week9实验记录

zxp

November 18, 2023

1 environment

cpu:Inter i5-12400f (2.5 GHz)

System:Ubuntu 22.04.1

Compiler:x86_64-linux-gun-gcc-11

2 Experiment

重新绘制O2、O3、Ofast的图片，用了统一的y轴方便比较

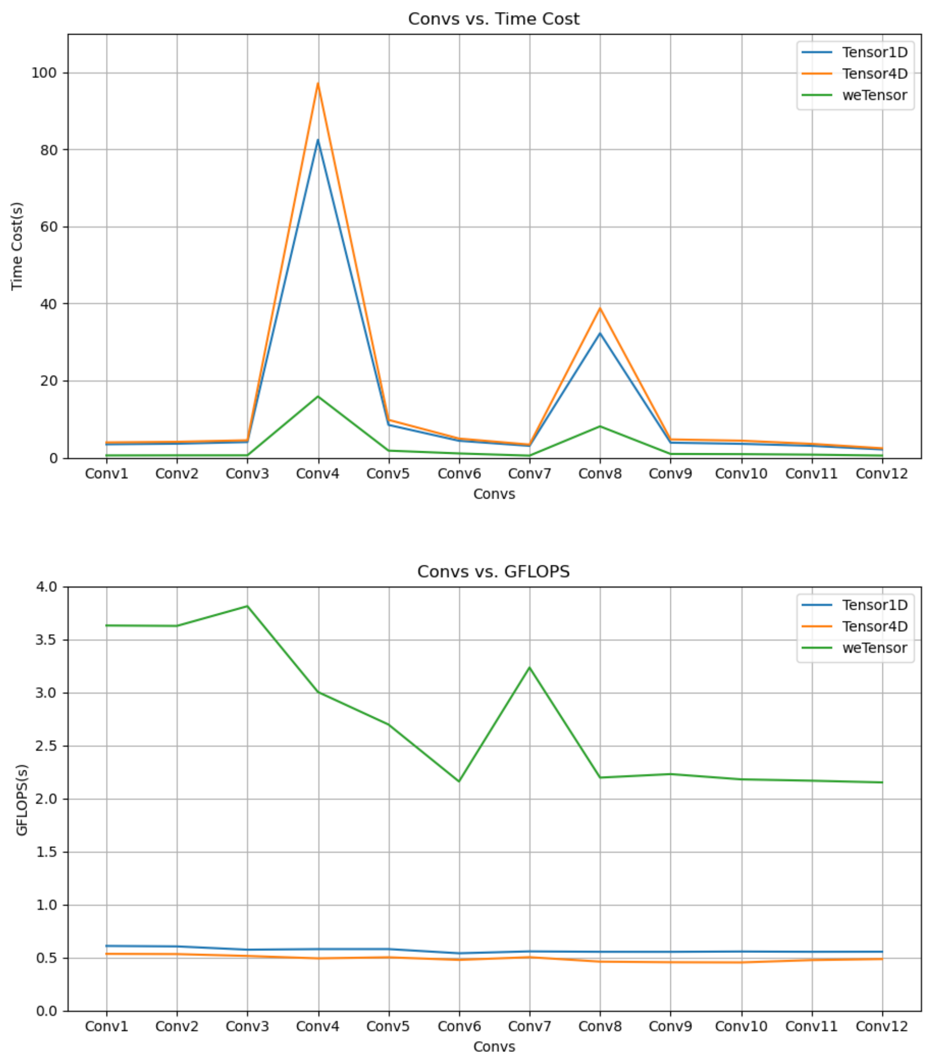


Figure 1: Compilation Optimization option: -O2

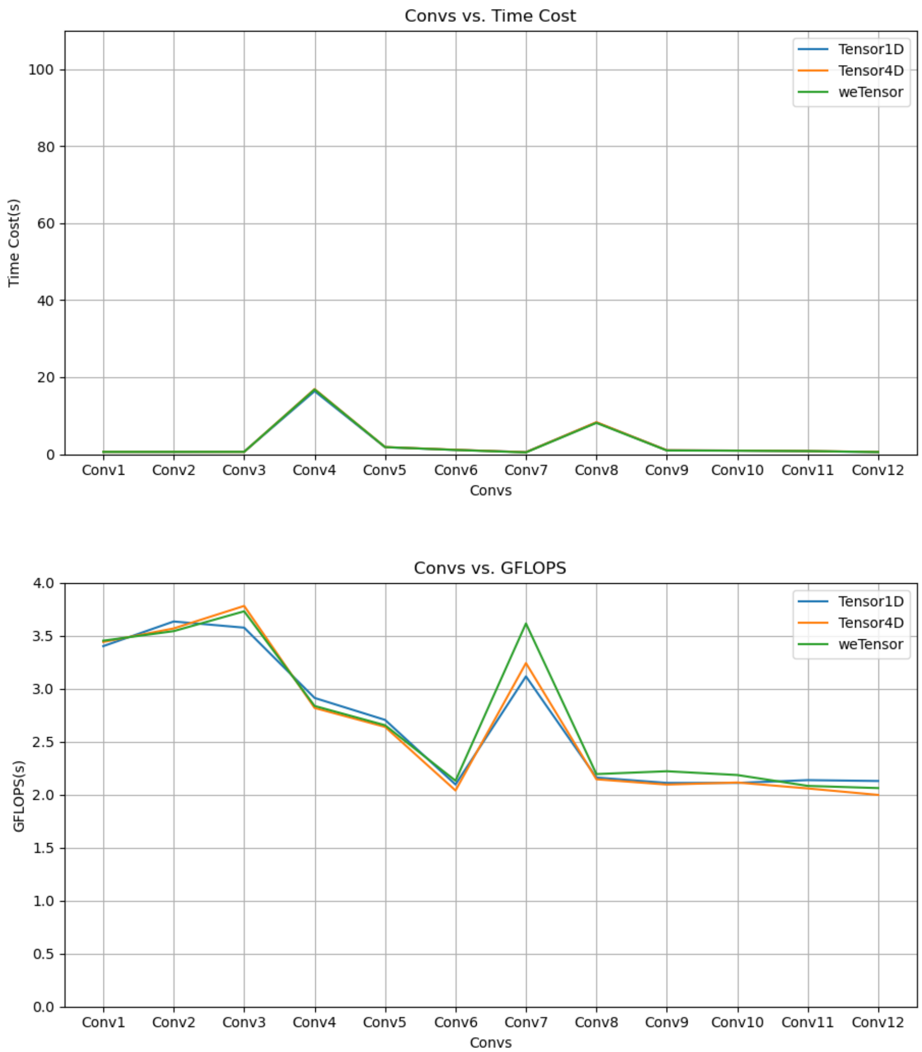


Figure 2: Compilation Optimization option: -o3

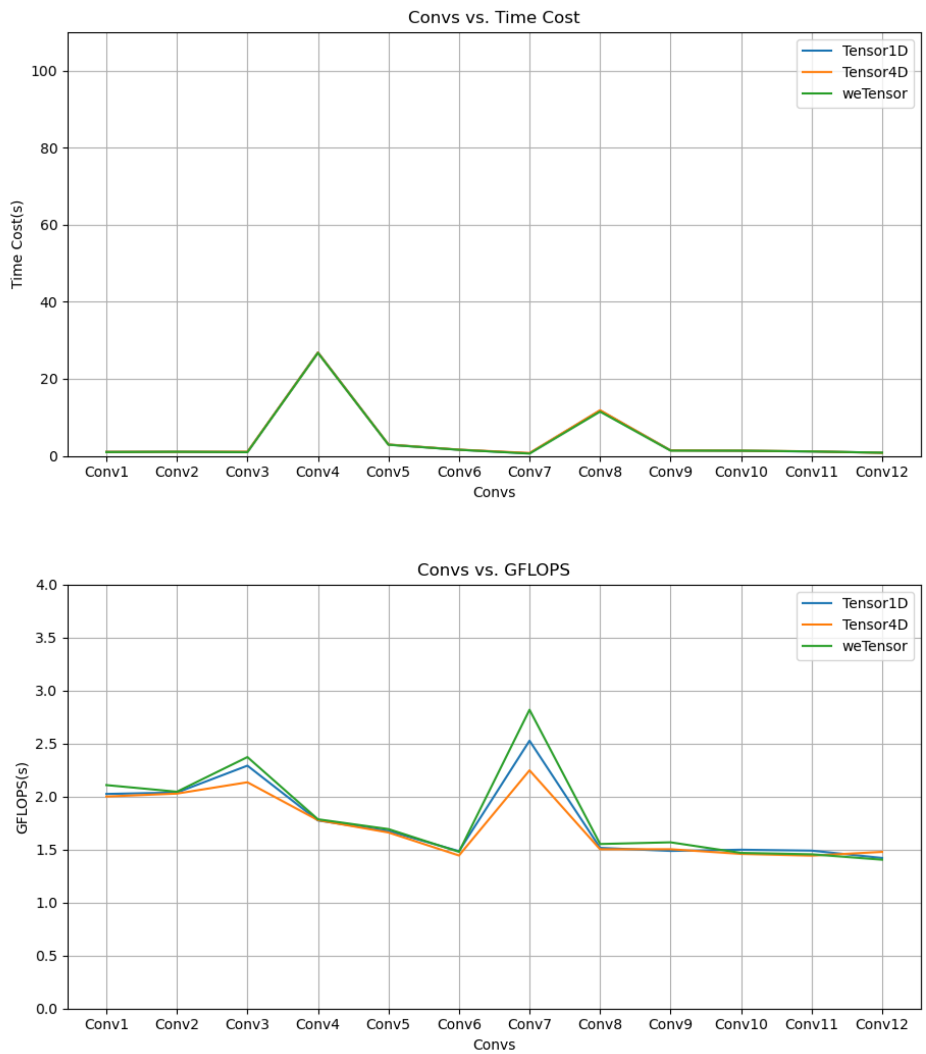


Figure 3: Compilation Optimization option: -Ofast -march=native

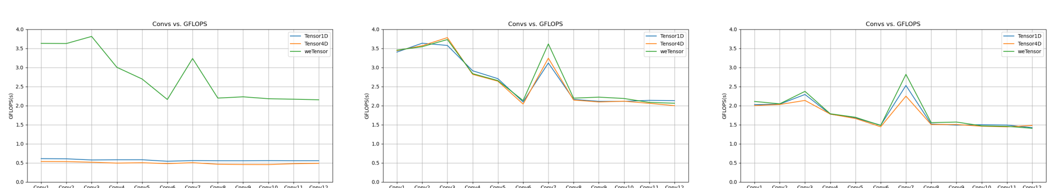


Figure 4: 三张图放在一起比较

2.1 Analysis

-O3大幅度优化了tensor1d和tensor4d在直接卷积的表现，在-O3优化下tensor1d和tensor4d十分接近wetensor，甚至在Conv2、3、4、11、12下tensor1d或tensor4d小幅度超越wetensor。
然而，在-Ofast -march=native的优化下，tensor1d、tensor4d和wetensor直接卷积反而比O3花费了更多时间。

3 Experiment2

Test the impact of different optimizations on direct convolution.

