面向多模态大模型的基础卷积算子优化

基于国产加速卡曙光K100_AI进行优化

指导老师：汤善江
负责人：王鑫培、李世明
联系方式：wangxinpei@tju.edu.cn
项目起始时间：2024.09-2024.12

技术背景

卷积运算是深度学习中广泛应用的基础算子，但由于卷积计算量庞大，尤其在处理大规模数据时成为计算瓶颈。为了提高卷积运算的计算效率，目前有多种优化方法。

Implicit GEMM

- 将卷积过程映射为矩阵乘法。
- 利用矩阵乘法的优化技术实现加速。能够充分利用现代处理器（如GPU）上的矩阵运算优化。

方法1

Winograd算法

- Winograd算法是通过数学变换来优化卷积计算量实现加速。
- 特别适用于小卷积核（如3x3卷积核）的情况。

方法2

快速傅里叶变换（FFT）卷积

- 快速傅里叶变换（FFT）卷积是一种通过在频域计算卷积的优化方法。
- 适用于大卷积核和大数据集，能够显著减少计算复杂度。

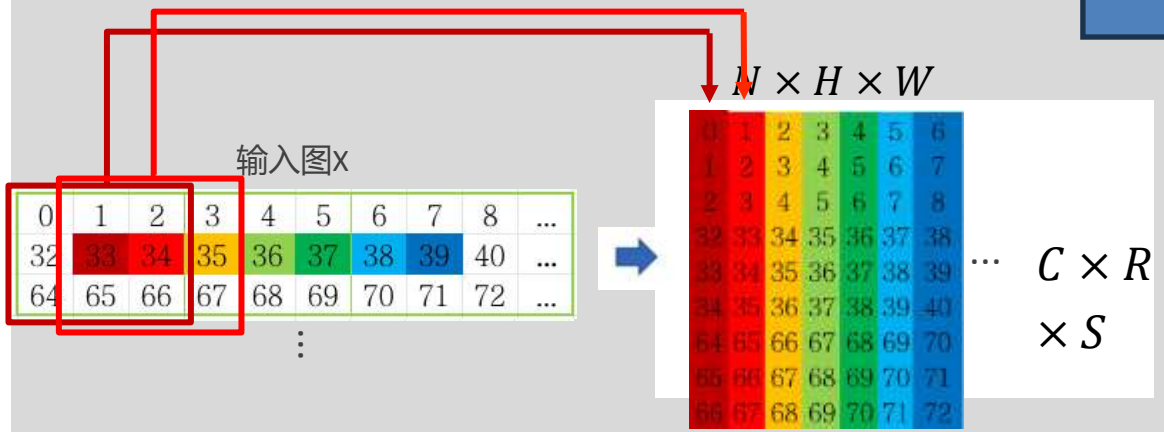
方法3

我们的工作：基于国产加速卡曙光K100_AI的硬件特点，使用方法一进行优化

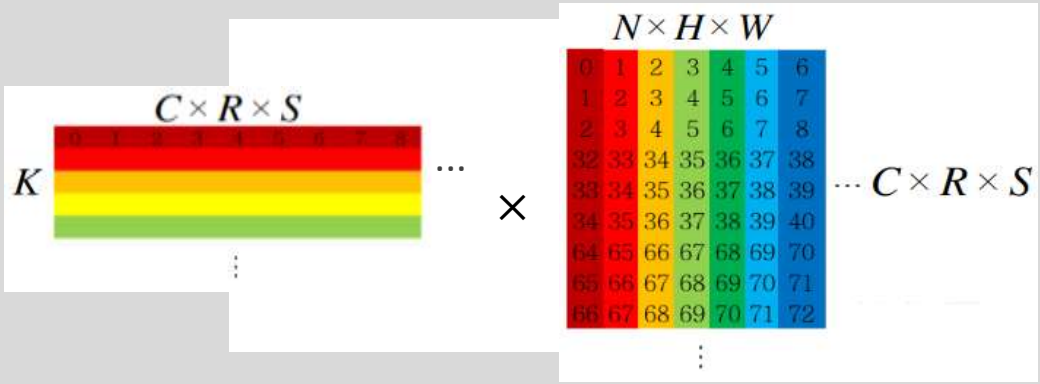
我们的工作和优化方法

一、卷积映射为矩阵乘法：Implicit GEMM算法

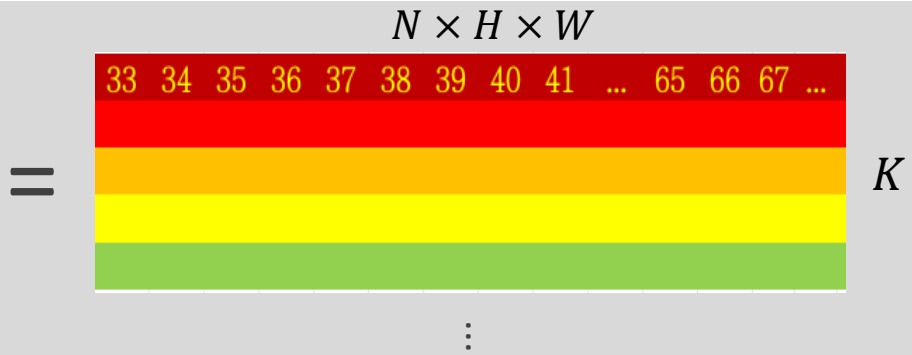
每个卷积核展开为一行（逻辑展开）



输入图的每个卷积滑动窗口展开为一列（逻辑展开）



卷积操作 -----> 矩阵乘法

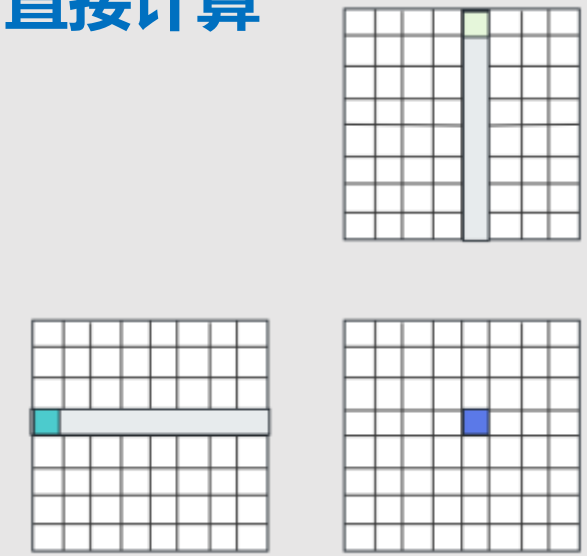


每行对应多张图的同一个输出通道

我们的工作和优化方法

二、提高计算访存比：矩阵分块，使用共享内存、寄存器来缓存和复用数据

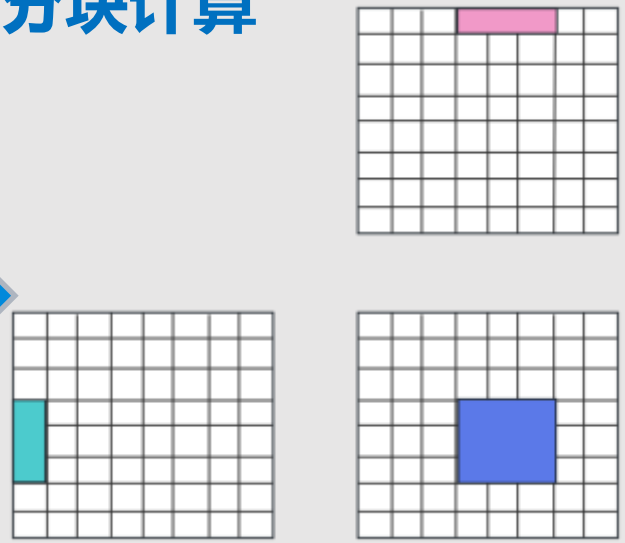
直接计算



每执行1次乘累加运算，需要进行2次访存

