week13实验记录

zxp

December 16, 2023

1 environment

cpu:Inter i5-12400f (2.5 GHz)

System:Ubuntu 22.04.1

Compiler:gcc 12.3

2 code

代码和上周一致。测试改成了循环100次取平均值。新增了按论文¡High performance zero-memory overhead direct convolutions ¿提出的优化直接卷积的方法改变直接卷积循环的顺序的直接卷积版本。

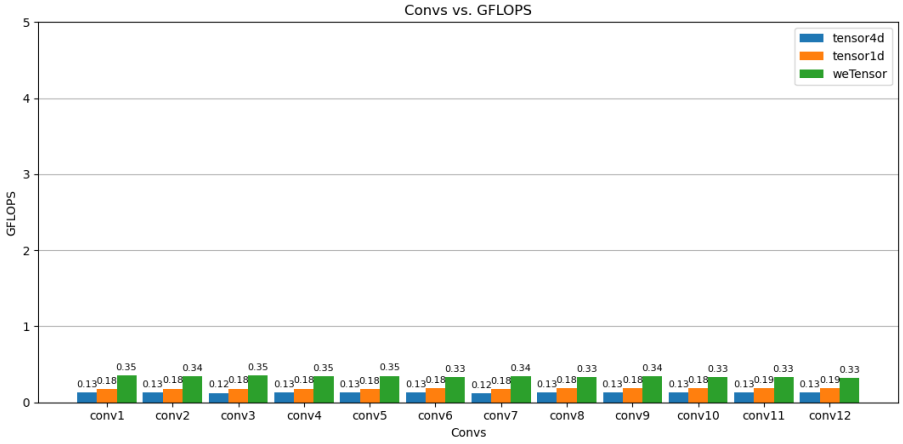

Figure 1: O0

3 Experiment

将不同优化选项的重新测了一遍，输入张量的batch设置为2，tensor1d/tensor4d和wetensor分开测试，12个conv分开测试，测试100次取平均值。折线图效果太差，绘制成了柱状图

