Lab3

3240104875
王耀
2025/7/21

基准优化

Shared memory & Tiling

我决定先优化 shared memory 这里。我写了__shared__变量，但是我在初始化这个变量时遇到了问题。边界元素要额外读取 R/2 和 S/2 的。如果我把这些全部给边界自己加载，有点慢，可是要是分配给中间位置的线程，又不太好找准则。

不过我看到 kernel 大小只有 3*3，这还是很好的。

我初步把加载__shared__变量的任务完成了，但是在 V100 上跑的时候说没找到可用的 GPU？看了一下提交作业的说明网页，原来 GPU 需要申请。现在出现了新的错误，说我使用 shared memory 过大。

我一看，cmake 文件设置的通道数是 128，我设置的__shared__变量整整高达 0x168000 字节，这已经远远超过了单个 block 能使用的 shared memory 数量。

看来我应该在循环中分开加载__shared__变量，来一个通道加载一次。但是这样放在循环里面会白白冗余很多读取时间，不过我把 k 也给删掉应该能减少冗余，尽管这样 shared memory 没有被充分利用。现在仍然会出现把输入图像每个通道读取 k 次的冗余现象。

读取 shared memory 实在是一件要求细心，要求考虑各种情况的高难度操作，最终我调整好了，846ms 到 478ms，shared memory 的作用还是比较明显的。

Double buffering

Double buffering 这一块，也就不得不用上异步拷贝了，否则很难实现 double buffering。我对原本的代码进行了修改，把直接赋值改成了读取。但是异步拷贝的函数调用总是说我传递参数类型不匹配。

编译成功了，但是运行之后发生错误。经过我的测试，我发现同样的结构，直接对 shared 进行赋值没有问题，但是换成异步拷贝之后就出现了有对有错。难道是我改进的过程中改了什么重要的东西？似乎是我 barrier 用得不对，但是一番探索之后，可以见的用的其实是没问题的。

为什么我使用了异步拷贝进行 double buffering，速度反而还没有直接赋值快？794ms。可能是因为我每个线程调用异步拷贝数量过多而每次的量太小。每次异步拷贝只取 1 字节，疑似有点太少了。

但是改成一次性拷贝 C/2 字节，结果却错完了，甚至我在检查输出错误数据的时候还提示我访问越界？最终发现是我更改了逻辑之后忽略了 x 和 y 的初始化位置，所以始终不对。更正后错误出现在 1228800 处，这个位置恰好是第一张图的 96*0 完成 128 通道处，也就是 96*1 的位置，而且稳定地结果是 0？解决了，是因为 halo 区域也参与了 result 赋值，最终导致 95，100 覆盖了 96，0，最后出现了为 0 的问题。

但是为什么我利用异步拷贝实现 double buffering 一次取 64 字节，速度也才 642ms？异步拷贝速度怎么还没有直接赋值快？

不是我喜欢的速度，直接 profile。可能由于我 srun 的缘故，我远程连接到登陆节点的 nsight compute 捕捉不到 kernel，这是不好的。我看到有同学说节点内有 nsight compute 的 gui 版本，但是我没找到。所以我决定下载一个 cli 版本然后传进去。诶，lab3 下面原来写了 ncu 的路径。

profile 结果的计算带宽利用率非常低，仅仅 9%。而内存读取带宽为 49.9%，还好。为什么计算利用率这么低？不过这里的 timeline 在哪，怎么找不到？

嗯，根据 nsight compute 的提示，我的核心理论利用率只有 25%，而实际利用率更是只有 12.5%。我来分析一下。根据 lab3 文档和源码，block 是 16*16 线程的，32 线程构成一个 warp，SM 处理分配来的 block。Block 的共享内存是 48kb，SM 的共享内存是 192kb。依据 NVIDA 文档，SM 设计出来可以并行 2048 个线程，即 64 个 warp，8 个 block。也就是说我最好把每个 block 使用的共享内存调到 192/8=24kb。

确实快了，速度降到了 427ms。但是理论占用还是显示 37.5，实际运行更是只有 8 个 warp 在 SM 中运算，这是为什么呢？

哦，还是最上面给我提示的问题，因为 kernel 调用的 grid 尺寸太小，只有 49 个 block，但是又 80 个 SM，所以会有些 SM 根本就没分到 block，而且我的线程之间工作量有差异，读取有的线程负责两个元素，有的负责一个，所以才会有这个有点低的理论占用。