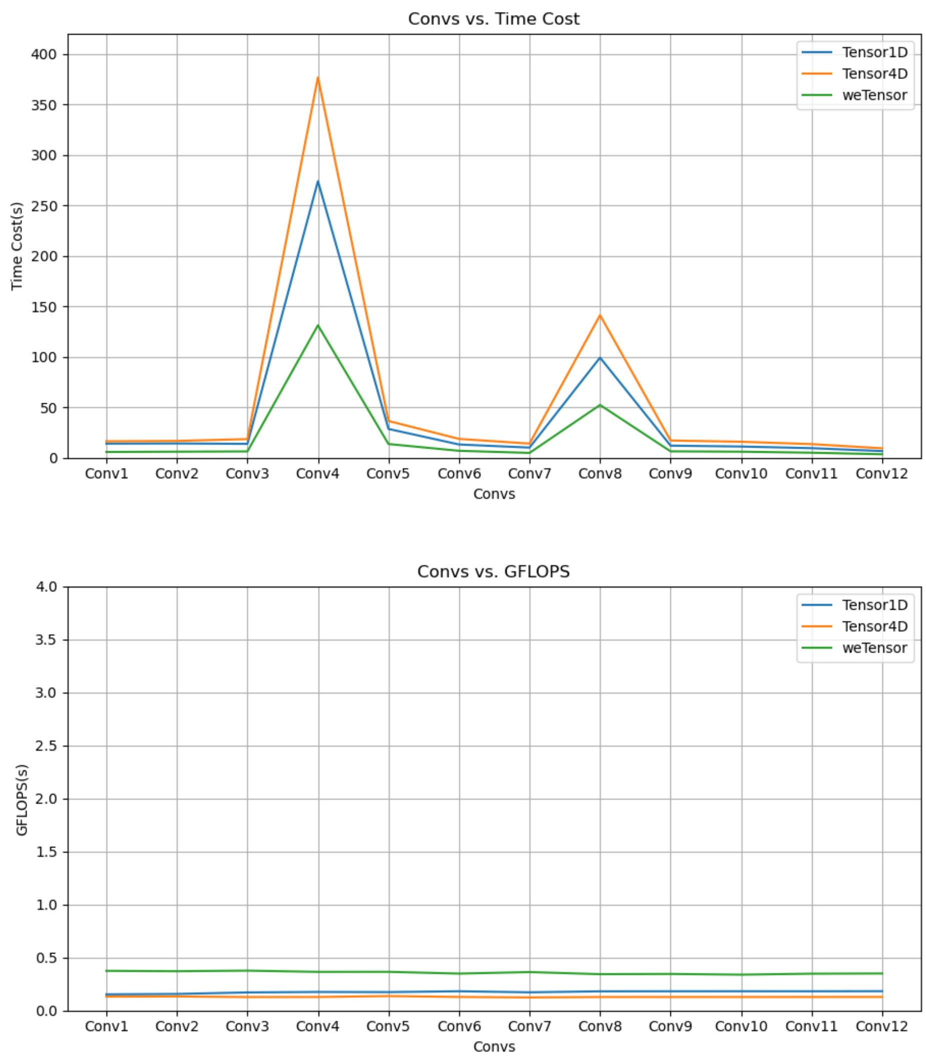
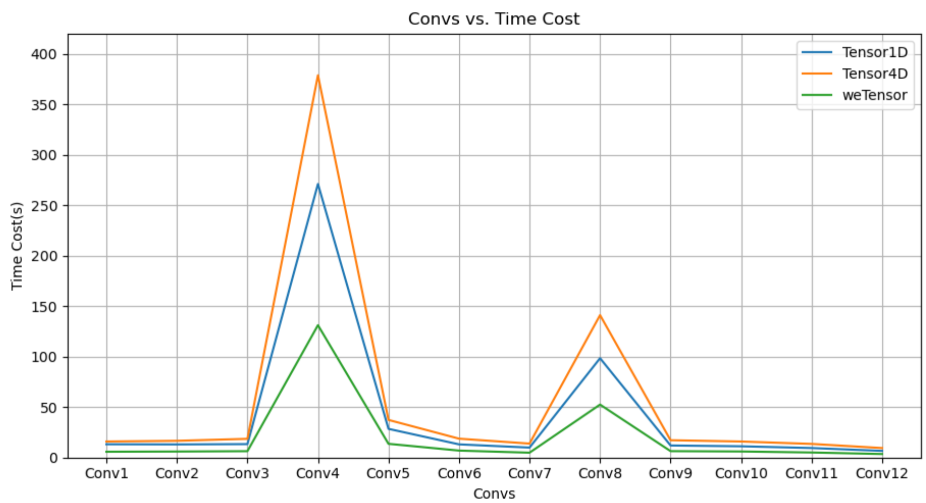


