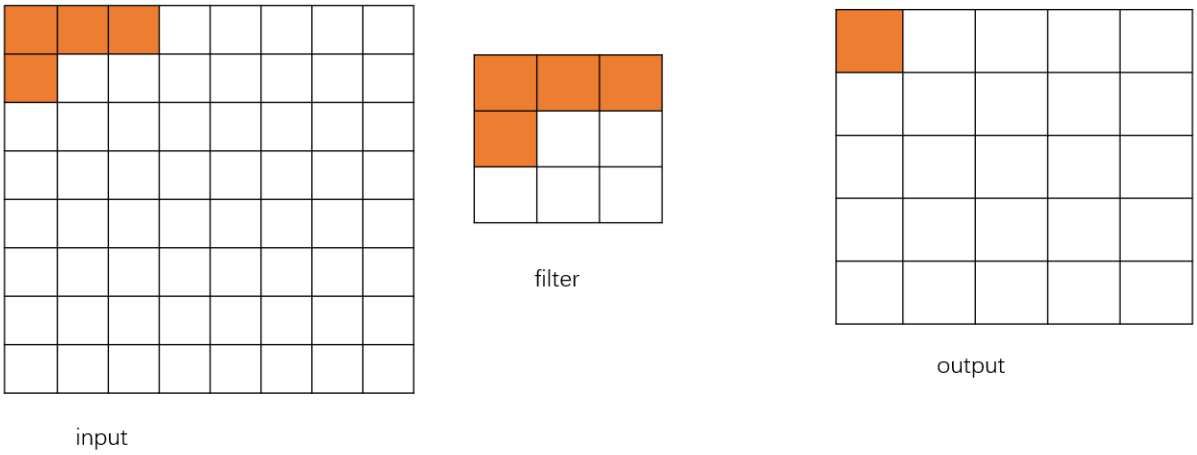

图 1: simd1

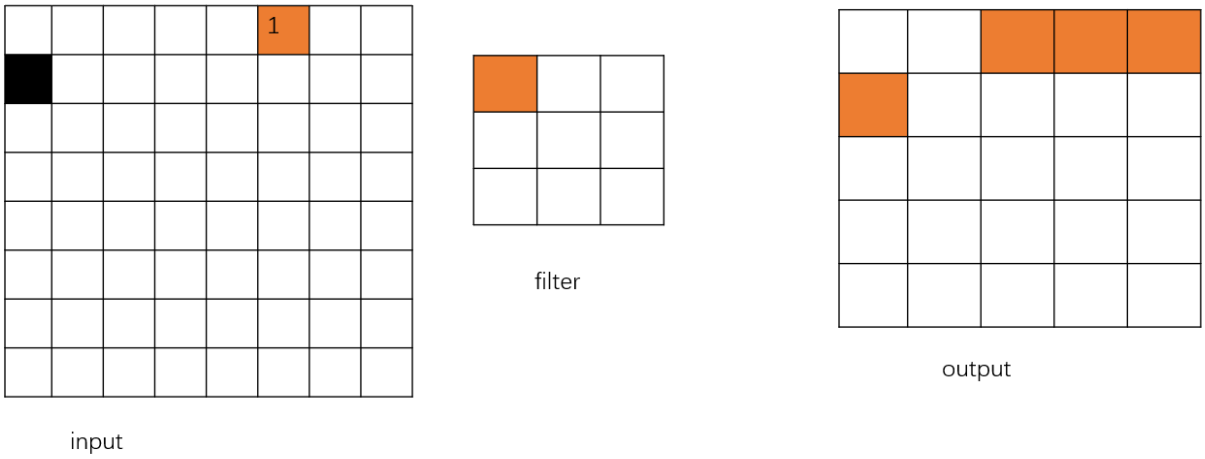

图 2: simd2

2.3 调整循环顺序

根据张新鹏前几周的论文我调节了循环顺序，具体循环顺序为 output.batch->output.channel->input.channel->output.height->filter.height->output.width->filter.width, 因为 simd 存储数据是输入张量窗口内连续的元素，所以选用 fiter.width 作为最底层，在使用第二种 simd 存储数据方式时，循环顺序的最后两层对调了一下。把调整后的循环顺序作用在上周最优化的方法中，发现所有的 gflops 都变小了。

