week20实验记录

zxp

February 3, 2024

1 environment

cpu:Inter i5-9300H (2.4 GHz)

System:Ubuntu 22.04.1

Compiler:gcc 12.3

2 code

更改了计算的部分

```
for (; l + 8 < length; l += 8){
        vec_a = _mm256_loadu_ps(src+l);
        vec_b = _mm256_loadu_ps(scalar+l);
        vec_c = _mm256_fmadd_ps(vec_a,vec_b,vec_c);
    }
```

将$l+8<length$改成$l+8<=length$。之前写的时候认为$l+8<=length$等于的时候数组会越界，然后发现每次最后8个元素没用上SIMD指令集，因为要用到的是A[L],A[L+1]到A[L+7]，一共8个，$l+8<=length$等于的时候数组并不会越界，就将判断条件改正确。
尝试使用了openMP和unroll和padding去优化im2win卷积（用的第二种数据布局）。为了比较方便的使用openMP我将input的batch设置成了8，openMP的线程设置成了8（和我电脑CPU的线程一致），对历遍output的batch循环使用openMP（通常是对这个纬度使用openMP，有些论文也指出但这个纬度并行性不够的时候应该使用其他纬度的并行性，比如推理的时候batch为1，对output的其他纬度使用openMP也是一样的，同时计算多个output的元素不会对结果产生影响）
尝试让im2win卷积得到的output是im2win张量，但计算得到的一个元素在im2win张量不同位置可能出现两次，所以卷积过程中需要数组来保存结果，然后将得到的结果重新排列放到im2win张量里，这周还没想到什么好的办法实现。