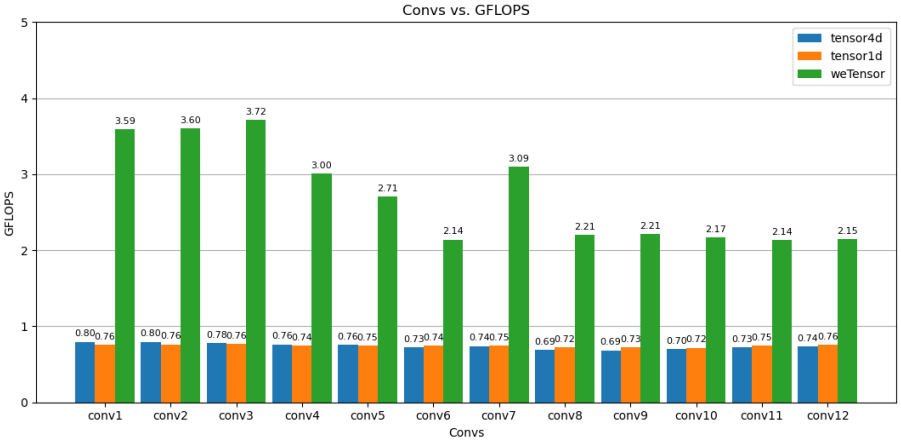

Figure 2: O2

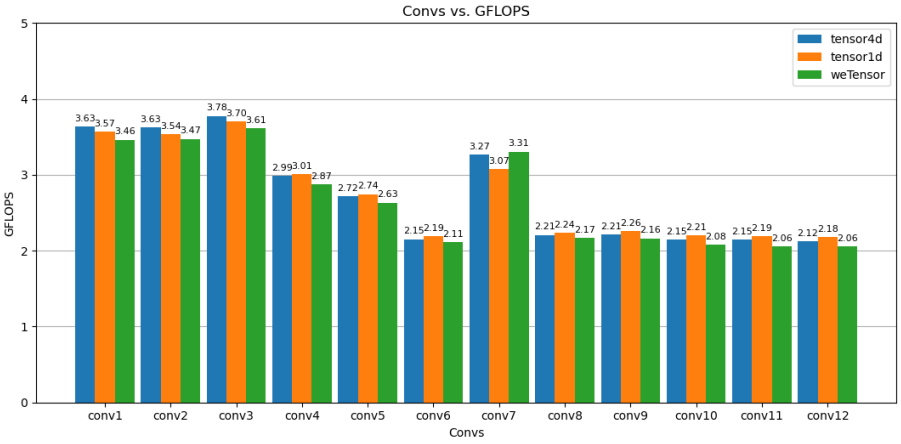

Figure 3: O3

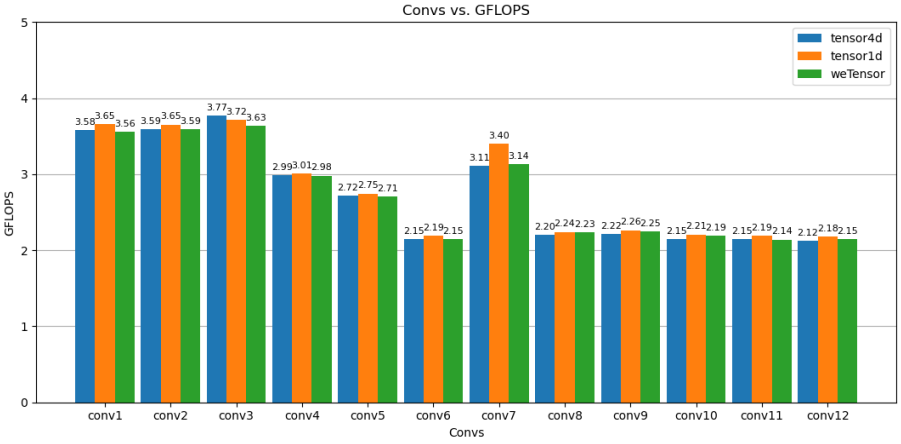

Figure 4: Ofast

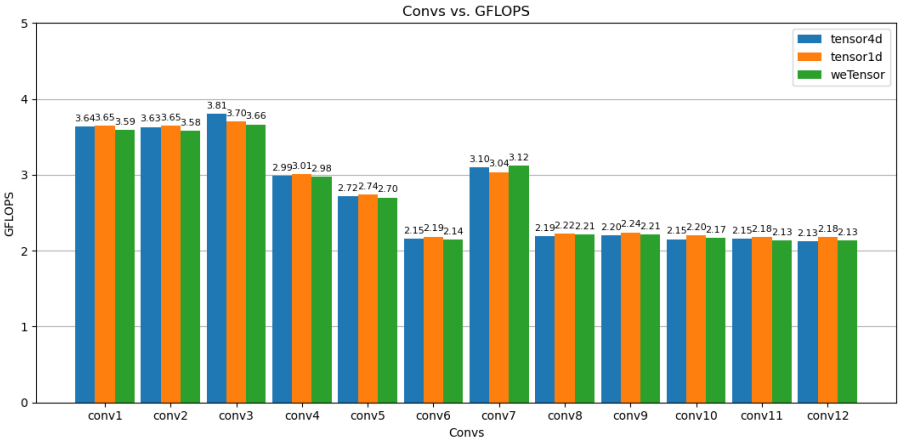

Figure 5: Ofast -march=native

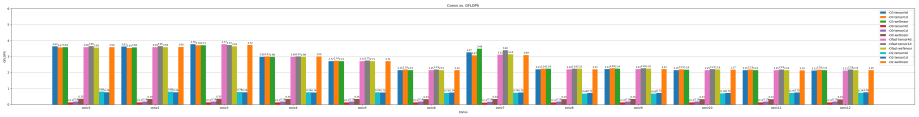

Figure 6: 把O0,O2,O3和Ofast放一起比较

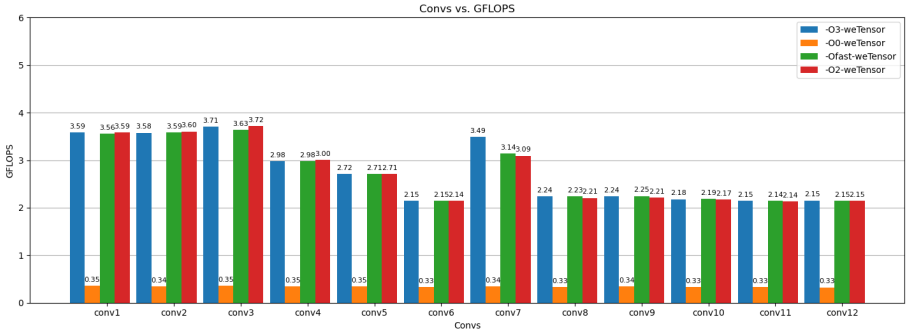

Figure 7: 只看wetensor

3.1 Analysis

因为不需要看conv之间的趋势，柱状图比折线图更适合。改成测试100次取平均值后，O0要跑很久，一晚上8小时跑不完tensor1d和tensor4d和wetenor，但记录的执行直接卷积的时间没这么夸张（最慢的conv4测试一次需要50秒，第二慢的conv8需要20秒，但其他的都是2秒左右），除了直接卷积花费了很多时间，给张量初始化的时候生成随机数也花费了大量时间，应该换一种方式生成随机数。