Figure 5: Compilation Optimization option: -O0 -fipa-profile



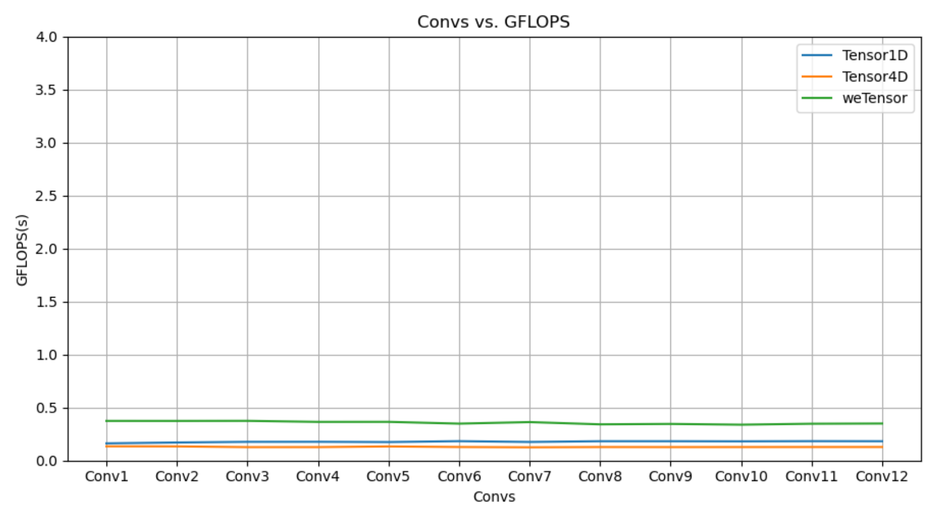
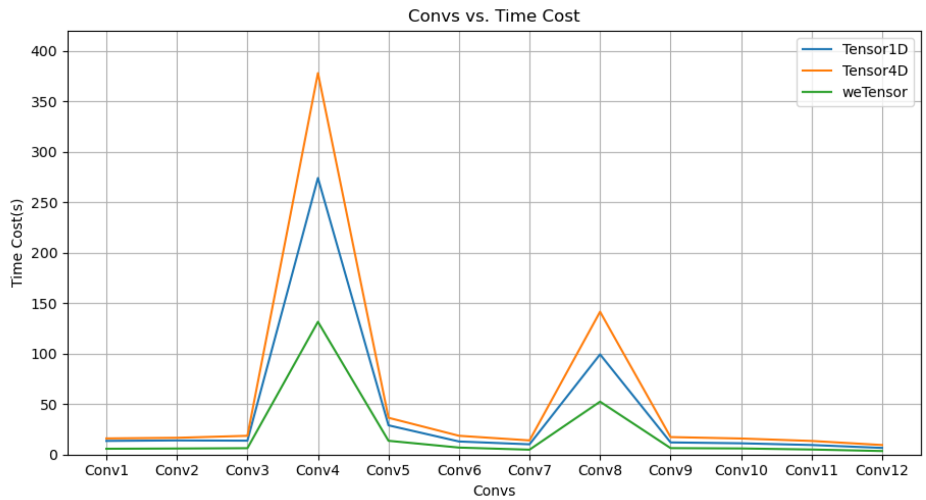


Figure 6: Compilation Optimization option: -O0 -fipa-pure-const



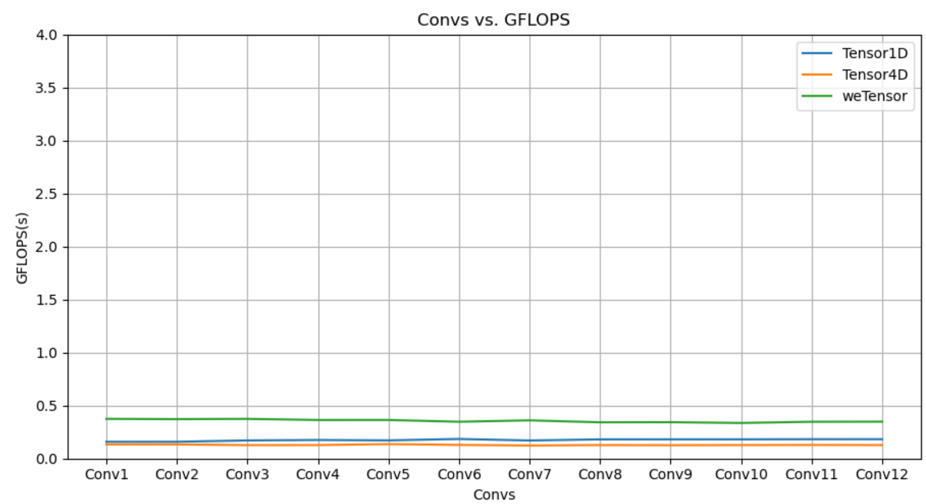


Figure 7: Compilation Optimization option: -O0 -fipa-reference-addressable

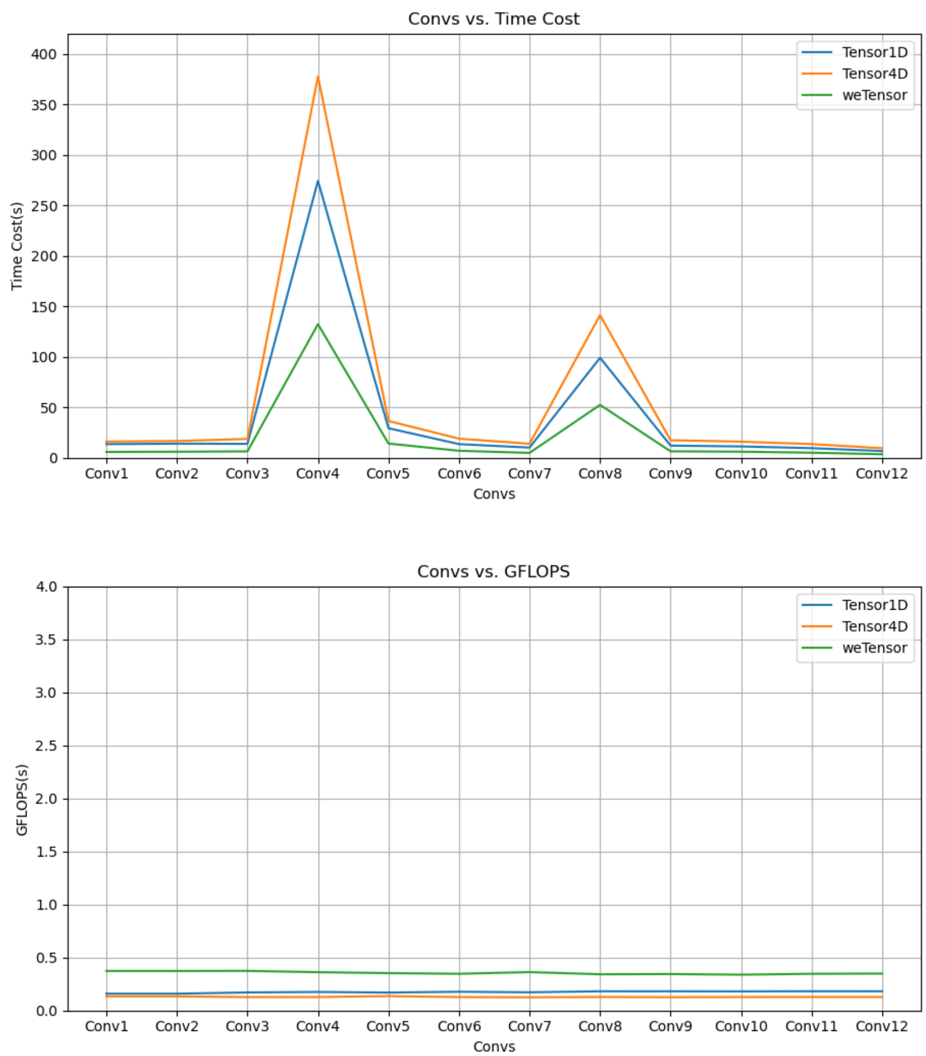
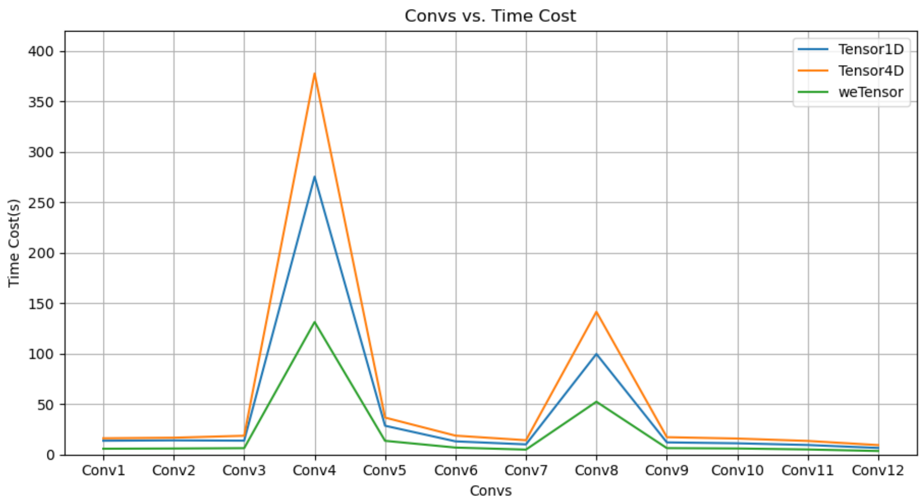