计算访存比：1/2 = 0.5

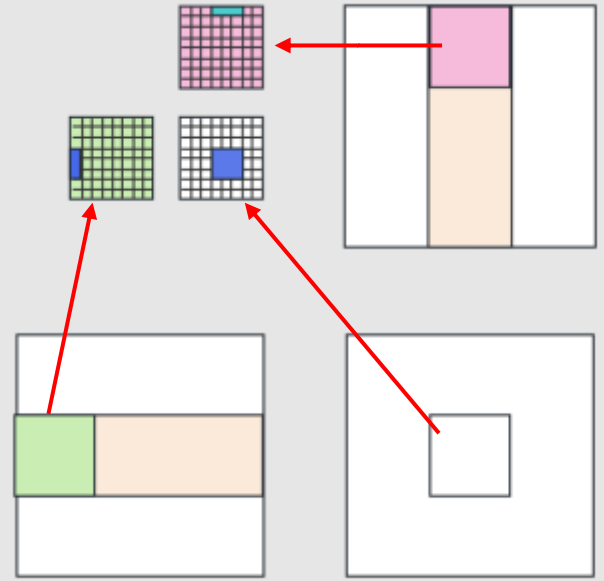
分块计算



每执行9次乘累加运算，需要进行6次访存

计算访存比：9/6 = 1.5

结合存储层次，多层分块



多层次的矩阵分块机制，更优的计算访存比

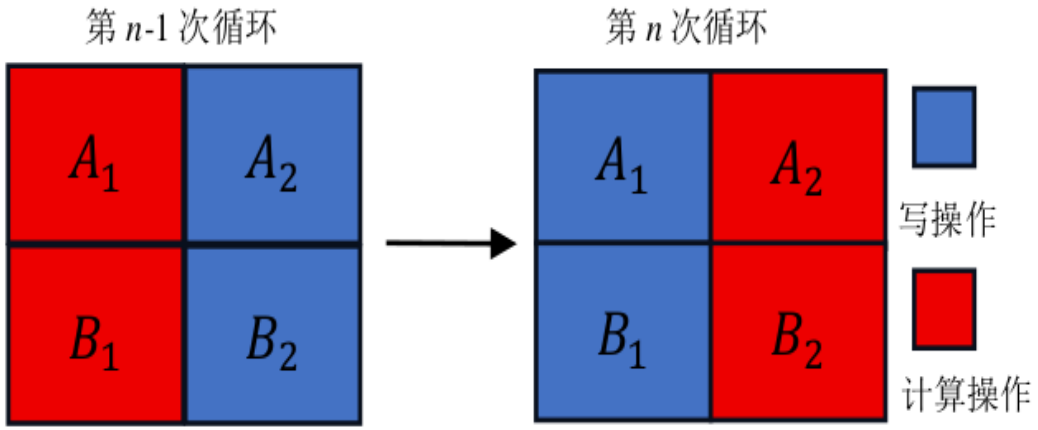
share memory : 一层分块，减少全局内存访问

register:二层矩阵分块，减少共享内存访问

我们的工作和优化方法

三、数据预取：双缓冲技术隐藏数据的访存延迟

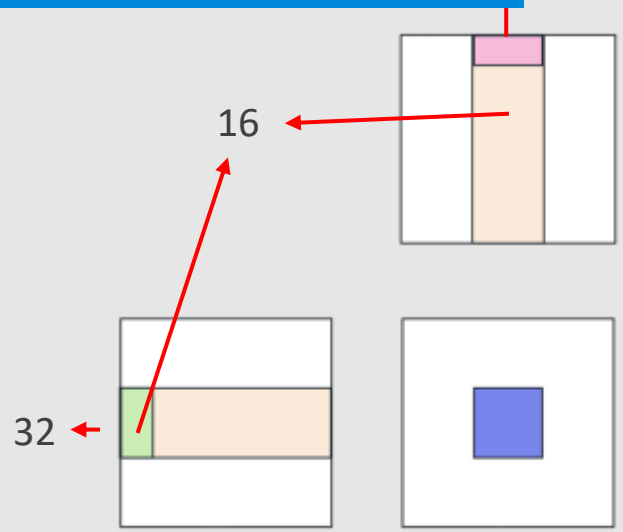
- **计算和访存重叠**：设置两个缓冲区，一个用于加载新的数据，另一个用于处理当前数据，从而减少等待时间，提升整体吞吐量。
- **双层双缓冲**：全局内存→共享内存 以及 共享内存→寄存器 都用双缓冲技术搬运数据。