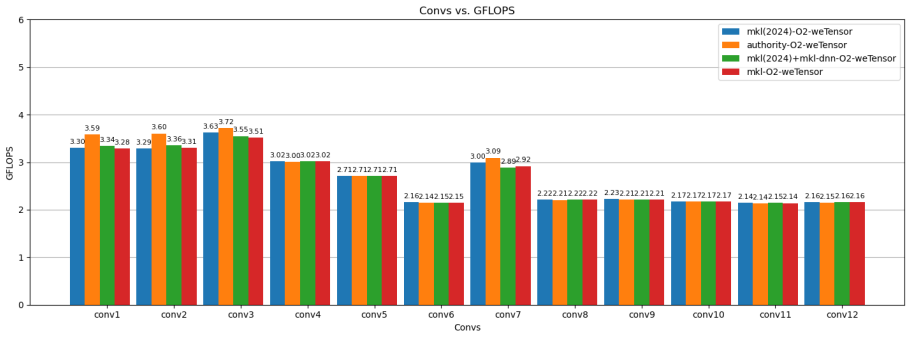

Figure 8: 对比不同版本的libtroch在O2的情况

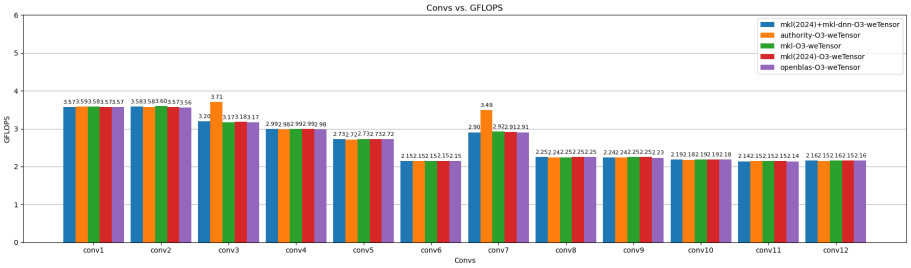

Figure 9: 对比不同版本的libtroch在O3的情况

4 Experiment2

前几周编译的libtorch和官方编译的版本性能有差距。但优化选项都是一样的(-O2 -fno-math-errno -fno-trapping-math)。官方编译的版本在指定编译选项多了一堆信息大部分是用于关闭警告。官方编译的版本开启了很多选项，大部分是加速库。其中大部分都只在GPU版本才能生效（我尝试在编译CPU版本的时候打开，会报警告然后不生效）。然后我编译的MKL版本和官方的差异还有MKL版本不同，前几周编译我并没有自己去下载MKL库，编译完成后libtorch的信息显示MKL版本为2020，而官方编译的版本用的MKL版本为2022，比我编译的版本更先进，于是我把MKL更新到2024。又编译了两个版本，MKL(跟新到2024)和MKL(2024)加上打开MKL-DNN 加速库。然后进行wetensor的性能对比。在编译GPU版本并开启官方编译时开启的加速库时遇到了错误，错误显示libtorch的NVCC子模块出错，还有Caffe 2子模块出错，还在解决，准备在环境中安装这两个包。