Figure 8: Compilation Optimization option: -O0 -fmerge-constants



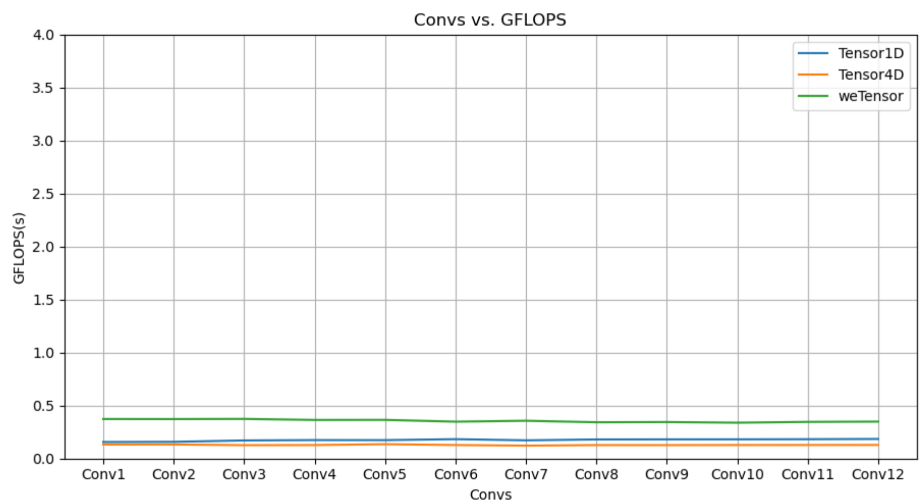


Figure 9: Compilation Optimization option: -O0 -fipa-pure-const

3.1 Analysis

These 5 have little impact on direct convolution.

