week19实验记录

zxp

January 27, 2024

1 environment

cpu:Inter i5-9300H (2.4 GHz)

System:Ubuntu 22.04.1

Compiler:gcc 12.3

2 code

改正tensor1D验证正确性部分https://floating-point-gui.de/errors/comparison按博客写的比较部分，将之前写的绝对误差改成博客中的相对误差。
更改了随机赋值的部分，之前用的随机赋值是1.0到10的均匀分布。libtorch中随机赋值有三种，第一种是0.0到1.0的均匀分布，第二种是均值为0方差为1的正态分布，第三种是自己设置范围的均匀分布。我现在换成了和libtorch的第一种相同的写法。改成这样的范围后对比直接卷积和im2win的用之前的比较写法（绝对误差）是10的-4次左右，用这周写的相对误差是10的-7次方左右。wetenor中的掉用libtorch的卷积和直接卷积和im2win卷积结果也是相差10的-4次方左右（我之前的写法是就是模仿wetensor的写法）。

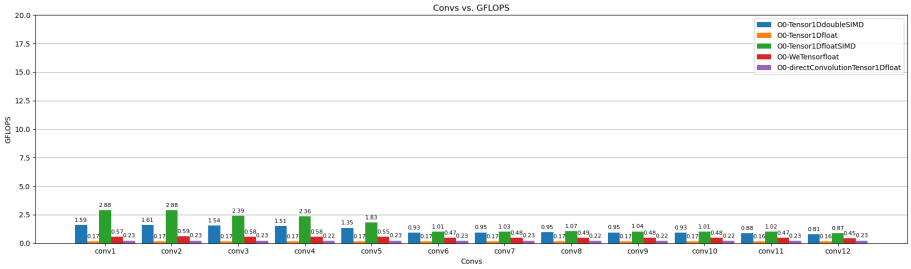

Figure 1: O0

3 Experiment

对im2win进行了优化，用于对比使用float和double的区别，这周写了double版本，优化的方式和float版本除了SIMD指令集一样，做的优化主要是使用SIMD和FMA和index-hosting和hosting。使用simd的地方是窗口部分，在256位寄存器能放8个float数据或者4个double数据，然后使用FMA。hosting优化单纯的hosting了一个输出张量的元素。

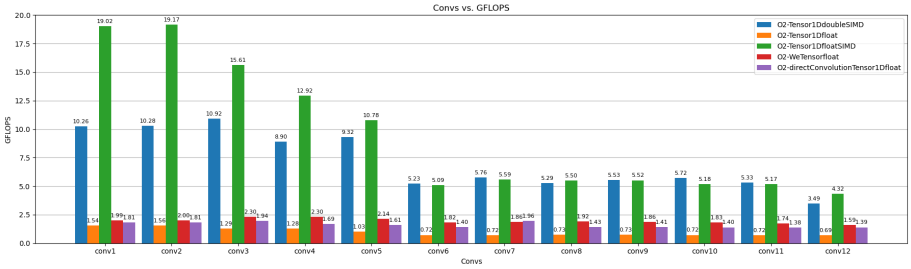

Figure 2: O2

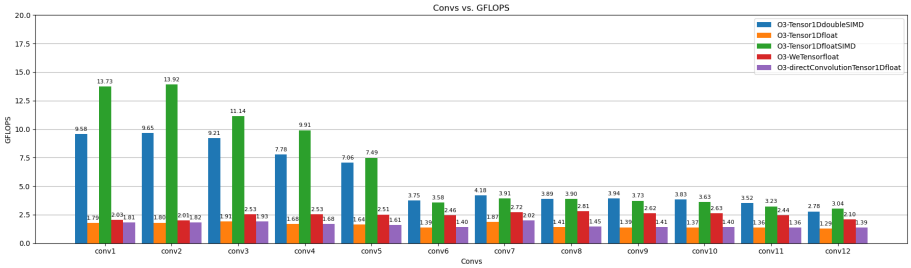

Figure 3: O3

3.1 Analysis

优化后肯定比未优化快，conv1-4上float比double快的很明显，在O2优化下conv1的情况float比double快了近两倍，但conv-6-11就几乎没区别，甚至O2情况下double还快一点点。应该卷积核太小了（conv6-12都是3x3）是计算的数量还不够，在这几个conv使用loop unrolling应该会达到接近conv1的效果

