week8实验记录

zxp

November 11, 2023

1 environment

cpu:Inter i5-12400f (2.5 GHz)

System:Ubuntu 22.04.1

Compiler:x86_64-linux-gun-gcc-11

2 Experiment

This week,we use the twelve specified convolution,use Tensor1D and Tensor4D and weTensor as the data structures. The code has been added with some parts based on last week(part of wetensor and remove directConvolution to main.cpp).
code：https://github.com/SakakibaraMako/Benchmark/v2

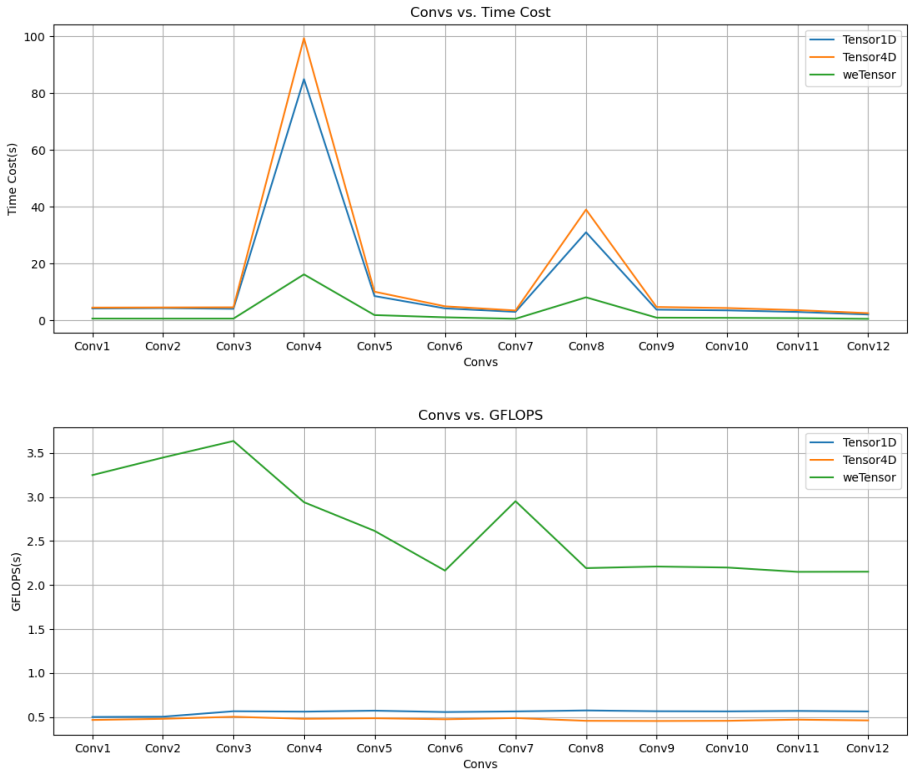

Figure 1: Compilation Optimization option: -O2

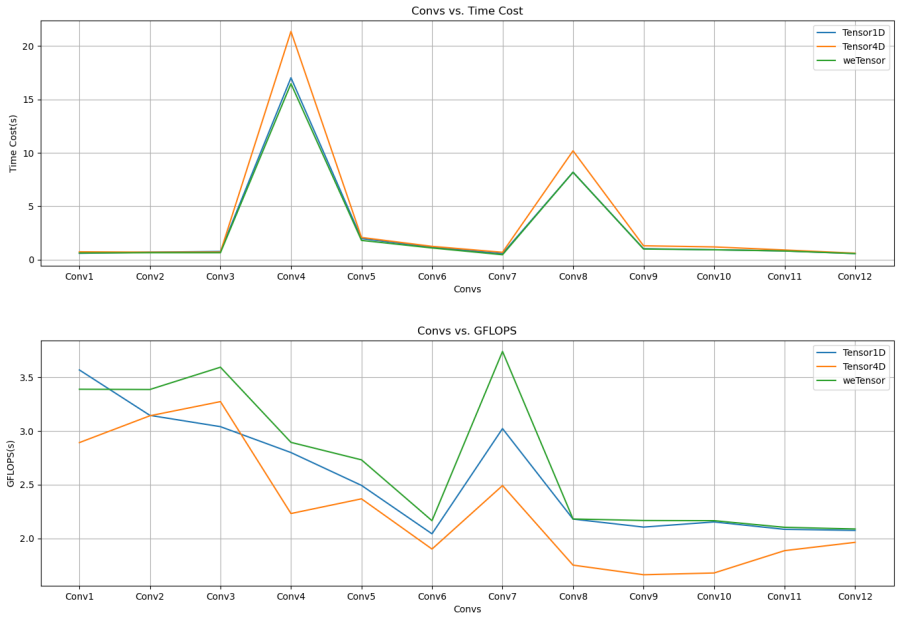

Figure 2: Compilation Optimization option: -o3

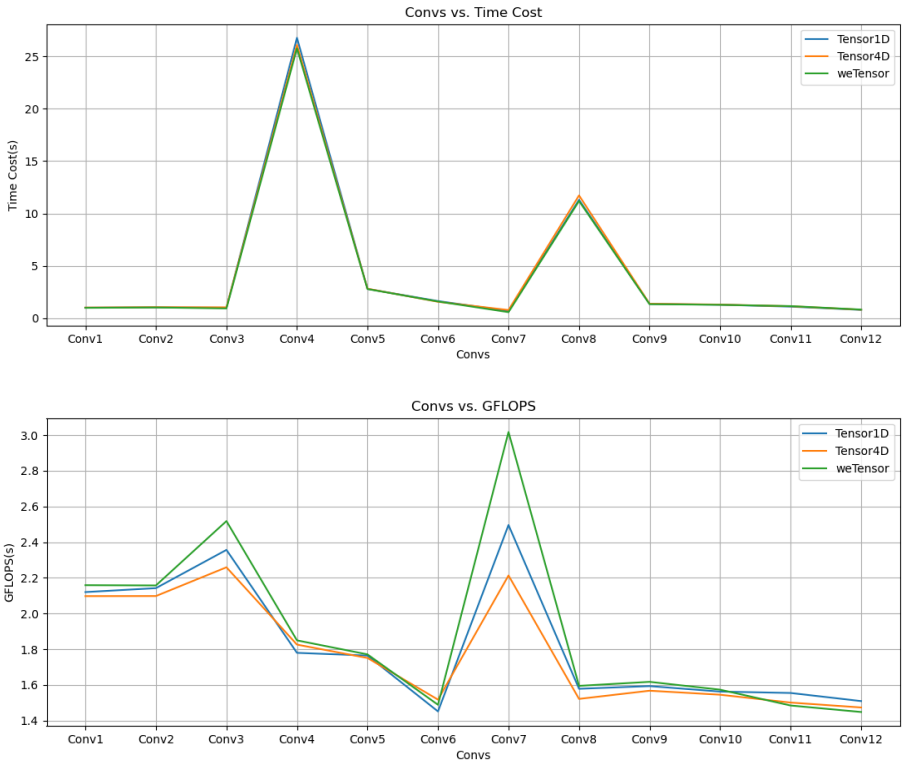

Figure 3: Compilation Optimization option: -Ofast -march=native

2.1 Analysis

under -O2、-O3 and -Ofast -march=native,wetensor is better than Tensor1D and Tensor4D.Last week,The calculation method of gflogs is error,the gflogs of conv 4 and conv 7 is higher.Maybe,the reason is the batch of conv 4 and conv 7 is smaller.

2.2 Conclusion

weTensor is better than Tensor1D and Tensor4D.

3 Experiment2

Test the impact of different optimizations on direct convolution.