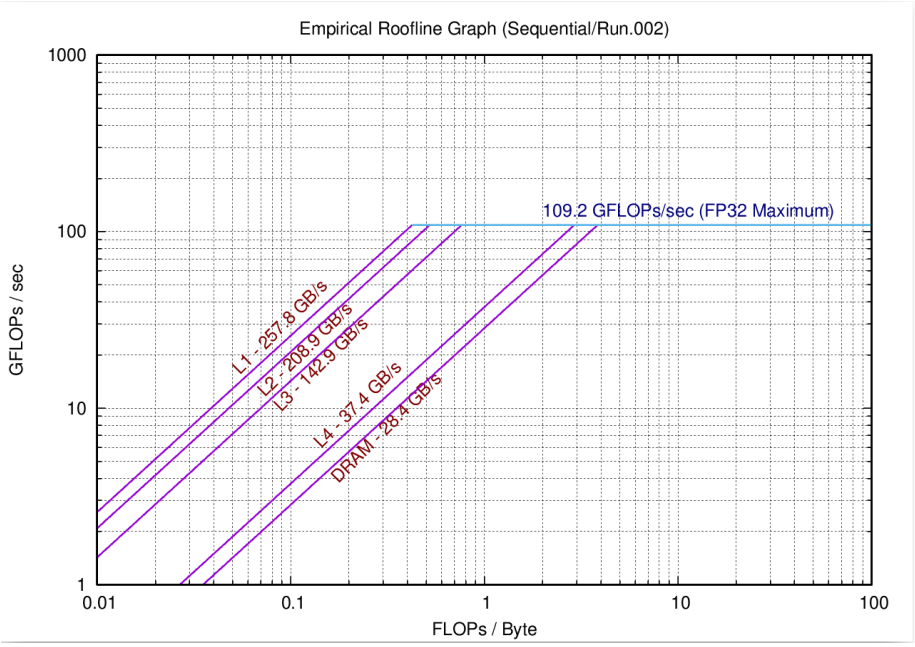

Figure 1: 不开openMP

3 roofline

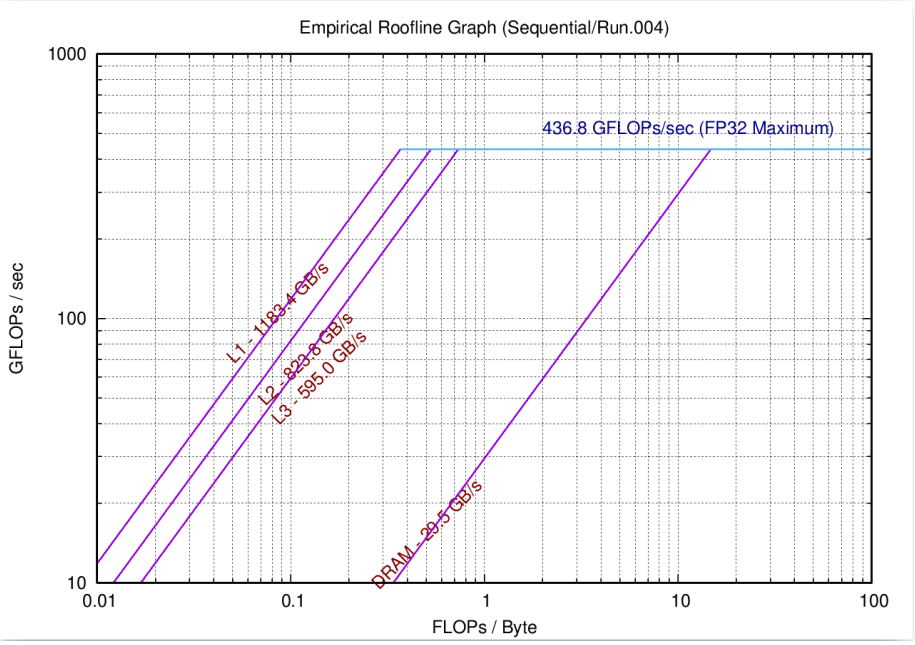

Figure 2: 开启openMP

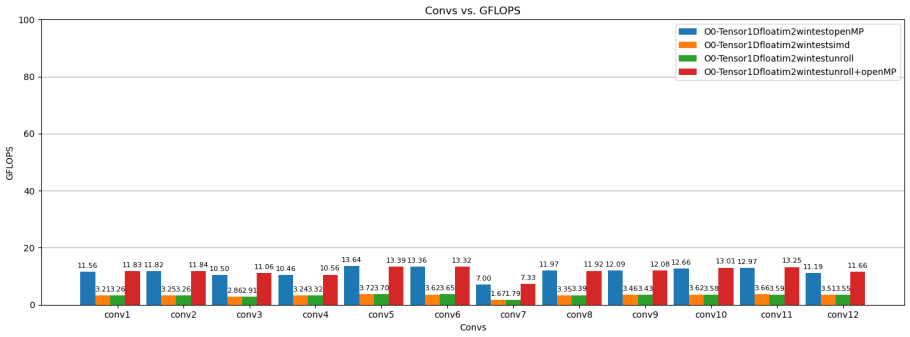

Figure 3: O0

4 Experiment

对im2win进行了优化，使用float。openMP的线程设置成了8，对历遍output的batch循环使用openMP。对output的width这个循环进行unroll，unroll八次计算，并且将这八次计算对应的output的八个元素hosting，使用一个256位的寄存器从output中加载这八个元素。