4 Experiment3

roofline

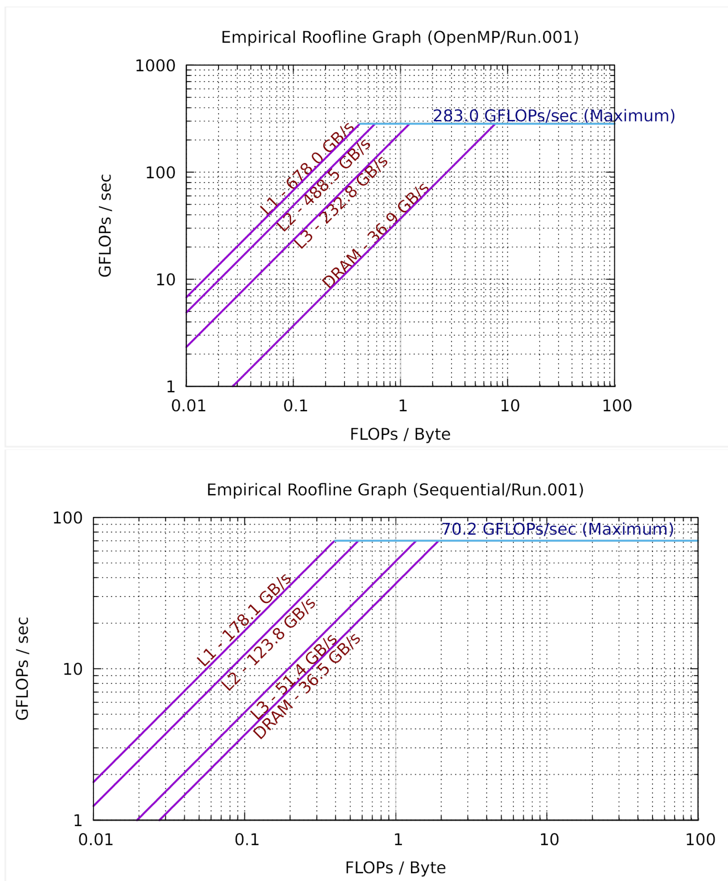


Figure 10: docker中绘制的

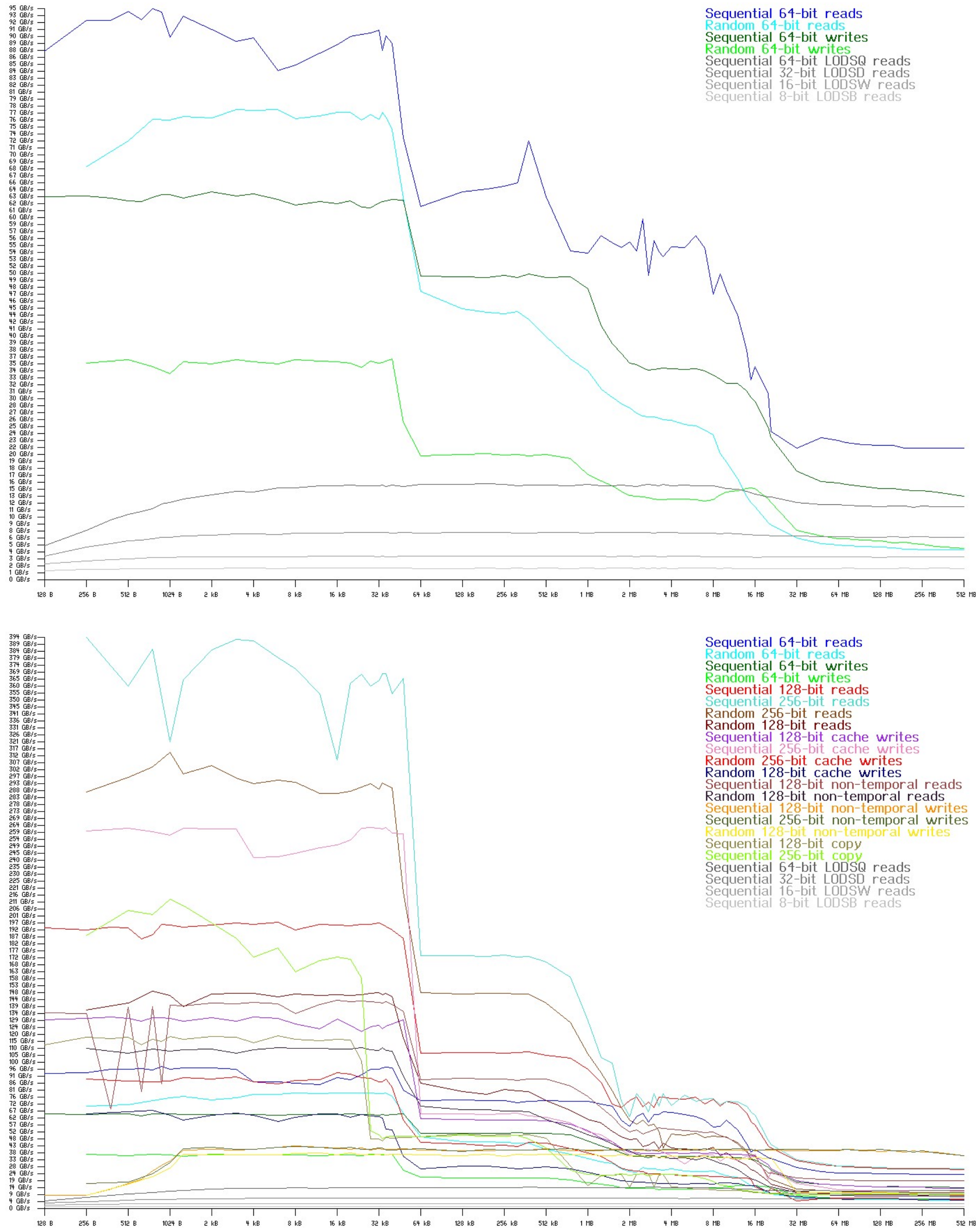


Figure 11: docker中绘制的

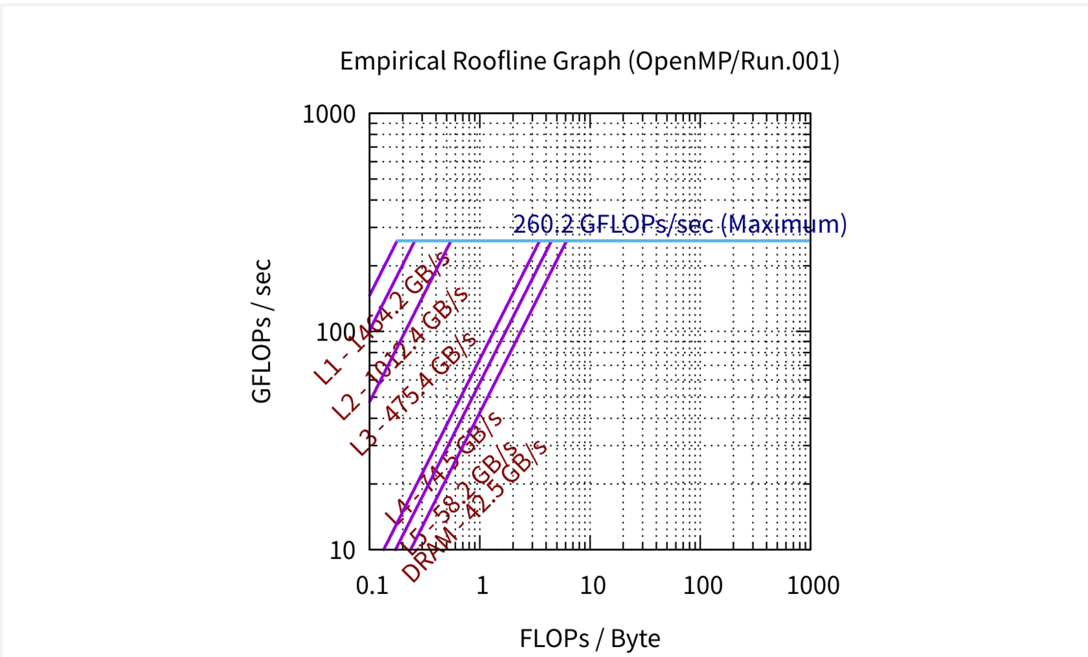


Figure 12: 直接绘制的

5 Experiment4

这周我测测试-O3比-O2多开的优化选项。尝试在O2优化的基础上一个个单独打开某个选项的效果，并且测试打开其他选项而不开该选项的效果。
O3比O2多开了16个选项。（具体数据见data.xlsx。折线图见figure文件夹）

