week16实验记录

zxp

January 6, 2024

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，多增加了几个循环顺序

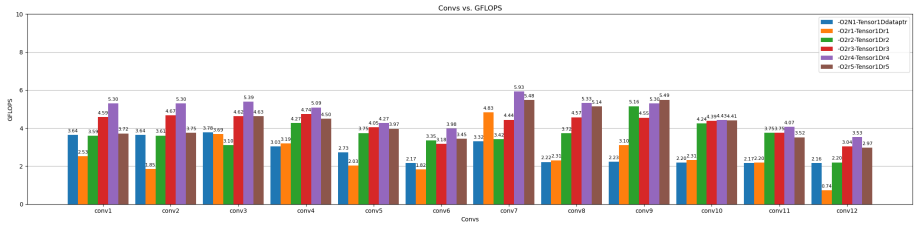

Figure 1: O2

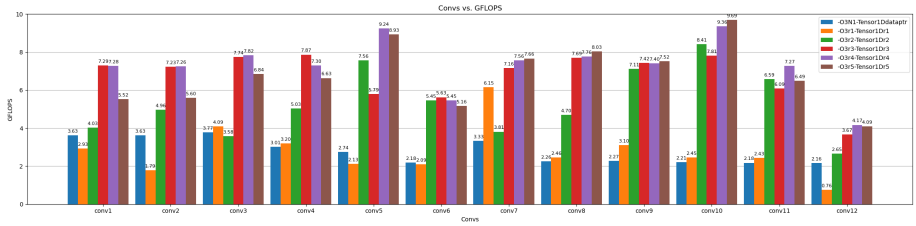

Figure 2: O3

3 Experiment

这周是实验目的是找出conv和循环顺序的规律，找出合适的循环顺序。前几周虽然得出改变循环顺序会使直接卷积性能发生改变的结论。但由于数据结构有三个（tensor1d、tensor4d和wetensor）循环顺序有四种导致画出的图不好用，于是这周我统一使用tensor_1d并且为了更好的找规律多测了俩个循环顺序。O0不明显，只放上比较明显的O2，O3

3.1 每个conv

Table 1: Twelve convolution layers of the DNN benchmarks.

NAME	INPUT	FILTER, STRIDE	OUTPUT
	$C_i \times H_i \times W_i$	$C_o \times H_f \times W_f, s_h(s_w)$	$C_o \times H_o \times W_o$
Conv1	$3 \times 227 \times 227$	$96 \times 11 \times 11, 4$	$96 \times 55 \times 55$
Conv2	$3 \times 231 \times 231$	$96 \times 11 \times 11, 4$	$96 \times 56 \times 56$
Conv3	$3 \times 227 \times 227$	$64 \times 7 \times 7, 2$	$64 \times 111 \times 111$
Conv4	$64 \times 224 \times 224$	$64 \times 7 \times 7, 2$	$64 \times 109 \times 109$
Conv5	$96 \times 24 \times 24$	$256 \times 5 \times 5, 1$	$256 \times 20 \times 20$
Conv6	$256 \times 12 \times 12$	$512 \times 3 \times 3, 1$	$512 \times 10 \times 10$
Conv7	$3 \times 224 \times 224$	$64 \times 3 \times 3, 1$	$64 \times 222 \times 222$
Conv8	$64 \times 112 \times 112$	$128 \times 3 \times 3, 1$	$128 \times 110 \times 110$
Conv9	$64 \times 56 \times 56$	$64 \times 3 \times 3, 1$	$64 \times 54 \times 54$
Conv10	$128 \times 28 \times 28$	$128 \times 3 \times 3, 1$	$128 \times 26 \times 26$
Conv11	$256 \times 14 \times 14$	$256 \times 3 \times 3, 1$	$256 \times 12 \times 12$
Conv12	$512 \times 7 \times 7$	$512 \times 3 \times 3, 1$	$512 \times 5 \times 5$

3.2 各个顺序

第一种，最初始的

```
for (i < output.batch)
  for (j < output.channel)
   for (m < output.height)
    for (n < output.width)
      for (r < input.channel)
        for (u < fiter.height)
          for (v < fiter.width)
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

第二种，论文《高性能零内存直接卷积中》的循环顺序，图中r1

```
for (i < output.batch)
  for (m < output.height)
    for (u < fiter.height)
      for (v < fiter.width)
        for (r < input.channel)
          for (n < output.width)
            for (j < output.channel)
```

```
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

第三种，图中r2

```
for (r < input.channel)
  for (u < fiter.height)
    for (v < fiter.width)
      for (i < output.batch)
        for (j < output.channel)
          for (m < output.height)
            for (n < output.width)
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

第四种，图中r3

```
for (i < output.batch)
  for (r < input.channel)
    for (j < output.channel)
      for (m < output.height)
        for (u < fiter.height)
          for (v < fiter.width)
            for (n < output.width)
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