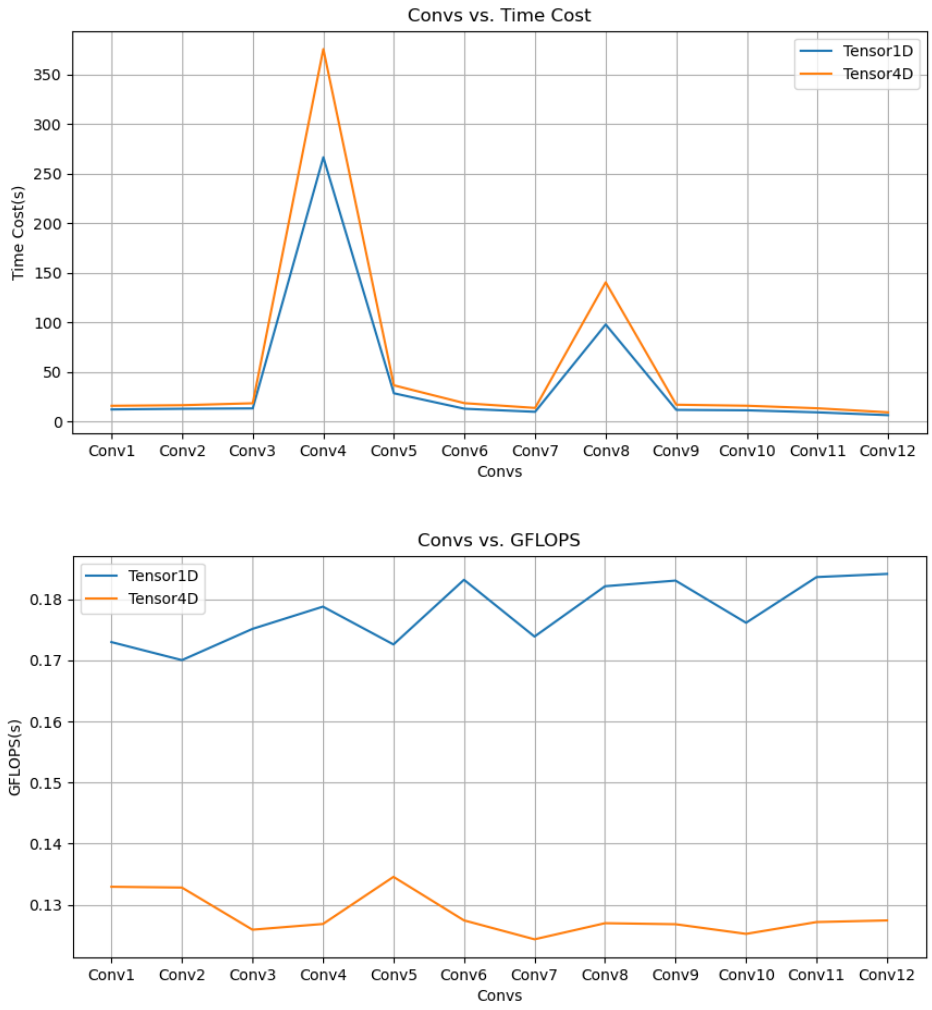

Figure 4: Compilation Optimization option: -O0

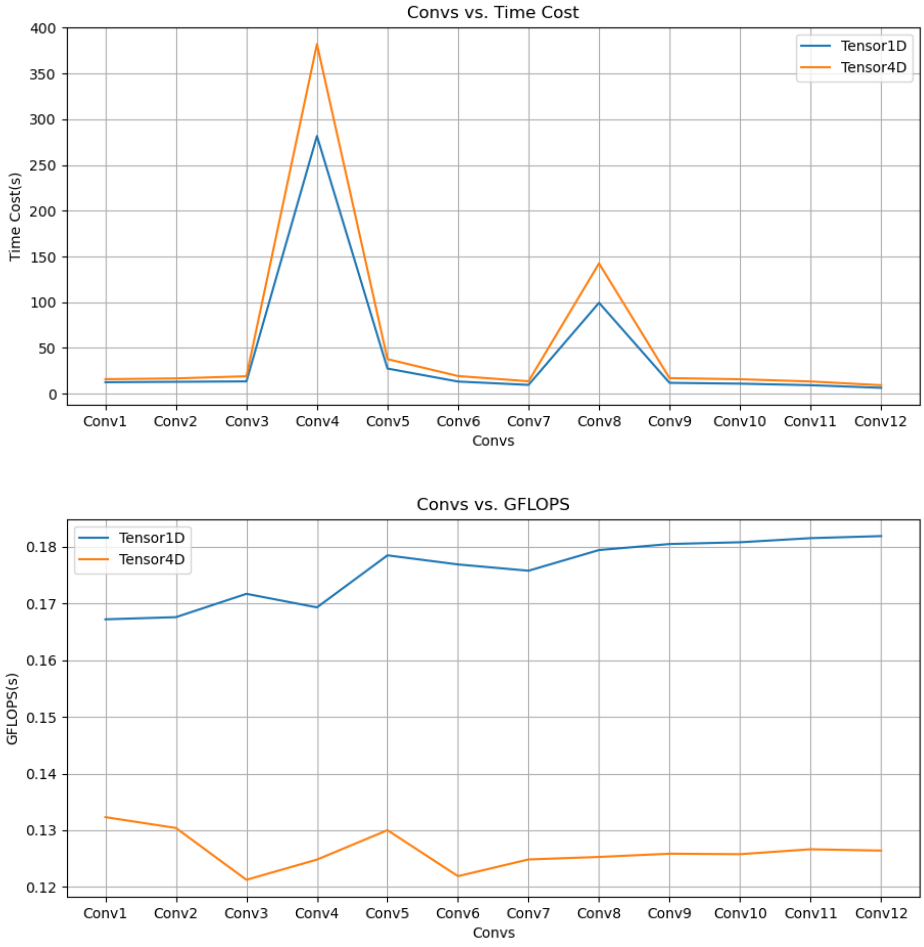

Figure 5: Compilation Optimization option: -O0 -fauto-inc-dec

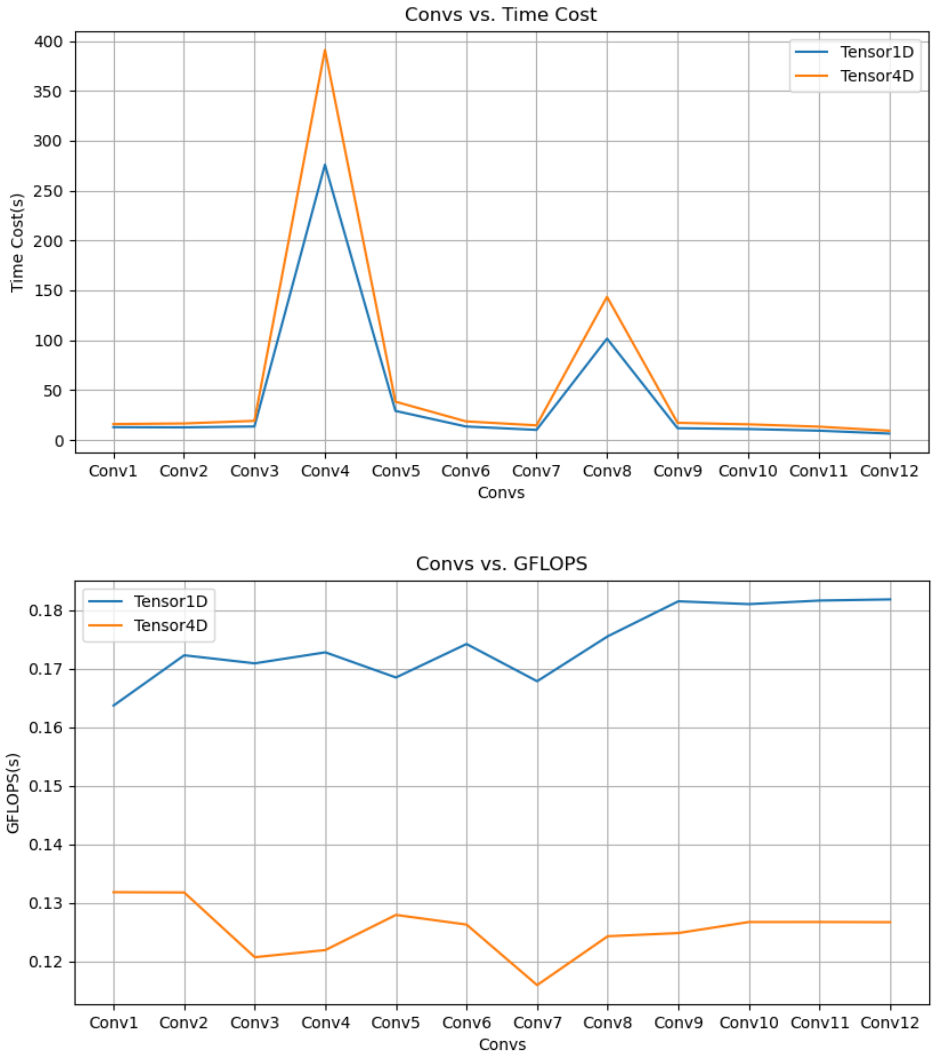

Figure 6: Compilation Optimization option: -O0 -fbranch-count-reg