1	O2	tensor1d-times:	3.46679	3.61929	4.0473	82.4708	8.49585	4.3772	3.057
2		tensor4d-times:	3.95206	4.11375	4.51243	97.1272	9.80776	4.95215	3.392
3	O3	tensor1d-times:	0.619723	0.601354	0.64825	16.3731	1.81675	1.12595	0.546
4		tensor4d-times:	0.612588	0.612413	0.613059	16.923	1.86218	1.157	0.525
5	-O2 -fgcse-after-reload	tensor1d-times:	3.4663	3.59709	4.04236	85.1644	8.55483	4.25432	3.036
6		tensor4d-times:	3.95285	4.09863	4.79943	97.3069	9.99623	4.90247	3.380
7	-O2 -fipa-cp-clone	tensor1d-times:	0.593494	0.619275	0.682872	16.4655	1.88597	1.12621	0.566
8		tensor4d-times:	0.7046	0.818112	0.766289	20.742	2.37913	1.47	0.878
9	-O2 -floop-interchange	tensor1d-times:	3.67828	3.74929	4.18258	83.4273	8.57173	4.33707	3.047
10		tensor4d-times:	4.14144	4.29216	4.62219	99.8363	9.74534	4.93331	3.440
11	-O2 -floop-unroll-and-jam	tensor1d-times:	3.51829	3.61445	4.21359	86.442	8.62653	4.47378	3.170
12		tensor4d-times:	4.16786	4.34306	4.67076	100.509	9.94717	4.957	3.445
13	-O2 -fpeel-loops	tensor1d-times:	3.47662	3.62063	4.05642	86.9827	8.81011	4.56376	3.209
14		tensor4d-times:	3.96028	4.11935	4.51666	101.258	10.3363	5.24944	3.439
15	-O2 -fpredictive-commoning	tensor1d-times:	3.47833	3.60687	4.06334	82.8534	8.48836	4.23963	3.041
16		tensor4d-times:	3.9774	4.11256	4.53322	99.366	9.85236	4.96741	3.397
17	-O2 -fsplit-loops	tensor1d-times:	3.79621	3.96256	4.41691	90.5012	8.94876	4.56643	3.248
18		tensor4d-times:	3.96062	4.11794	4.49844	96.4461	9.71338	4.93076	3.329
19	-O2 -fsplit-paths	tensor1d-times:	3.34082	3.48611	3.87353	80.084	7.87781	3.97886	2.887
20		tensor4d-times:	3.96319	4.10801	4.53143	96.8992	9.77629	4.91671	3.393
21	-O2 -ftree-loop-distribution	tensor1d-times:	3.47922	3.62848	4.03843	82.5841	8.45551	4.28688	3.051
22		tensor4d-times:	3.97612	4.11073	4.50272	97.0073	9.78734	4.95111	3.393
23	-O2 -ftree-loop-vectorize	tensor1d-times:	3.62125	3.75759	4.20283	83.3296	8.51064	4.31931	3.040
24		tensor4d-times:	4.12762	4.24636	4.73357	97.5197	10.0823	4.95895	3.399
25	-O2 -ftree-partial-pre	tensor1d-times:	3.32908	3.49269	3.86043	81.034	7.99165	3.91487	2.929
26		tensor4d-times:	3.98101	4.14321	4.47108	97.6925	9.93149	4.99115	3.295
27	-O2 -ftree-slp-vectorize	tensor1d-times:	3.77132	3.92034	4.32413	88.6368	8.83443	4.37532	3.157
28		tensor4d-times:	3.94852	4.09097	4.49547	96.3549	9.76309	4.94877	3.365
29	-O2 -funroll-completely-grow-si	tensor1d-times:	3.47049	3.59988	4.04809	82.5988	8.45777	4.37184	3.042
30		tensor4d-times:	3.95152	4.09815	4.50938	97.0846	9.82135	4.9396	3.395
31	-O2 -funswitch-loops	tensor1d-times:	3.45255	3.59824	3.83732	79.8728	7.84337	3.86727	2.827
32		tensor4d-times:	4.08929	4.22208	4.50682	96.9162	9.79671	4.92355	3.373
33	-O2 -fvect-cost-model=dynamic	tensor1d-times:	3.46834	3.61647	4.04625	82.533	8.4413	4.26797	3.053
34		tensor4d-times:	3.95424	4.10999	4.49094	96.9625	9.79128	4.921	3.394
35	-O2 -fversion-loops-for-strides	tensor1d-times:	3.46643	3.62047	4.02116	82.6121	8.48728	4.24863	3.054
36		tensor4d-times:	3.95273	4.09259	4.49219	97.0446	9.77464	4.93718	3.392