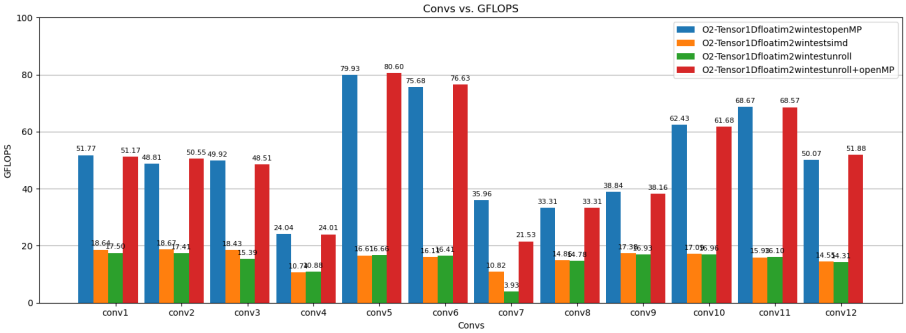

Figure 4: O2

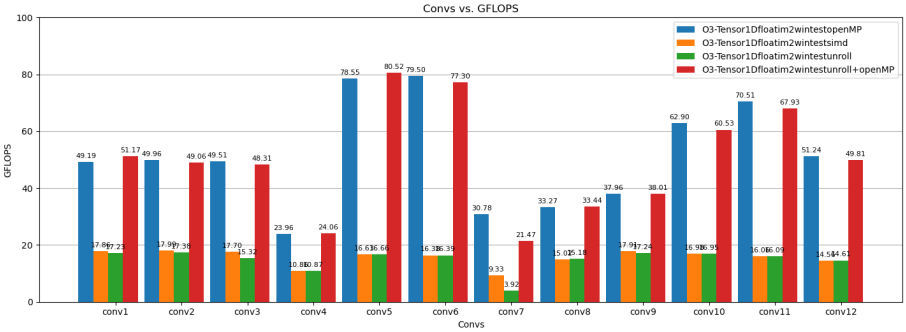

Figure 5: O3

4.1 Analysis

使用openMP的效果很明显，gflop成倍数增长，但unroll效果很不明显，很多conv性能甚至降了，还需要多试。

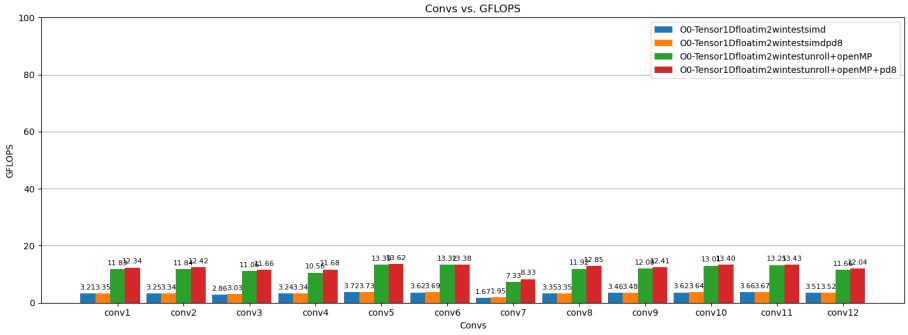

Figure 6: O0

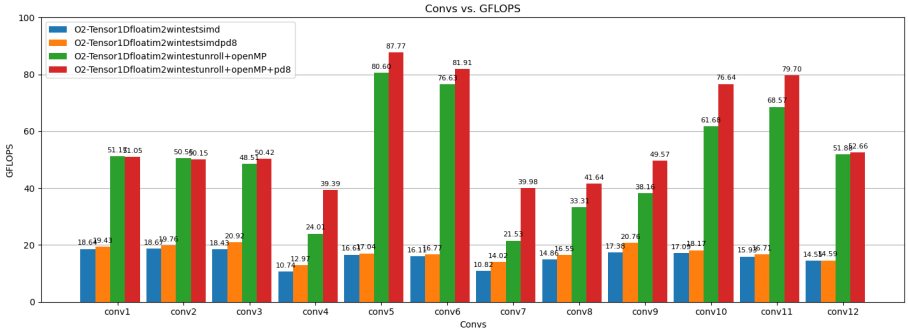

Figure 7: O2

5 Experiment2

还试了填充，在im2win转化和过滤器转化的时候填充。因为数据在内存中是以一维数组的方式储存的，所以只需要填充过滤器和防止数组越界填充最后一个窗口，填充的时候填充至8的倍数（便于使用256位寄存器），填充的是0，这样即使把下一个窗口不参与计算的元素载入向量寄存器也会因为和过滤器填充的0相乘得到0而不影响结果。这周得到的下图，做的时候是对im2win的每行最后一个窗口做了填充，但跑完之后仔细想想，数据在内存中是以一维数组的方式储存的，下一行放在上一行后面，即使不填充也不会越界，只需要整个im2win张量最后一个窗口进行填充（还没试，这周是对对im2win的每行最后一个窗口做了填充），这样填充花费的空间就很少很少。

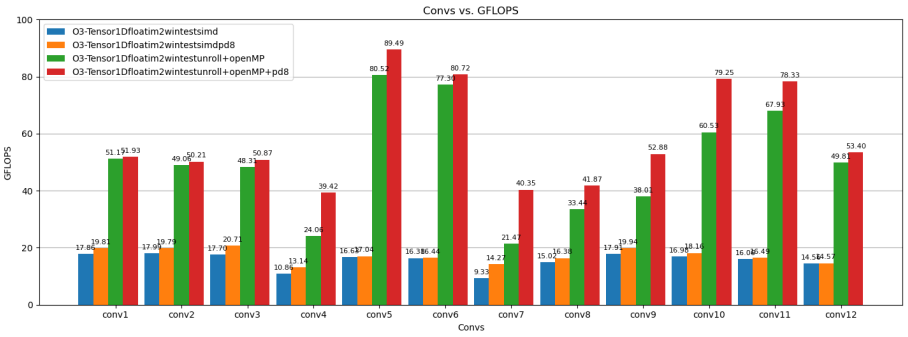

Figure 8: O3

5.1 Analysis

填充的效果还是很明显的，特别是conv7，O2和O3情况下性能增加近了一倍，conv7的过滤器太小了$3 \times 3 \times 3$，总共27个元素，不填充的话，前24个元素可以用3次256位寄存器算完，而剩下的需要独立算3次。填充后可以使用4次次256位寄存器直接算完。接下来的优化应该在填充的基础上做