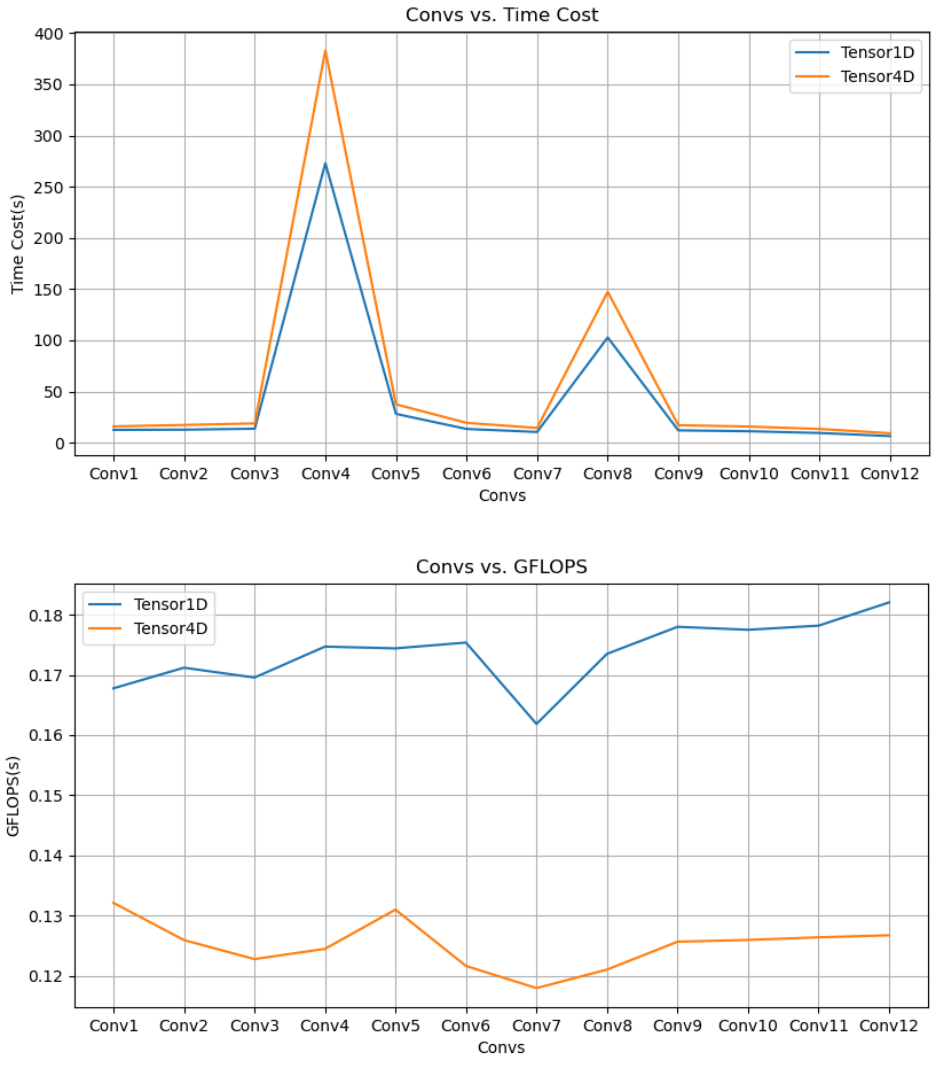

Figure 7: Compilation Optimization option: -O0 -fcombine-stack-adjustments

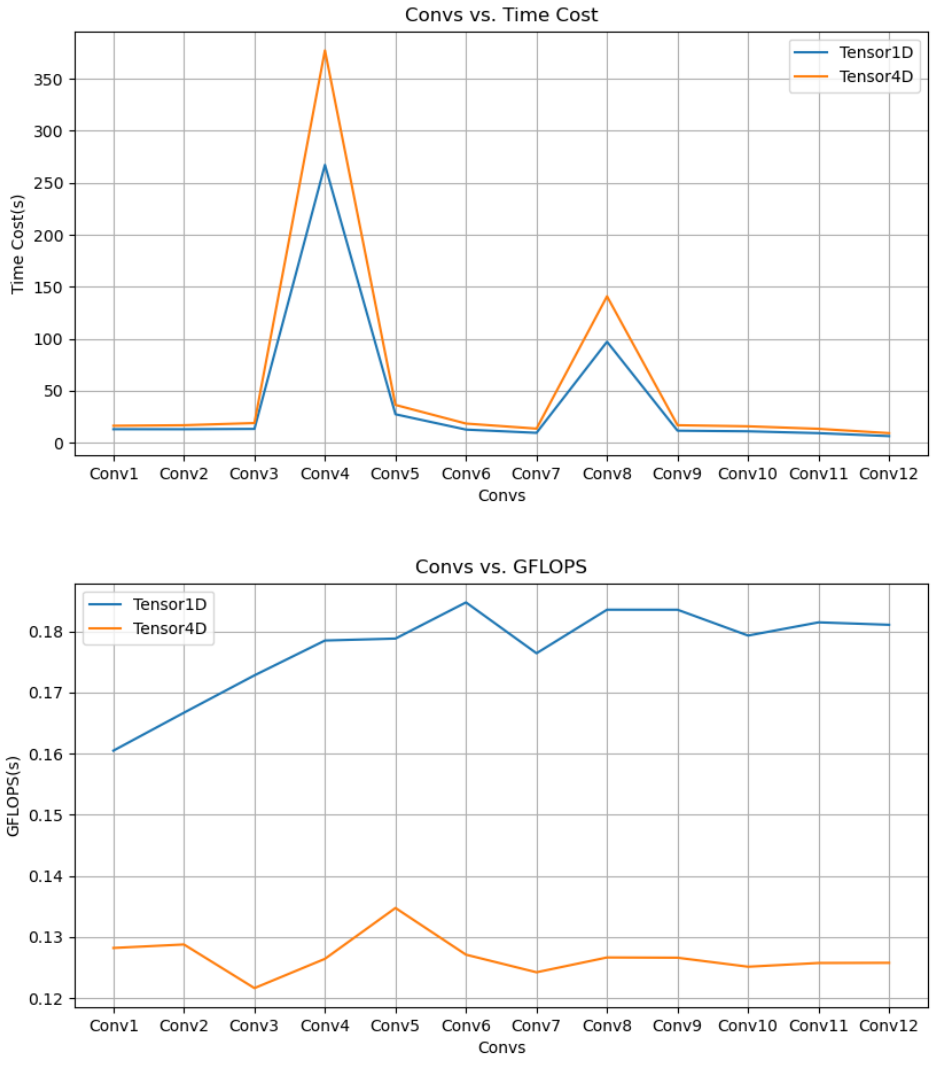

Figure 8: Compilation Optimization option: -O0 -fcompare-elim

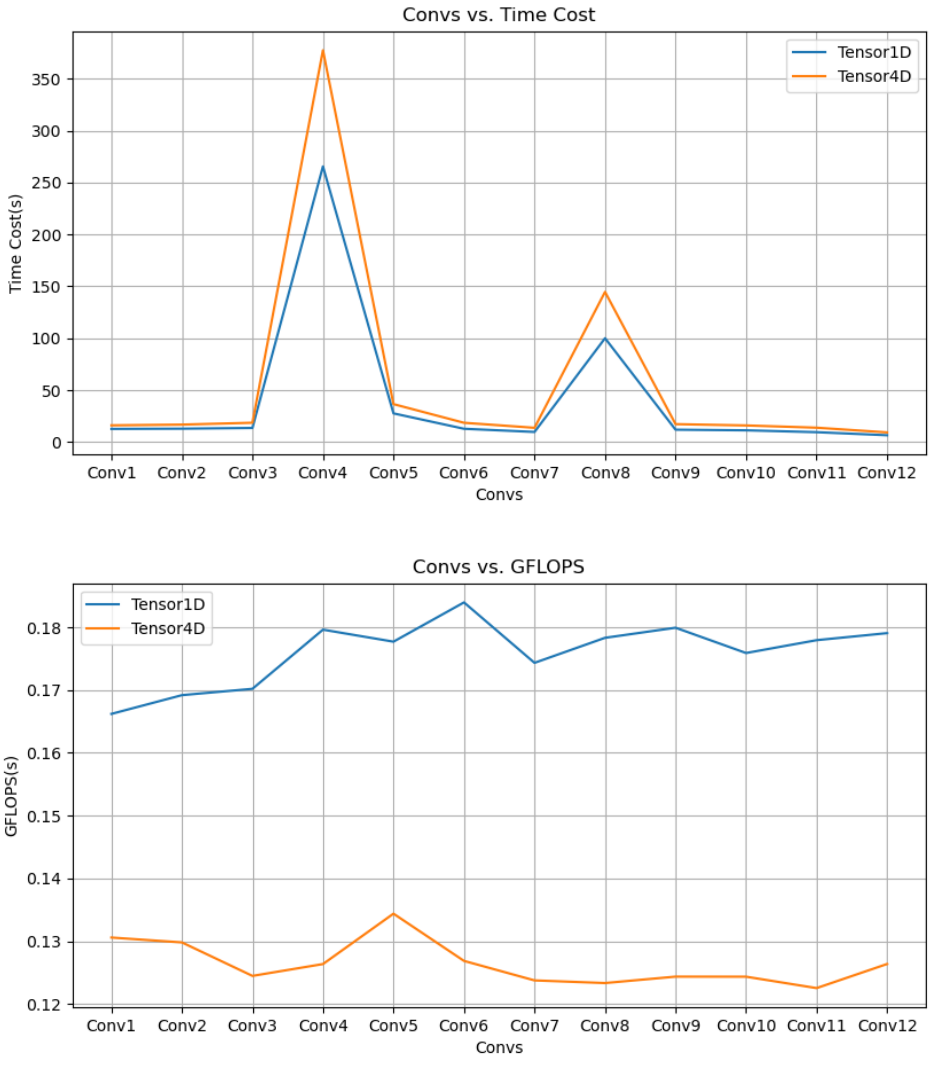

Figure 9: Compilation Optimization option: -O0 -fcprop-registers

3.1 Analysis

-fauto-inc-dec
生成auto-inc/dec指令
-fbranch-count-reg
Replace add, compare, branch with branch on count register.用计数寄存器上的分支代替添加、比较、分支。
-fcombine-stack-adjustments
Looks for opportunities to reduce stack adjustments and stack references.寻找机会减少堆栈调整和堆栈引用。
-fcompare-elim
Perform comparison elimination after register allocation has finished.在寄存器分配完成后执行比较消除。
-fcprop-registers
Perform a register copy-propagation optimization pass.执行寄存器复制传播优化步骤。
These 5 have little impact on direct convolution.