第五种，图中r4

```
for (i < output.batch)
  for (j < output.channel)
    for (m < output.height)
      for (r < input.channel)
        for (u < fiter.height)
          for (v < fiter.width)
            for (n < output.width)
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

第六种，图中r5

```
for (i < output.batch)
  for (j < output.channel)
    for (r < input.channel)
      for (u < fiter.height)
        for (v < fiter.width)
          for (m < output.height)
            for (n < output.width)
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

3.3 Analysis

3.3.1 Conv3

步长为2 表现最好的是循环顺序是第五种

```
for (i < output.batch)  //32
  for (j < output.channel)  //64
    for (m < output.height)  //111
      for (r < input.channel)  //3
        for (u < fiter.height)  //7
          for (v < fiter.width)  //7
            for (n < output.width) // 111
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

表现最差的是第三种

```
for (r < input.channel)  //3
  for (u < fiter.height)  //7
    for (v < fiter.width)  //7
      for (i < output.batch)  //32
        for (j < output.channel)  //64
          for (m < output.height)  //111
            for (n < output.width)  //111
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

表现同样不这么好的第六种

```
for (i < output.batch)  //32
  for (j < output.channel)  //64
    for (r < input.channel)  //3
      for (u < fiter.height)  //7
        for (v < fiter.width)  //7
          for (m < output.height)  //111
            for (n < output.width) //111
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

64x111肯定是要比7x7大的，而且tensor_1d的数据结构连续性排序比较好的依次是（width，height，width）
conv3的特点是output的宽和高特别大。conv3表现比较特殊的地方是，表现最差的是第三种循环顺序。其他表现好的循环顺序（第四五种）并没有特别大的差异。
但实际上因为步长的存在，input的数据宽高连续性是很差的，而且input的长宽和output的长宽一致都是靠m和n这两层循环来历遍，而且这两层数据很大。最差的第三种和表现不那么好的第六种都是output的宽和高在内俩层。而且，其他步长不为1的conv（conv1-4）在这两种循环顺序（第三第六种）都均表现

不好，只是没con3这么极端（表现最差的都还是把连续性不如宽的高放在最内层）。

3.3.2 Conv5

表现最差的是第二种，因为连续性不如宽的高放在最内层。表现最好的是循环顺序是第五种

```
for (i < output.batch)  //2
  for (j < output.channel)  //256
    for (m < output.height)  //20
      for (r < input.channel)  //96
        for (u < fiter.height)  //5
          for (v < fiter.width)  //5
            for (n < output.width) //20
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

其他conv都表现不错的但conv5表现不好的第四种

```
for (i < output.batch)  //2
  for (r < input.channel)  //96
    for (j < output.channel)  //256
      for (m < output.height)  //20
        for (u < fiter.height)  //5
          for (v < fiter.width)  //5
            for (n < output.width)  //20
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

表现较好，但其他conv上表现一般的第三种

```
for (r < input.channel)  //96
  for (u < fiter.height)  //5
    for (v < fiter.width)  //5
      for (i < output.batch)  //2
        for (j < output.channel)  //256
          for (m < output.height)   //20
            for (n < output.width)  //20
output(i,j,m,n)+=input(i,r,m*s+u,n*s+v)*fiter(j,r,u,v)
```

conv5特殊的地方是input的宽高特别小。最差和最好的内三层都是一样的，第五种循环顺序在所有的conv上表现都不错（在这六种不是最好的也是第二好的），第五种循环input的channel（也等于fiter的channel）比第四种里面，这个值要比output的height大，因为conv5的intput和output的宽高比较小，所以第四种循环表现不好吧（同样是intput和output的宽高比较小的conv6第四种是表现最好的，但conv6全部的循环顺序表现都很差，可能是conv6实在太小

了）。
conv5在第三种循环顺序表现相比其他conv比较好，conv6在第三种循环顺序下表现也不错，应该也是intput和output的宽高比较小，但channel比较大，把channel放在比较里面的原因。

3.3.3 Conv9-11

虽然前几周认为第四种循环顺序在conv9和11上表现不好，可能是花的图比较混乱的原因，现在图中只有一个数据结构加上多了几种循环顺序，再看看conv9和conv11，表现和其他的conv就很接近，没差别多少。

3.3.4 结果

我觉得没必要就单纯的循环顺序而言，把比较大的output的hight放在最内层性能就会比把相对output比较小的fiter的hight放在最内层好，数据充足数据连续性好一些，其他的那几层循环影响在大部分conv上是比较细微的。
我认为没必要就纯干找循环顺序不同的一点细微变化，现在看的那些论文中的循环顺序还要考虑其他优化影响，比如分块和SIMD，我想先去把其他优化试一下然后回头再看循环顺序。