CONV1: 37.2744->16.2515

CONV2: 42.0359->17.0452

CONV3: 31.5668->14.2324

CONV4: 33.7065->14.7614

CONV5: 63.4516->47.2494

CONV7：17.1666->6.28178

2.4 水平相加

查找了指令集有支持向量水平相加的方法，对 128 位寄存器来说没啥问题，但是对于 256 位寄存器还是不能直接得到我们想要的值如下：
假设我们存入数据:
m1 = 1 2 3 4 5 6 7 8
m2 = 11 12 13 14 15 16 17 18
m3 = _mm256_hadd_ps(m1,m2)
这个时候 m3 值为:
1+2 3+4 11+12 13+14 5+6 7+8 15+16 17+18
我们想要是一个寄存器存储的数据的第一个元素是 1+2+..8，第二个元素是 11+12+..18，因为 hadd 的计算方法，我们做不到这样的数据集合
因此我的处理方法是，存储连续四个 avx 值并让他们相加，如下:
m1 = 1 2 3 4 5 6 7 8
m2 = 11 12 13 14 15 16 17 18
m3 = 21 22 23 24 25 26 27 28
m4 = 31 32 33 34 35 36 37 38
mid1 = m1 hadd m2 = 1+2 3+4 11+12 13+14 5+6 7+8 15+16 17+18
mid2 = m3 hadd m4 = 21+22 23+24 31+32 33+34 25+26 27+28 35+36 37+38
mid3 = mid1+mid2 = 1+2+3+4 11+12+13+14 21+22+23+24 31+32+33+34 5+6+7+8
15+16+17+18 25+26+27+28 35+36+37+38
然后使用 mid3 的第一个加上第五个就能得到我们想要的第一个结果的值。
对于 128 位的 hadd 则完全可以之际将四个元素使用 hadd 得到想要的结果如下
m1 = 1 2 3 4
m2 = 11 12 13 14
m3= 21 22 23 24
m4 = 31 32 33 34
mid1 = m1 hadd m2 = 1+2 3+4 11+12 13+14
mid2 = m3 hadd m4 = 21+22 23+24 31+32 33+34
mid3 = mid1 hadd mid2 = 1+2+3+4 11+12+13+14 21+22+23+24 31+32+33+34
使用这个方法后的 gflops 提升了，虽然提升的幅度不大:
CONV1: 37.2744->39.4872
CONV2: 42.0359->44.79
CONV3: 31.5668->32.9375
CONV4: 33.7065->34.4129
后面的步长为 1，所以没有使用 hadd,

2.5 openmp

这个实验主要是在 CONV1 的方法上使用了 openmp：坐标介绍:
unroll1x4: 是使用 128 位寄存器存储输入张量连续 4 个元素

RB1x4: 对输出张量宽分块。

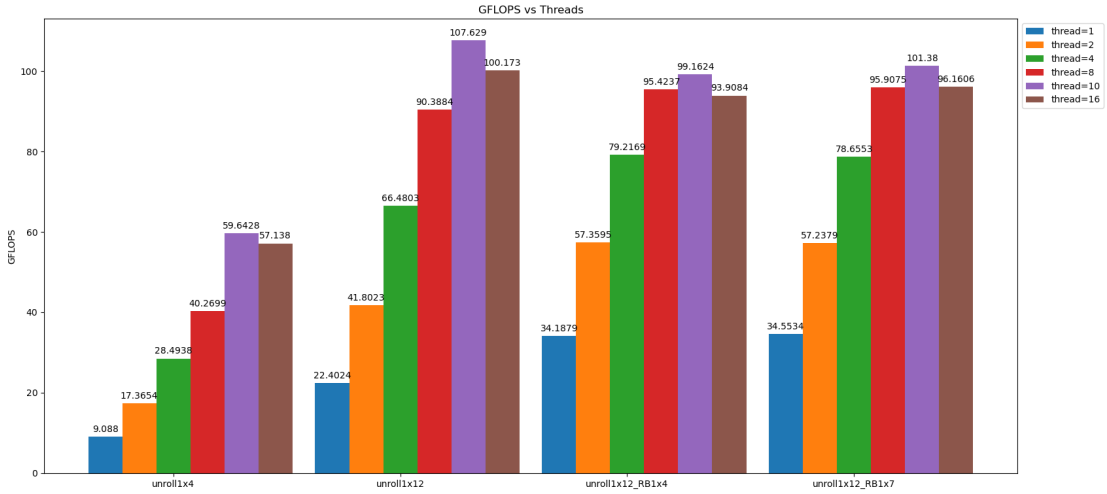

图 3: openmp

从图中可以看到，从线程数 1 到 2 的 gflops 并不会提高 2 倍，提升率也在逐渐减少。我的 openmp 并行化加在 input.batch 之前，而我设置的 batch 大小为 10，所以我在线程数为 10 的时候也记录了一下结果，在实验开始之前，我查了一下自己的 cpu 可以支持的最大线程数为 16。

3 总结

本周主要进行了四个小实验，主要目的是继续优化直接卷积，有效的直接卷积的优化就是使用了 hadd 寄存器水平相加的指令，循环排序是数据连续性较好应该是可以很好的优化算法，并且把输出张量的宽作为最底层的数据连续性应该是比较好的，但是如果数据放在最底层，我们需要不断的写入输出张量宽度上每个元素的值，原来在第四层 for 循环里面写入一次就够了，现在需要在第七层 for 循环里面，每次计算一次数据就需要写入。还有对于寄存器的数据的定义，初始化为 0，这些操作次数也变多了，虽然这些可能并不会在很大程度上影响性能。openmp 的实验可以看出线程选取的数量并不是越大越好，我们需要合理的选取线程数量，使性能达到最佳状态。