4 Experiment3

这周为了测试wetensor在直接卷积的表现，我在main.cpp文件实现了对wetensor的直接卷积，但上周实现对tensor1d和tensor4d的直接卷积是在tensor1d和tensor4d类里面，于是我把对tensor1d和tensor4d的直接卷积函数放在了mian.cpp。然后问题来了，在-O2级别的优化下这周最慢的conv花了100s，而上周这个conv只花了20多秒！上周-O2优化的耗时和-O3差距不算特别大，但这周差距巨大。
我怀疑是偶然，于是又按上周的方式跑了一遍，效果和上周差不多。于是我开始找原因。
对比了这周和上周代码的不同，首先我怀疑是循环计数的数据结构（上周是size_t这周开始用了int64_t）和循环是否用大括号括起来，测试后发现这俩并不会对直接卷积产生影响。
然后，尝试在直接卷积的时候使用Tensor1D和Tensor4D而不是Tensor，结果快了一点，但效果没上周的好。
然后把T改成double，没效果
改成在直接卷积函数的内部申请输出张量的空间，没效果
然后想试试把tensor1d和tensor4d里面的直接卷积函数的申请输出张量的空间改成在外面申请，没影响
然后想不出这周和上周的代码有什么差异了，就开始测-O3比-O2多开的优化选项
目前测了3个(-fgcse-after-reload—-fipa-cp-clone—-floop-interchange)，运气很好，测到一个优化选项开启后效果很好。
-fipa-cp-clone
Perform cloning to make Interprocedural constant propagation stronger.执行克隆以使程序间常量传播更强。
但是有个问题，不开O2优化，直接开这个优化没什么效果