4.1 Analysis

换了编译选项效果不明显，还在尝试把官方开的选项全部打开

5 Experiment3

前几周得出的结果中可以看到conv7在不同优化下表现十分不稳定。从conv7的输入张量尺寸特别大(长宽244×244)而卷积核特别小(长宽3×3)，按之前的直接卷积，最内层历遍卷积核，数据没有连续性论文《High performance zero-memory overhead direct convolutions》指出为了饱和运行算，应该让输入张量的行/高在更里面的循环新增了按论文《High performance zero-memory overhead direct convolutions》提出的优化直接卷积的方法改变直接卷积循环的顺序的直接卷积版本。（图中带r的就是改变了循环顺序的版本）

```
//初始的循环顺序
Input: Input I, Kernel Weights F, stride s;
Output: Output O
for i = 1 to $b_O$ do
    for i = 1 to $C_O$ do
         for j = 1 to $H_o$ do
              for k = 1 to $W_o$ do
                   for l = 1 to $C_f$ do
                       for m = 1 to $H_f$ do
                            for n = 1 to $W_f$ do
```

```
//论文<High performance zero-memory overhead direct convolutions 提出的循环顺序>
Input: Input I, Kernel Weights F, stride s;
Output: Output O
for l = 1 to $H_o$ do
     for n = 1 to $H_f$ do
          for m = 1 to $W_f$ do
               for i = 1 to $C_i$ do
                   for k = 1 to $W_o$ do
                        for j = 1 to $C_o$ do
                            $ O_{j,k,l} += I_{i,×ks+m,
                                ×ls+n} × F_{i,j,m,n}$
```

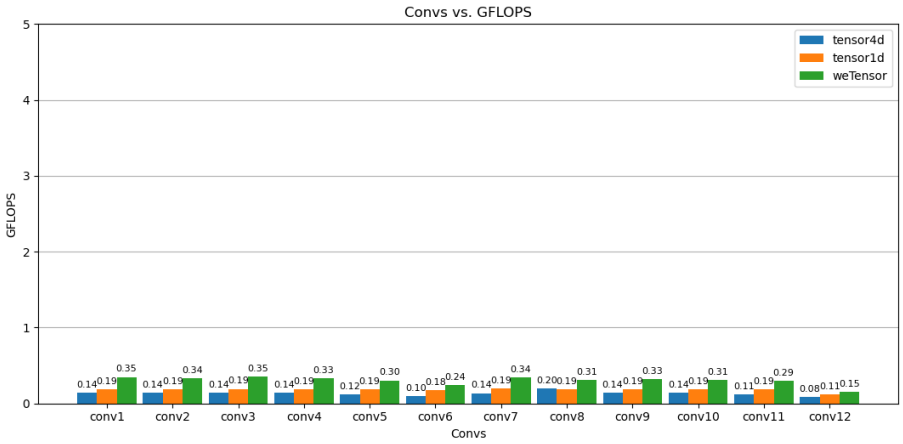
Convs vs. GFLOPS
tensor4d
tensor1d
weTensor
GFLOPS
Convs
conv1
conv2
conv3
conv4
conv5
conv6
conv7
conv8
conv9
conv10
conv11
conv12