标红的是需要花费最长时间的conv，从图中可以看出，这16个选项中影响最大的是-fipa-cp-clone。可以看到开启这个优化后，性能已经相当接近O3

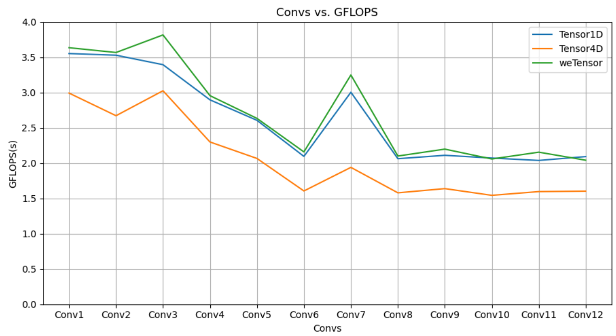


Figure 13: 开启-O2 -fipa-cp-clone优化

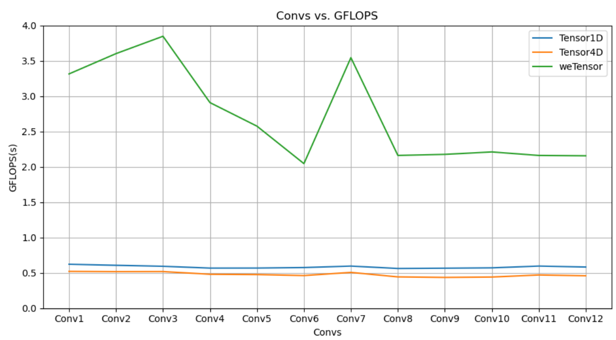
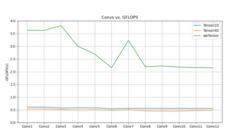
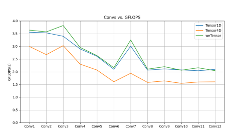
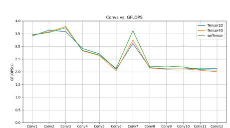


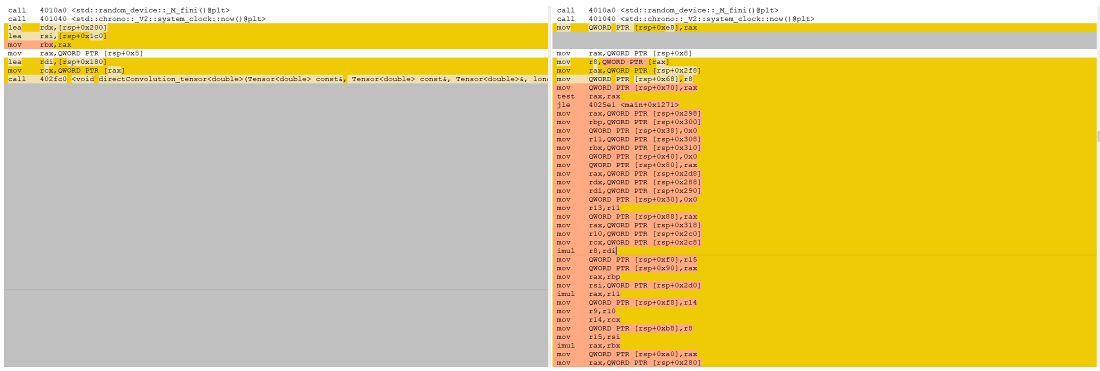
Figure 14: 开启-O3的其他选项但不开启-fipa-cp-clone优化







这个优化选择的官方解释是Perform cloning to make Interprocedural constant propagation stronger.执行克隆以使程序间常量传播更强。
看看这个优化选项做了什么。
左边为O2优化，右边为O2加上-fipa-cp-clone优化。开了这个优化后汇编码



从1479行涨到1679行。开了这个优化后最主要的不同是，这个优化把调用的直接卷积函数复制到了调用的位置。不开这个优化到函数调用的时候会执行call命令，跳到对应的函数部分，开了这个优化后会直接执行直接卷积。并且不是一模一样的复制，就长度而言tensor1d直接卷积部分要比tensor4d的长。

6 Experiment5

这部分是找为什么在-Ofast -march=native的优化下，tensor1d、tensor4d和wetensor直接卷积反而比O3花费了更多时间。先写结论，不是优化选项的问题，是后面的-march=native影响了性能。
Ofast的比O3多开了7个优化选项，但也关闭3个优化选项。我尝试在O3的基础上一个个把Ofast多开的优化选项打开和在-Ofast -march=native的基础上把Ofast比O3少开的优化选项打开，结果发现影响不大，是-march=native的问题（具体数据见data.xlsx。折线图见figure文件夹）
然后尝试在O3的基础上加上-march=native和直接用Ofast，发现开了-march=native后性能变差

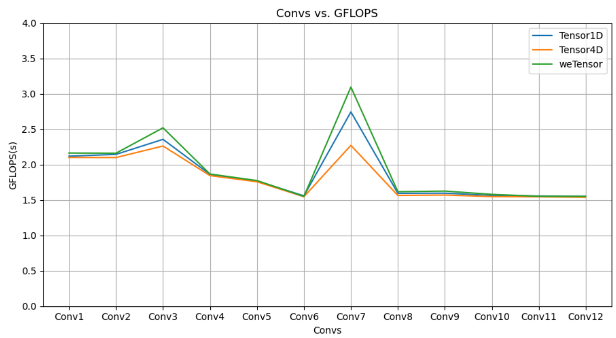


Figure 15: Compilation Optimization option: -O3 -march=native

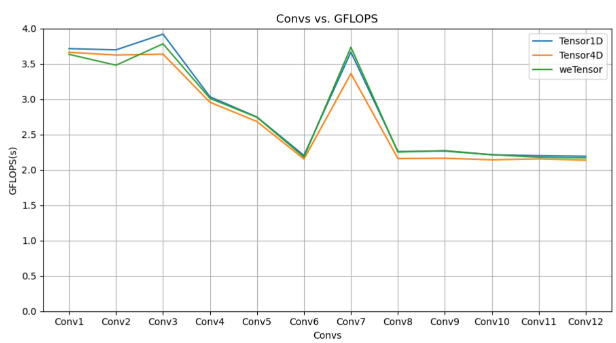


Figure 16: Compilation Optimization option: -Ofast