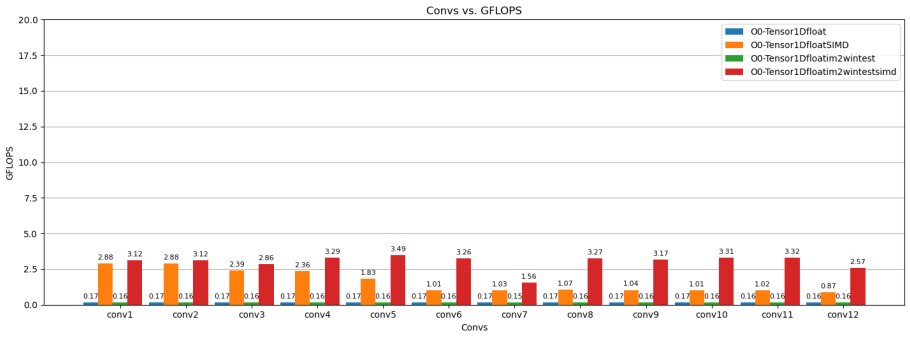

Figure 4: O0

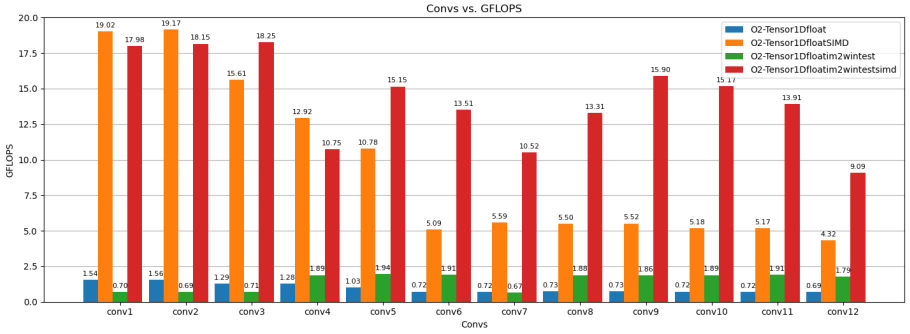

Figure 5: O2

4 Experiment2

对上周试的另一个数据布局进行优化，和上面那个实验用的完全一样，数据类型用的float。

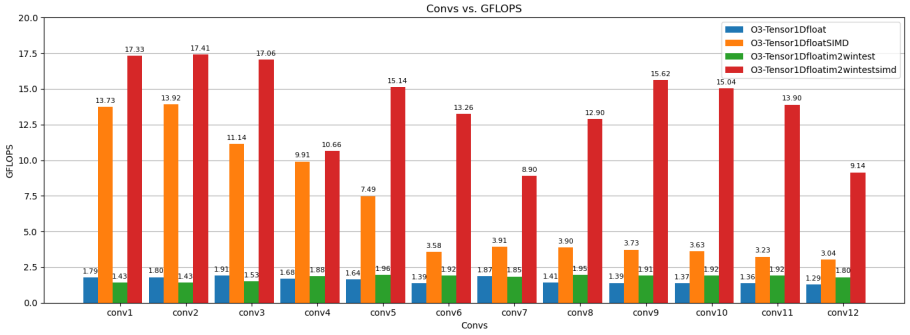

Figure 6: O3

4.1 Analysis

进行优化后新的数据布局表现要好，特别是在conv6-12，之前的数据布局表现不好是一个窗口大小只有3x3，太小了，而这个数据布局把核所有的channel也放在一个窗口，这样窗口要大channel倍，所有效果要好。