Figure 10: O0

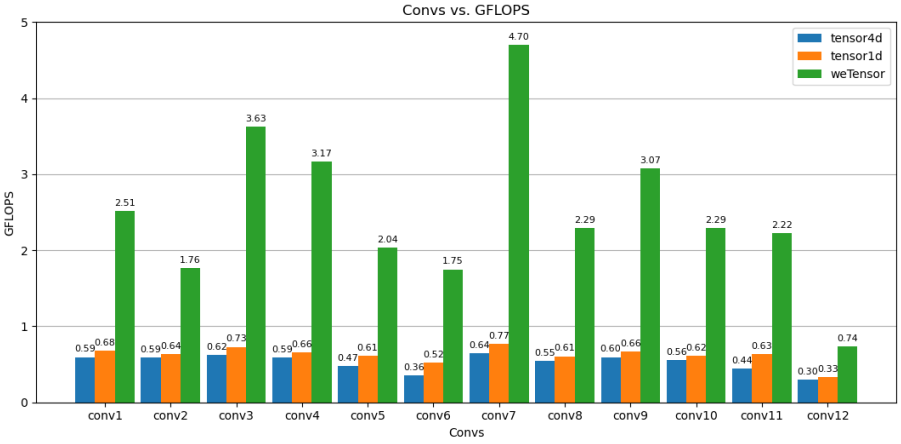

Figure 11: O2

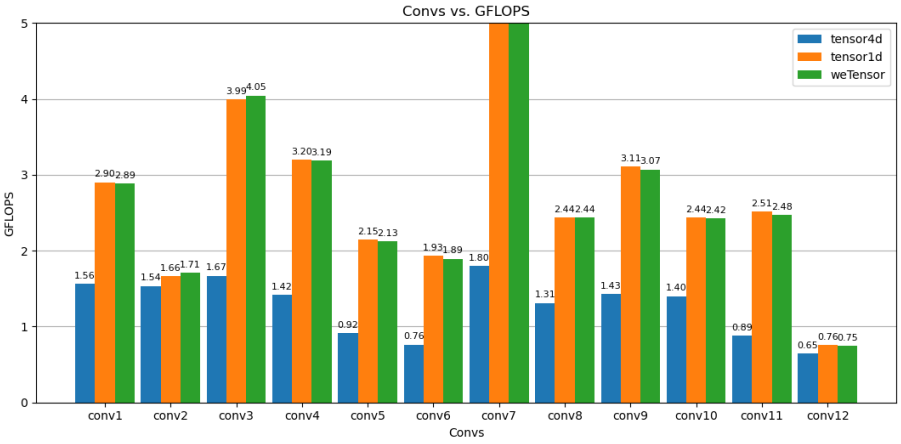

Figure 12: O3

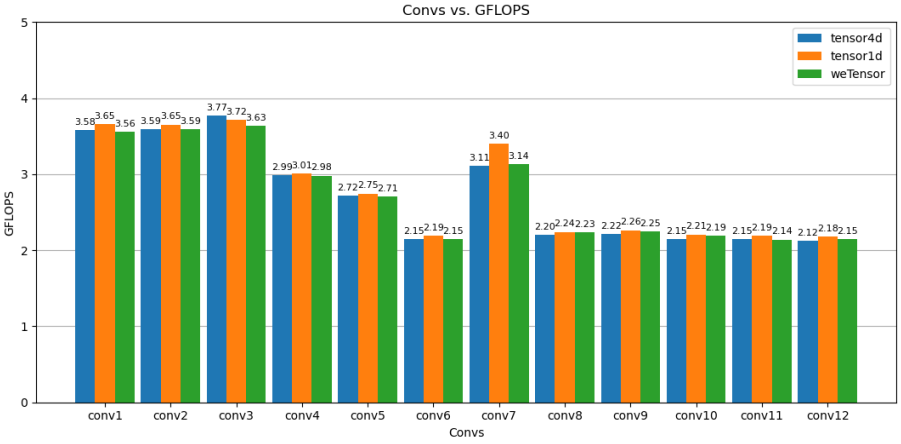

Figure 13: Ofast

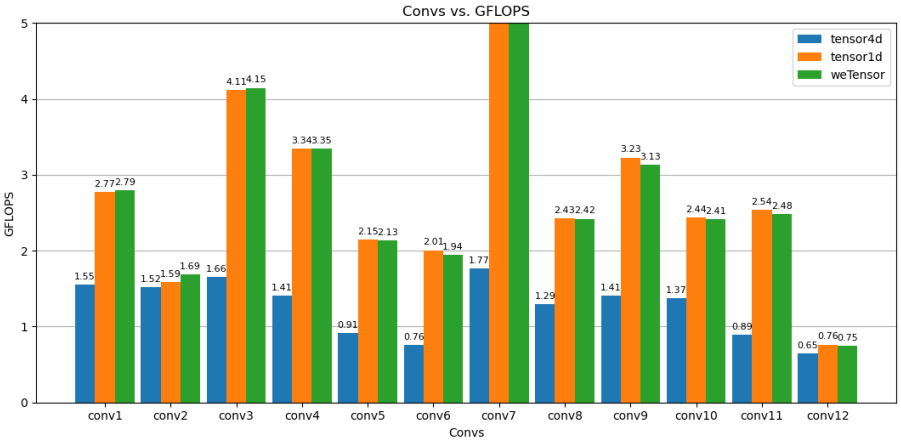

Figure 14: Ofast -march=native

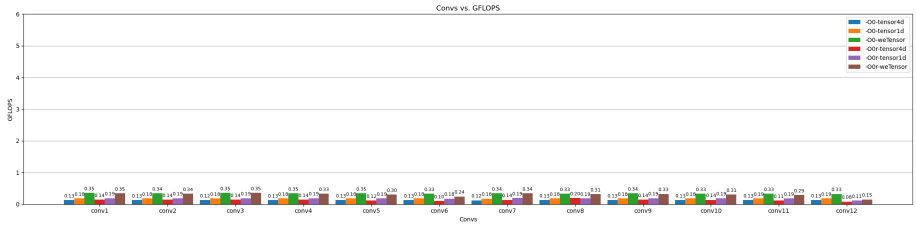

Figure 15: O0放一起对比

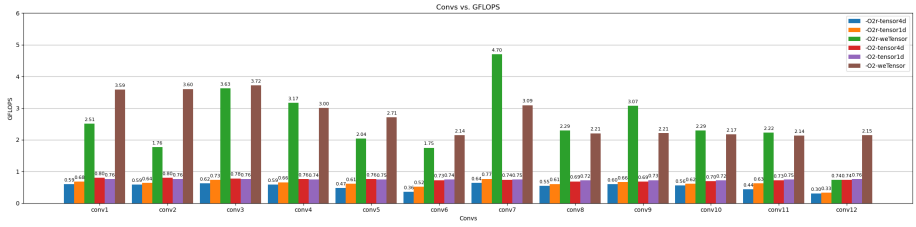

Figure 16: O2放一起对比

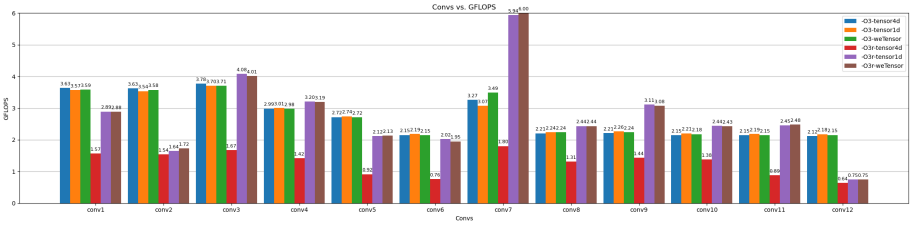

Figure 17: O3放一起对比

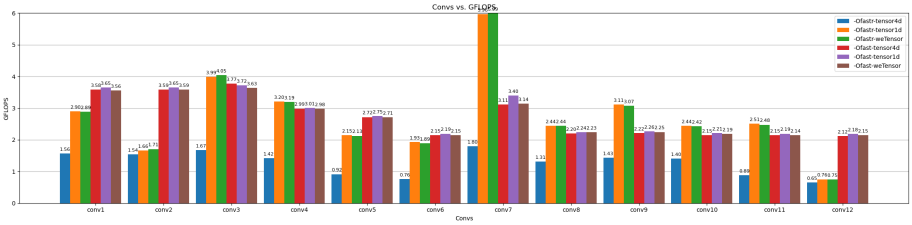

Figure 18: Ofast放一起对比

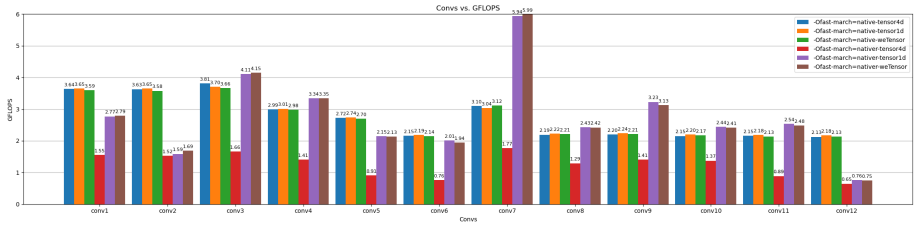

Figure 19: Ofast -march=native放一起对比

5.1 Analysis

更改循环顺序后，conv7的性能变化十分夸张，虽然在O0情况下不明显，但O3情况下wetensor的gflops甚至达到了6，比不改变循环快了近1倍。但是tensor1d和tensor4d的表现差异巨大，在O3情况下，tensor1d性能上升了，但tensor4d性能反而下降了。出现这种情况应该是因为把输入张量的height放在了比较里面的循环，但tensor4d在height上是不连续的。

```
Tensor4D(int64_t N, int64_t C, int64_t H, int64_t W) {
        this->batch = N;
        this->channel = C;
        this->height = H;
        this->width = W;
        // 给四维数组分配空间
        this->tensors.resize(N);
        for (int64_t i = 0; i < this->batch; ++i) {
            this->tensors[i].resize(C);
            for (int64_t j = 0; j < this->channel; ++j
                ) {
                this->tensors[i][j].resize(H);
                for (int64_t k = 0; k < this->height;
                    ++k) {
                    this->tensors[i][j][k].resize(W);
                }
            }
        }

    }
```