我们的工作和优化方法

四、使用内联汇编控制延迟与同步，使用Tensorcore加输矩阵乘法

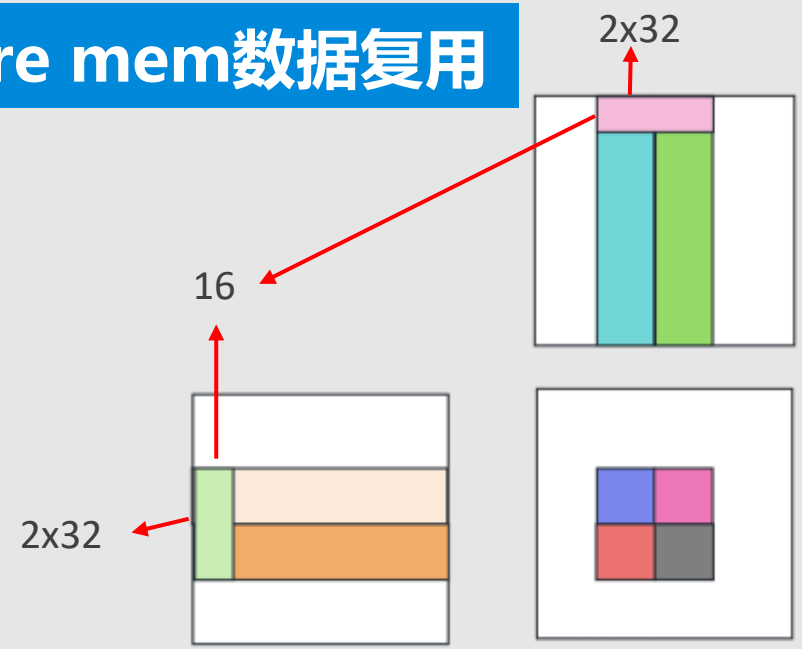
使用tensorcore加速



- global mem → share mem：读取2个32x16，计算1个32x32
- ds_read_m32x16_b16：从share mem读取32x16的块。
- v_mmac_f32_16x16x16_f16 ： 16x16矩阵乘法。
- 2条ds_read指令+4条v_mmac指令：实现读取32x16、16x32，计算32x32的矩阵乘法。

进一步优化

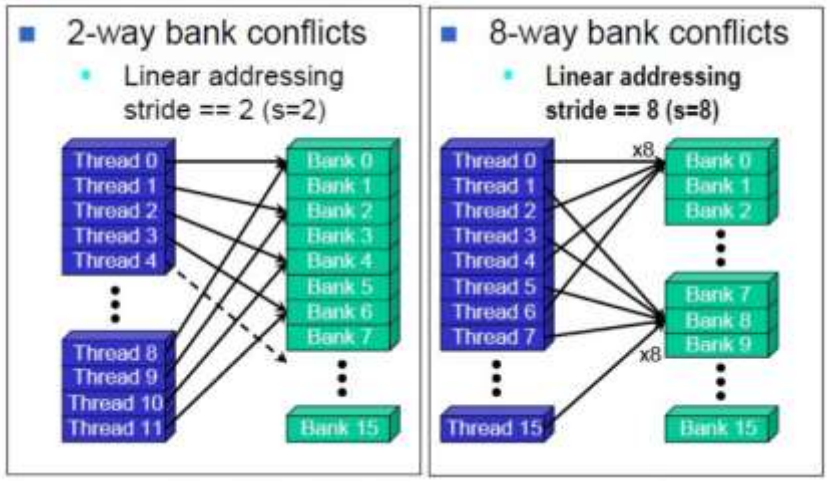
增加share mem数据复用

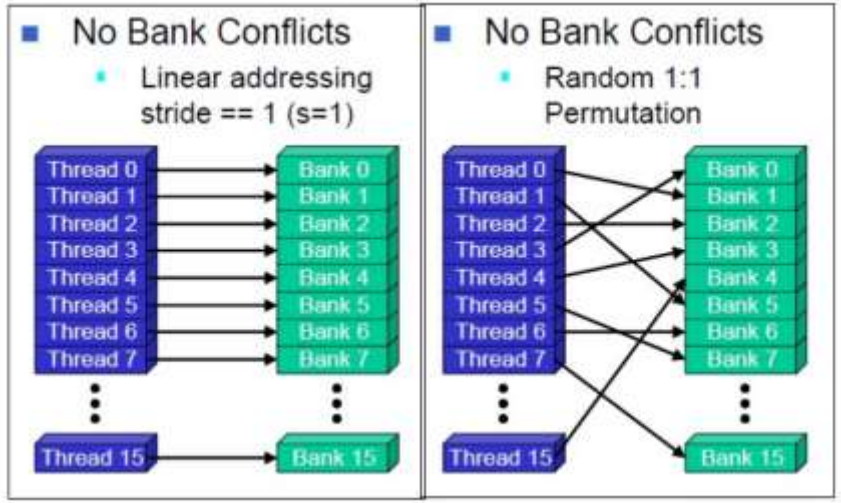


- global mem → share mem：读取4个32x16，计算4个32x32，计算访存比增加一倍
- 理论上每次从global mem读取的分块越多，计算访存比更高，但实际上受到share mem 容量限制