Bank conflict

现在 double buffering 完成，我来看一看 bank conflict 的问题。我看着看着感觉这个有点像哈希表。主打一个借助固定的规律是的数据在内存中尽量分散。那么借助哈希表的规律和 GPT 大人的指导，直接把共享内存中的数组写成 18*19*32 的似乎最好。

不对，这 GPT 在扯淡。32 个 bank，这样还是会恰好完全冲突。把 32 改成 33 应该好一点。这样改过之后把数据存储打散，速度又提升了，现在来到了 260ms。

Cooperating fetching

至于 cooperating fetching，我一开始就把任务分散给了中央线程，每个线程至多进行两次异步拷贝，所以到这里基础任务就告一段落了。这里把最后这个版本的 profile 数据写一下：kernel 的访存大小是，compute throughput 是 18.32%，memory throughput 是 47.44%，

```
Verifying...
Conv2d Reference Kernel Start...
CPU Conv2D:
3515 ms
0.0268 TFLOPS
Checking Results...
Correct
Your Conv2d:
260.4659 ms
0.3623 TFLOPS
h3240104875@v01:~/test/HPC101/src/lab3$
```

进一步优化

但是 nsight 的报告里已经明显看得出来瓶颈在数据读取了。这里如果我解决不掉，恐怕速度会一直上不来，所以我在报告里加了这个环节。

Nsight compute 给出了几个问题：加载和存储严重的地址不连续(uncoalesced)，以及 bank conflict 仍然存在，1.5-way bank conflict。GPT 告诉我一个 bank 是 4B，我调整了一下 padding，然后让共享内存数据 32B 对齐，速度提升到了 248ms。汇报说我的内存访问带宽利用率非常差，sector 内 32B 的带宽我平均线程才用 1.1B，warp 出现大量的阻塞停滞状态，可见异步拷贝严重影响了整体速度。

但是我每次异步读取 32B，至多调用 2 个 sector，怎么看也不会只有 1.1B 的使用率。这实在是太抽象了。GPT 大人在搜索网页之后告诉我，一个 warp 的内存指令会被整合成 request，然后因为 sector 和 request 之间还有一层 wavefront，导致了数据访问会被打散，所以就出现了这样的 1.1B 的抽象结果。

这里还是搞得我晕头转向的。我看到有同学说做好 shared memory 和 dp4a 就能 oj 得 100 了，也看到有同学遇到了和我一样的问题，一个 section 利用率只有 3.6。

我把异步拷贝换成了阻塞性拷贝，发现结果比异步拷贝的速度更快。异步拷贝 224，阻塞拷贝 190，这是人吗？我为了提高 sector 的利用率，专门更改了 fetch 函数，让 warp 内的 32 个线程合取 32 个连续的元素，但是却仍然提示利用率低，难道是没有及时同步线程？

不过加入了__ldg，速度确实提高了很多，来到了 103ms，oj 是 91ms，81 分。可是新的.ncu-rep 文件里显示扇区利用率还是非常低，1/32B

Compute (SM) Throughput [%] 14.70

Memory Throughput [%] 15.22

L1/TEX Cache Throughput [%] 27.24

L2 Cache Throughput [%] 3.15

DRAM Throughput [%] 0.15

这里什么都低倒是提醒了我，我可能一直都被指令的延迟给害了。我试了试放弃读取，纯计算，这是 1ms 的计算量（dp4a），那么剩下的都是读取。

我怀疑是读取的时候共享内存因为 bank conflict 导致的没法合并，所以尝试一次性取 4 个 8bit 元素，也因此把共享内存的使用再次拉到 48kb，结果快了，到了 61ms，但是 nsight 报告上还是说扇区利用率低。可能加速只是因为写入的 bank conflict 降下来了，扇区的问题还是没有解决。