我推测改成其他顺序tensor1d和4d表现会更好，因为数据结构和论文中用的不一样，而且论文中最内层选择输出张量的channel是为了更好地利用SIMD指令，但是我并没有用到SIMD指令，选择输入张量的width或者height在数据连续性上会更好。并且，虽然conv7效果好了，但其他conv表现并不好，比如conv1和conv2和conv12。conv12是因为输入和卷积核的width和height特别小，但conv1和conv2的也是输入的width和height比卷积核大（虽然没conv7差异大）。

还需要尝试更多的循环顺序和论文中提出的其他优化

6 Experiment4

上周发现使用gcc12.3编译器在O3情况下，wetensor在conv3的表现大幅下降，这个实验是为了找出是哪个优化选项导致的。

_-O3	weTensor	3.5856	3.57638	3.70936	2.98152	2.71547	2.14944	3.4895	2.23541	2.23731	2.18016	2.14781	2.15256
_O3_2	weTensor	3.46342	3.47429	3.61079	2.87133	2.63348	2.11023	3.30628	2.16788	2.16282	2.08091	2.05842	2.06124
_-O2	weTensor	3.58765	3.60406	3.7182	3.00457	2.70652	2.14146	3.09439	2.2068	2.20962	2.17217	2.13696	2.14796
_-O2-fpeel	weTensor	3.56856	3.5927	3.74364	3.00465	2.70519	2.14611	3.20961	2.20544	2.21479	2.16388	2.13894	2.14962
_-O2-fsplit	weTensor	3.5919	3.61259	3.76895	3.00485	2.70442	2.14883	3.0898	2.20637	2.21789	2.16654	2.14299	2.14935
_-O2-fvers	weTensor	3.5951	3.58706	3.75115	3.00766	2.70505	2.14563	3.17116	2.20652	2.2175	2.1694	2.14376	2.14803
_-O2-ftree	weTensor	3.57689	3.6057	3.71187	3.0068	2.70502	2.14725	3.17818	2.20602	2.21367	2.17125	2.13961	2.15325
_-O2-fsplit	weTensor	3.6028	3.6065	3.80011	3.00499	2.70291	2.15014	3.10715	2.20654	2.22018	2.16944	2.14427	2.14917
_-O2-fpred	weTensor	3.59518	3.59483	3.71482	3.00655	2.7072	2.14727	3.19309	2.20555	2.20903	2.16579	2.14557	2.15238
_-O2-floop	weTensor	3.60932	3.60753	3.72091	3.00678	2.70828	2.14445	3.2274	2.20704	2.21533	2.16343	2.1439	2.15239
_-O2-floop	weTensor	3.59954	3.6157	3.73323	3.00473	2.70809	2.14153	3.18688	2.20489	2.21282	2.17441	2.14942	2.15592
_-O2-fvect	weTensor	3.60929	3.61608	3.72672	3.00918	2.70203	2.14703	3.12809	2.20495	2.21512	2.17375	2.13667	2.15074
_-O2-fipa-c	weTensor	3.59969	3.60857	3.7884	3.00437	2.70355	2.15618	3.22495	2.20441	2.20819	2.17542	2.14967	2.15572
_-O2-funsv	weTensor	3.58639	3.58345	3.61759	2.98217	2.71604	2.15192	3.39758	2.23549	2.24661	2.18039	2.15215	2.15893
_-O2-ftree	weTensor	3.60744	3.60672	3.74803	3.00531	2.70779	2.15332	3.25041	2.20688	2.21735	2.17303	2.14452	2.14961
_-O2-fgcse	weTensor	3.41739	3.53002	3.63451	2.91144	2.65788	2.13162	3.08614	2.20458	2.22225	2.16345	2.1465	2.15417

Figure 20: 把结果放表格中

6.1 Analysis

这周并没有出现上周的情况，wentensor在conv3的情况下的O3的表现和O2接近。担心是偶然，我又测试了一遍，结果还是和上周不同，然后我还是把O3比O2多开的每个优化都尝试了一遍，结果每个优化下conv3的性能都没明显下降。然后我觉得是偶然的时候，发现了一个这周和上周的差异，这周是测试100次取平均值，上周是测试50次取平均值。然后我这周又测试了一遍测试50次取平均值，结果和上周一样。然后我仔细看了一遍每次运行的时间，发现conv3特别小，测试50次取平均值运行一次只需要0.15秒左右，测试100次取平均值需要0.12秒左右。还需要多次测试不同情况。