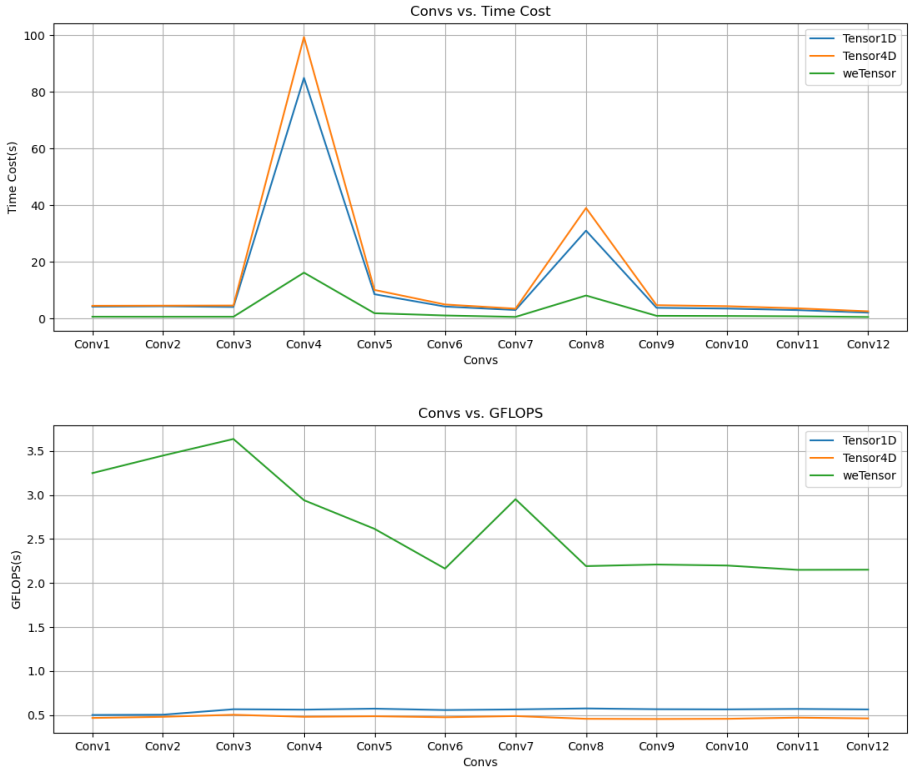

Figure 10: 这周开启-O2优化的效果

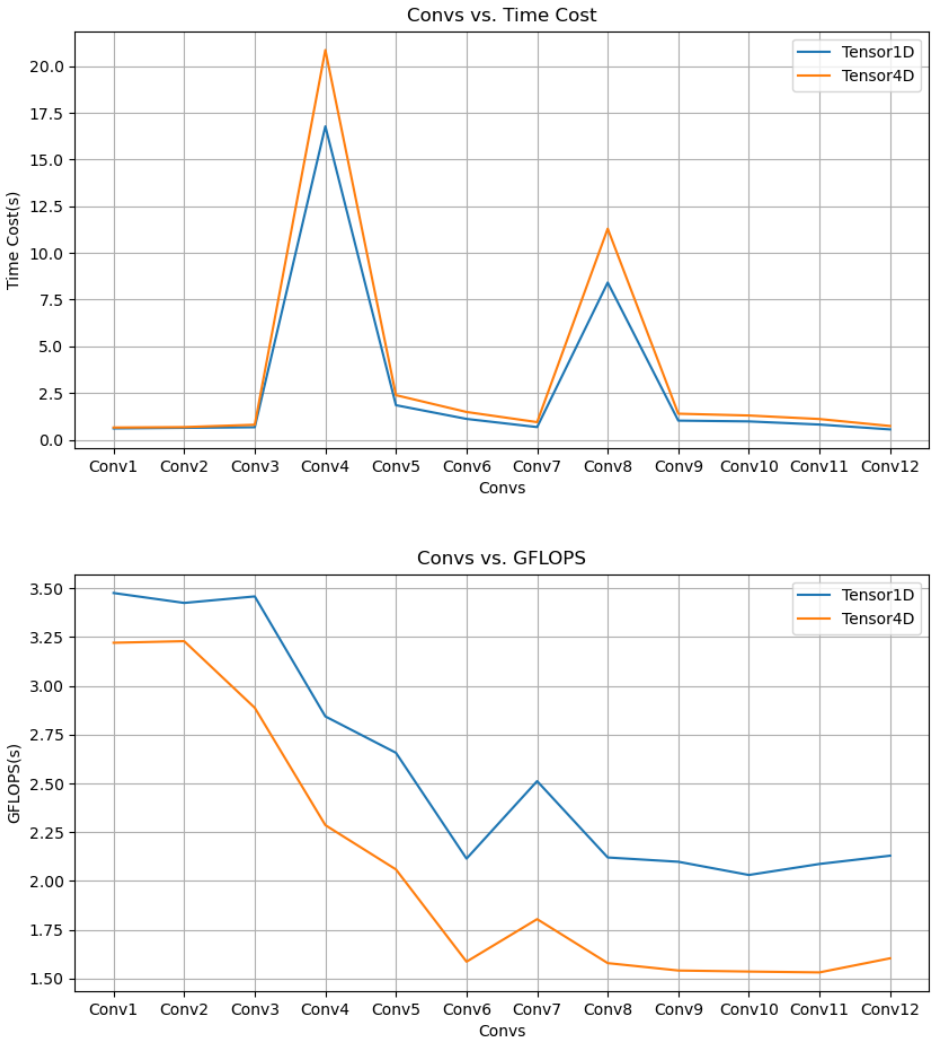

Figure 11: 上周开启-O2优化的效果

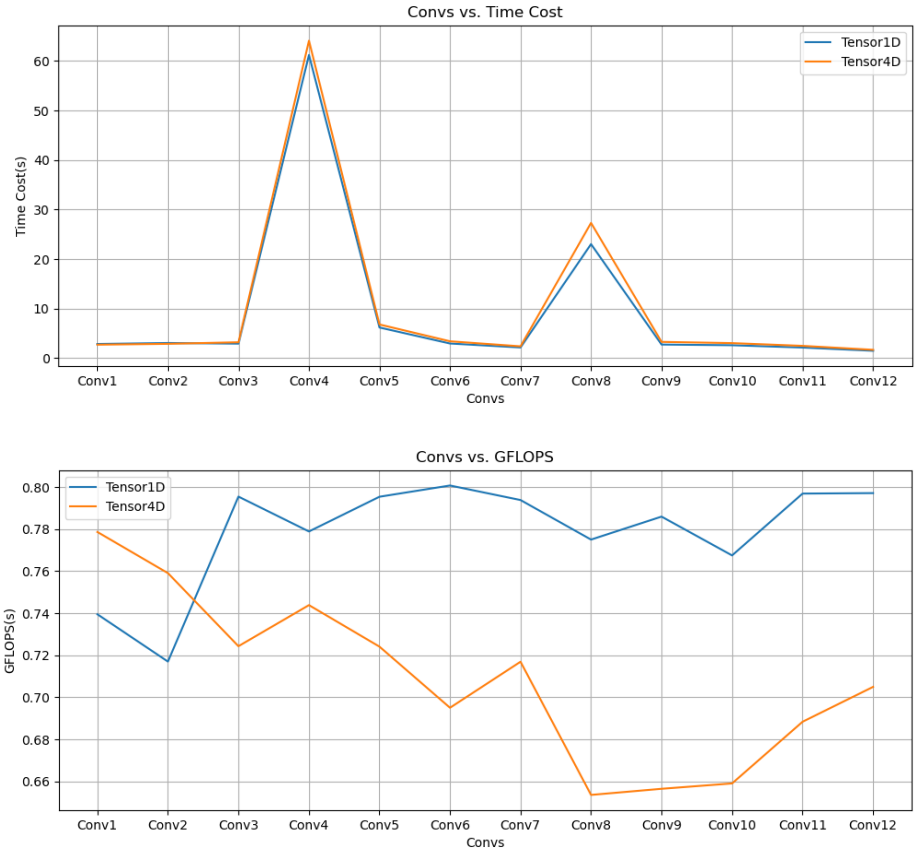