减少了输入的重复读取，时间能 50ms 了。Oj 测试是 89.37，好难受的分数。不过扇区利用率始终低下的原因可能在这时候转变了：前面是 bank conflict 导致的无法合并，因为一个线程只读 1B，而 bank 是 4B 的，连续访问内存写入共享内存，也就会导致共享内存的 bank conflict。而在最新版，我为了保证 32 线程的连续读取，选择了一线程 4B 读 int，32 连续，导致读取了太多数据，共享内存使用过多，留给 warp 滞留等待调度的空间少，nsight 的报告也提到了这一点。

进阶任务

DP4A

使用这个倒是挺好改的，速度提升到 224ms。这里是相对上面“进一步优化”之前的。

Tensor core

wmma 提供的只有快速矩阵乘法，但是卷积运算要换成矩阵乘法就免不了显式或者隐式的 im2col，这就意味着不能直接用 wmma 的读取函数，也就是又绕回了刚才的读取瓶颈。本来想让 cuDNN 当 mvp，我做躺赢狗，结果发现 cuDNN 是主机侧的库，kernel 内用不了。

琢磨一个 im2col 的 API，琢磨完了发现这是个 host 调用生成模板的。太可恶，难道只能自己写吗。

N 的循环内展开成 256*(R*S*16)和(R*S*16)*K，之后再根据通道累加就可以了，这样利用内存是 41*2kb，有点大了。那么我改成 256*(R*S*4)和(R*S*4)*K 的。读取这一块，为了避开扇区瓶颈，我决定去尝试 cuda::wmma::load_matrix_sync 之后再存入，尽量的去绕开刚才的瓶颈。

诶，我可以仍然用这个函数向共享内存中写入，然后再从共享内存中读取进行计算。不对，wmma 对矩阵的用途做了明确区分，所以这个方案可能不太可行。

不对，好像异步拷贝要发挥合并的优势，好像需要同一 warp 内写完全一样的参数，这好像是我之前完全忽略的一个点，也可能是导致我异步拷贝效率低下的罪魁祸首。

不不，我改一下，我觉得一次就读 8B 就跳跃内存好像不太好，完全不能 32 线程合并，我决定先读 64*(R*S*16)和(R*S*16)*K。

经过多次调整，最终决定是 128*16 的，慢慢来罢，要不然共享内存就要不够了。好不容易解决掉了所有的编译错误，结果程序似乎陷入了死循环，跑了很长时间也没结束。我写了几个 printf 看了一下，是被卡在了 bar.arrive_and_wait，这倒是个奇怪的事，上次用 double buffering 就没有出现。哦，我的 barrier 忘记初始化了。

在改了对齐，倍数等等很多细节之后，终于跑通了，但是不光答案错了，时间也很长，四百多 ms。根据我的检查，应该是数据的读取阶段出了问题。而且第一次调用读取数据的函数是没有错误的，所以问题可能出现在边界处理上，或者我写的读取函数只能读初始情况。我再次细化锁定了位置，就是在 ii==0&&c==0 时才会出错。这说明就是这里的边界情况。

我来进一步排查是函数问题还是调用问题。经过特殊情况的检查，可以确定函数是没有问题的，那么就是调用和 double buffering 的问题。经过我的调整，a 数据的读取没有问题了，但是 w 数据的读取似乎还存在问题。

终于改对了，但是现在速度非常慢。整整四百多，我认为这是非常不好的。我在更改了异步拷贝的使用方法之后仍然出现了扇区利用率低下的问题。燃尽了，我只能再优化一下速度就算了。

最终还是没能跨过数据读取的瓶颈，扇区利用率仍然低下。我尝试了合作式的异步拷贝，但是在数据的读取上发生了未能解决的混乱，最后没能解决问题。

```
2025-07-28 09:43:25.697 Submission completed
Submit is completed
Message:
        judge successfully finished
Score 89.37 max.100 (Unweighted)
Judgement Message:
       Correct, Int8: 2137.75 GFLOPS, Half: 110.84 GFLOPS

Your Conv2d Cuda Kernel Start...
Verifying...
Conv2d Reference Kernel Start...
CUTLASS GPU Conv2D:
1.55552 ms
60.6690 TFLOPS
Checking Results...
Correct
Your Conv2d:
443.2051 ms
0.2129 TFLOPS
h3240104875@v02:~/test/HPC101/src/lab3$
```