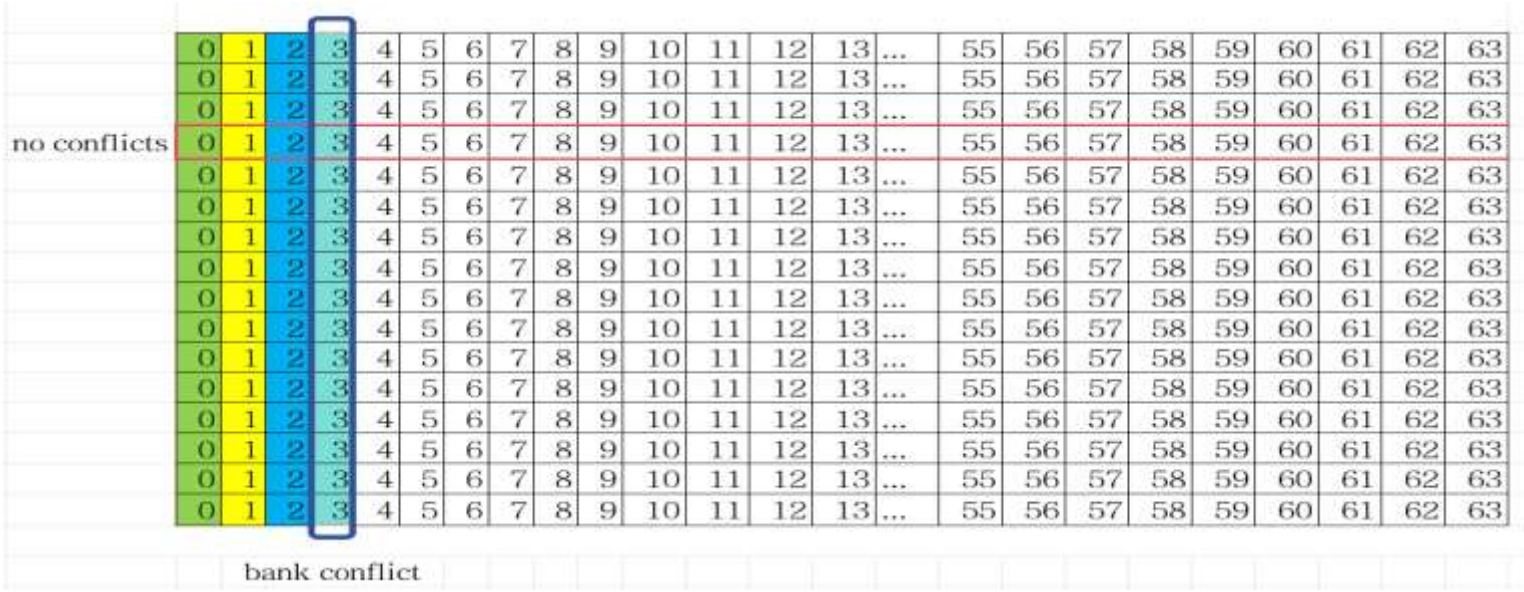
我们的工作和优化方法

五、降低share mem访问的Bank冲突





- k100_AI的共享内存具有32个bank。每个bank的大小为4字节,2个float16的长度。同一个workgroup的不同线程在同一个时钟周期内访问的同一个bank中的**不同数据**，产生bank冲突。
- pweight 的64x16分块读入share mem时需要转置成16x64，每行128字节刚好对应32个bank,即转置后同一列在同一个bank,写同一列的线程会产生bank冲突。
- 解决思路：使写同一列的线程尽可能少，同一个线程不要跨越写多列，一个线程只负责一列。