Figure 12: 直接卷积的时候使用Tensor1D和Tensor4D而不是Tensor

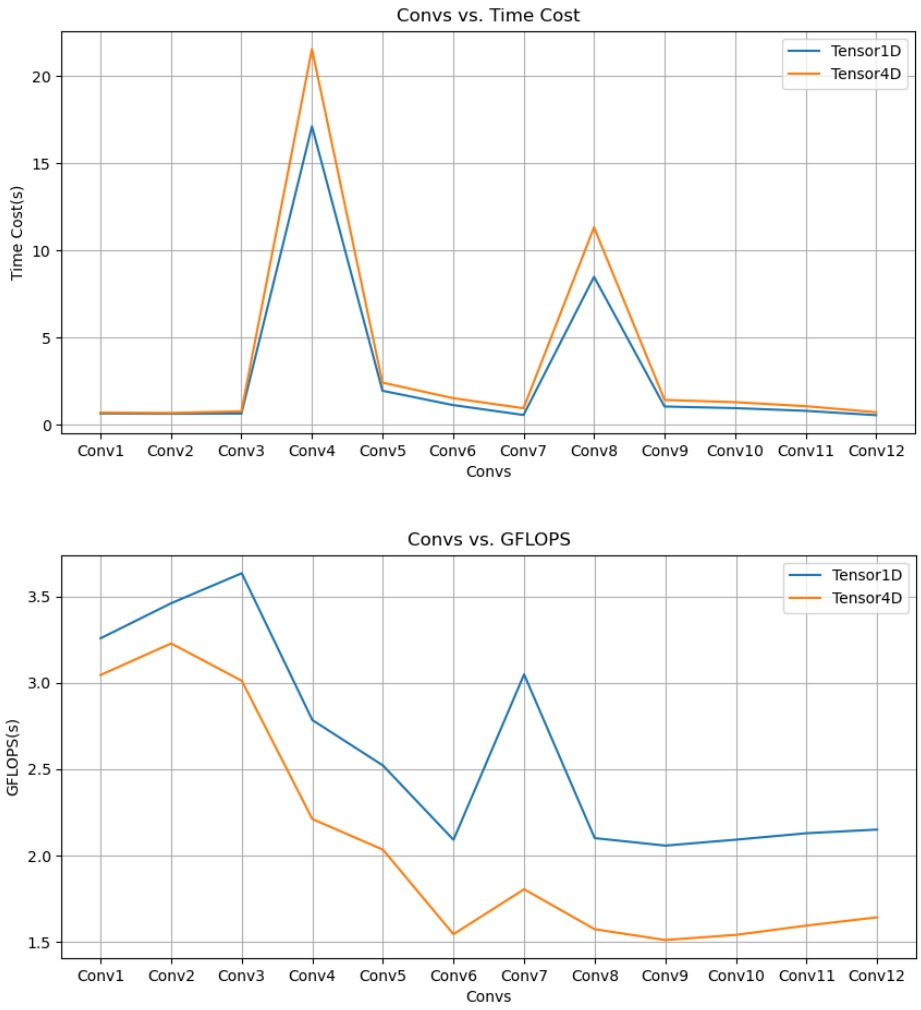

Figure 13: 开启-O2-fipa-cp-clone

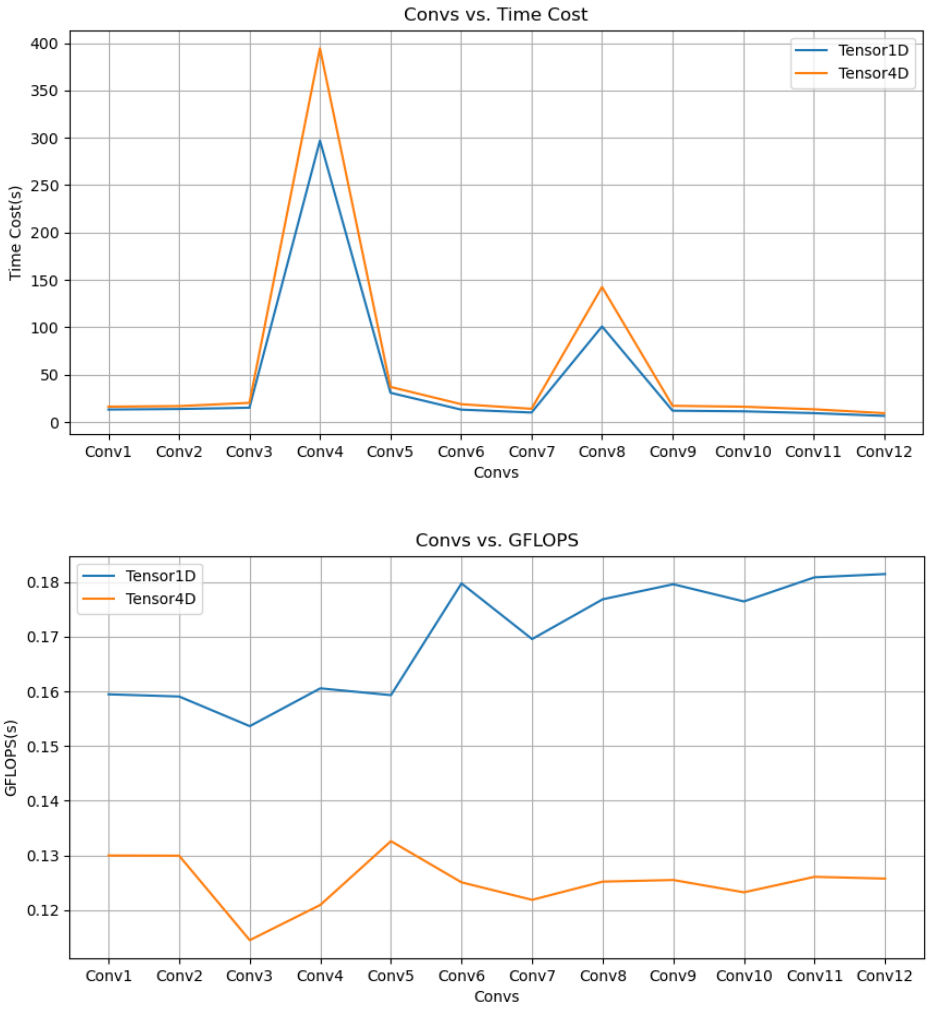

Figure 14: 开启-O0-fipa-cp-clone