我们的工作和优化方法

六、合并写回：复用共享内存调整

问题：tensorcore各线程的计算结果在全局内存中的逻辑位置并不连续，直接写回global mem会造成很大的时间开销

线程id

$N \times H \times W$

K

0	16	32	48	0	16	32	48	0	16	32	48	0	16	32	48
1	17	33	49	1	17	33	49	1	17	33	49	1	17	33	49
2	18	34	50	2	18	34	50	2	18	34	50	2	18	34	50
3	19	35	51	3	19	35	51	3	19	35	51	3	19	35	51
4	20	36	52	4	20	36	52	4	20	36	52	4	20	36	52
5	21	37	53	5	21	37	53	5	21	37	53	5	21	37	53
6	22	38	54	6	22	38	54	6	22	38	54	6	22	38	54
7	23	39	55	7	23	39	55	7	23	39	55	7	23	39	55
8	24	40	56	8	24	40	56	8	24	40	56	8	24	40	56
9	25	41	57	9	25	41	57	9	25	41	57	9	25	41	57
10	26	42	58	10	26	42	58	10	26	42	58	10	26	42	58
11	27	43	59	11	27	43	59	11	27	43	59	11	27	43	59
12	28	44	60	12	28	44	60	12	28	44	60	12	28	44	60
13	29	45	61	13	29	45	61	13	29	45	61	13	29	45	61
14	30	46	62	14	30	46	62	14	30	46	62	14	30	46	62
15	31	47	63	15	31	47	63	15	31	47	63	15	31	47	63

解决方法：先写入共享内存，再重新进行线程分配，每个线程写回连续的16个数。

16　　16

线程id

32

0	...	0	1	...	1
2	...	2	3	...	3
...	...	....	...	...	...
60	...	60	61	...	61
62	...	62	63	...	63

我们的工作和优化方法

七、更多优化细节：减少内存事务、合并访存、循环融合等

减少内存事务

- 尽可能使用更长的数据类型进行访存，如float4类型，在一次内存事务中读取更长的数据长度。
- 减少了访存次数，提高带宽利用效率。

合并访存

- L1 cache为16k，128个缓存行，每个缓存行128字节。每次访存都会将128字节加载到缓存行。
- global中连续的128字节应在更少的访存次数和更近的访存中被利用。
- 每次迭代从global中读取32x32的数据会比64x16更友好，因为后者跨越更多缓存行，每个缓存行利用率更低。

循环融合

- 将两个明显独立的循环合并到同一个循环中交错执行。
- 将小的循环直接展开，减少循环计算消耗。

其他优化：控制单个线程使用的寄存器个数，控制每个group申请的共享内存大小，提高occupy，在多种制约因素中取得平衡。

创新性

访存平衡

卷积映射成矩阵乘法的过程中，pwight存储的连续性不变，pin存储的连续性受到破坏，导致在同一次迭代中，读取相同数据量时两者的延迟不一致。

调优思路：每次迭代pwight读取的块是pin的两倍，对决赛第2、5组数据有明显提升。

数据填充

决赛第1、6组数据的K维度不符合32倍数的矩阵分块和tensorcore指令要求。

调优思路：在计算时将K维度填充到32倍数，在读取和写回时，忽略补充的维度。与不填充使用普通指令相比，填充后使用tensorcore指令有很大提升。

共享复用

写回global阶段使用share mem重排时，不单独申请空间，而是复用前期缓冲pweight和pin的share mem,以提高利用率。

成果和总结

项目成果参赛计算机系统能力大赛-先导杯，凭借优异的实验效果和方法创新，荣获全国第五名（全国三等奖）

实验测试六组不同代表性特征的数据

n	c	h	w	k	r	s	u	v	p	q
16	128	64	64	27	3	3	1	1	1	1
16	256	32	32	256	3	3	1	1	1	1
16	64	128	128	64	3	3	1	1	1	1
2	1920	32	32	640	3	3	1	1	1	1
2	640	64	64	640	3	3	1	1	1	1
2	320	64	64	4	3	3	1	1	1	1

序号	优化前耗时 (us)	优化后耗时 (us)	加速比
1	3515.627197	263.397278	13.35
2	14719.093750	439.263794	33.52
3	14718.625977	589.392639	24.97
4	43832.644531	1433.512207	30.57
5	46812.703125	1299.222534	36.00
6	809.932861	183.935257	4.40

总结：

- 优化的整体思路和目标是使计算隐藏访存延迟，提高ALU占比。
- 本次卷积算子优化从多方面进行了深入探索与优化，包括计算与访存比的提升、共享内存利用、数据重用、访问冲突的控制、tensorcore指令的使用。同时使用双缓冲、数据分块、循环融合等多种优化手段，我们在初赛和决赛数据集上都取得了显著的性能提升，成功将运行时间大幅缩短。
- 本次项目验证了通过合理的算子优化策略，可以在满足准确性要求的前提下显著提升卷积算子的运行效率。项目中采用的优化策略和手段兼具一般性与特殊性，在国产dcu中具有广泛的借鉴价值，为提升深度学习模型的计算效率提供了有效思路。