LAB3 attention

BaseLine

运行cpu版本代码，只有 `Average 0.540321Gflop/s`，提升空间很大。

直接利用CUDA并行计算多个元素

仿照c代码的思路，只不过是将每个计算单独写一个算子，全部从内存中拿数据，只不过多线程并行，block_size设置为256，grid为所需要的个数，于是在没有太多更改的情况下代码效率飙升。

```
Description:    Optimized implementation.

Size: 63        Gflop/s: 7.53 (2048 iter, 0.136 seconds)
Size: 64        Gflop/s: 7.7 (2048 iter, 0.139 seconds)
Size: 65        Gflop/s: 8.27 (2048 iter, 0.136 seconds)
Size: 127       Gflop/s: 48.8 (2048 iter, 0.172 seconds)
Size: 128       Gflop/s: 44.8 (2048 iter, 0.192 seconds)
Size: 129       Gflop/s: 51.2 (2048 iter, 0.172 seconds)
Size: 191       Gflop/s: 118 (1024 iter, 0.121 seconds)
Size: 192       Gflop/s: 87.8 (1024 iter, 0.165 seconds)
Size: 193       Gflop/s: 118 (1024 iter, 0.124 seconds)
Size: 255       Gflop/s: 194 (1024 iter, 0.175 seconds)
Size: 256       Gflop/s: 145 (512 iter, 0.119 seconds)
...
Size: 1984      Gflop/s: 774 (8 iter, 0.162 seconds)
Size: 1985      Gflop/s: 1.33e+03 (16 iter, 0.188 seconds)
Size: 2047      Gflop/s: 1.48e+03 (16 iter, 0.185 seconds)
Size: 2048      Gflop/s: 785 (8 iter, 0.175 seconds)
Size: 2049      Gflop/s: 1.45e+03 (16 iter, 0.189 seconds)
Size: 4095      Gflop/s: 1.56e+03 (2 iter, 0.176 seconds)
Size: 4096      Gflop/s: 801 (2 iter, 0.343 seconds)
Size: 4097      Gflop/s: 1.53e+03 (2 iter, 0.180 seconds)
Size: 8191      Gflop/s: 1.6e+03 (2 iter, 1.371 seconds)
Size: 8192      Gflop/s: 820 (2 iter, 2.681 seconds)
Size: 8193      Gflop/s: 1.58e+03 (2 iter, 1.395 seconds)
Average 836.121197
```

注意到Gflop/s 对 n 特别敏感，尤其在32倍数时性能骤降。

- 原因猜测：

1. 线程对 global memory 的访问模式更容易发生 bank conflict 或地址 aliasing，多个线程访问地址步长为 4096（即一整行）时，地址可能落在相同的 L1 cache line 或 memory controller 上。导致 memory coalescing 失败，或者 GPU memory subsystem conflict。
2. block 数很规整，CUDA scheduler 会以非常统一但可能单一的 pattern 调度 block。

利用shared memory提升访存效率

我们利用每个block的shared memory来提升访存效率，在一个block中同时运算一个tile中的所有值，共同使用这个部分内的内存，同时按 `TILE_SIZE*TILE_SIZE` 的block窗口进行滑动。

我们可以对以下两个主要的矩阵乘法操作使用 shared memory + tiling 技术：

`QK^T = Q * K^T`

`Y = softmax(QK^T) * V`

这两个操作是标准的矩阵乘法，非常适合做 tile-based shared memory 加速。shared memory 减少了全局内存访问次数，多线程 tile 内的计算共用数据，提升带宽利用。新写两个算子 `tiled_matmul_attention` `tiled_matmul_QKT`。

运算效率继续上升，而且上一阶段的对于规模 `n` 敏感的问题得到了解决。

```
Description:    Optimized implementation.

Size: 63        Gflop/s: 6.42 (2048 iter, 0.159 seconds)
Size: 64        Gflop/s: 6.69 (2048 iter, 0.160 seconds)
Size: 65        Gflop/s: 7 (2048 iter, 0.161 seconds)
Size: 127       Gflop/s: 40.5 (1024 iter, 0.103 seconds)
Size: 128       Gflop/s: 38.3 (1024 iter, 0.112 seconds)
Size: 129       Gflop/s: 42.4 (1024 iter, 0.104 seconds)
Size: 191       Gflop/s: 98.2 (1024 iter, 0.145 seconds)
Size: 192       Gflop/s: 75.3 (1024 iter, 0.193 seconds)
...
Size: 1920      Gflop/s: 1.62e+03 (16 iter, 0.140 seconds)
Size: 1921      Gflop/s: 1.7e+03 (16 iter, 0.134 seconds)
Size: 1983      Gflop/s: 1.75e+03 (16 iter, 0.143 seconds)
Size: 1984      Gflop/s: 1.68e+03 (16 iter, 0.149 seconds)
Size: 1985      Gflop/s: 1.73e+03 (16 iter, 0.145 seconds)
Size: 2047      Gflop/s: 1.77e+03 (16 iter, 0.155 seconds)
Size: 2048      Gflop/s: 1.7e+03 (16 iter, 0.161 seconds)
Size: 2049      Gflop/s: 1.74e+03 (16 iter, 0.159 seconds)
Size: 4095      Gflop/s: 2.05e+03 (2 iter, 0.134 seconds)
Size: 4096      Gflop/s: 1.95e+03 (2 iter, 0.141 seconds)
Size: 4097      Gflop/s: 2.03e+03 (2 iter, 0.136 seconds)
Size: 8191      Gflop/s: 2.14e+03 (2 iter, 1.026 seconds)
Size: 8192      Gflop/s: 2.02e+03 (2 iter, 1.088 seconds)
Size: 8193      Gflop/s: 2.13e+03 (2 iter, 1.032 seconds)
Average 1131.267935
```

调参

调整 `TILE_SIZE` `BLOCK_SIZE`，有一点小提高： `TILE_SIZE=16` `BLOCK_SIZE=128或者256`

`Average 1131.515203`

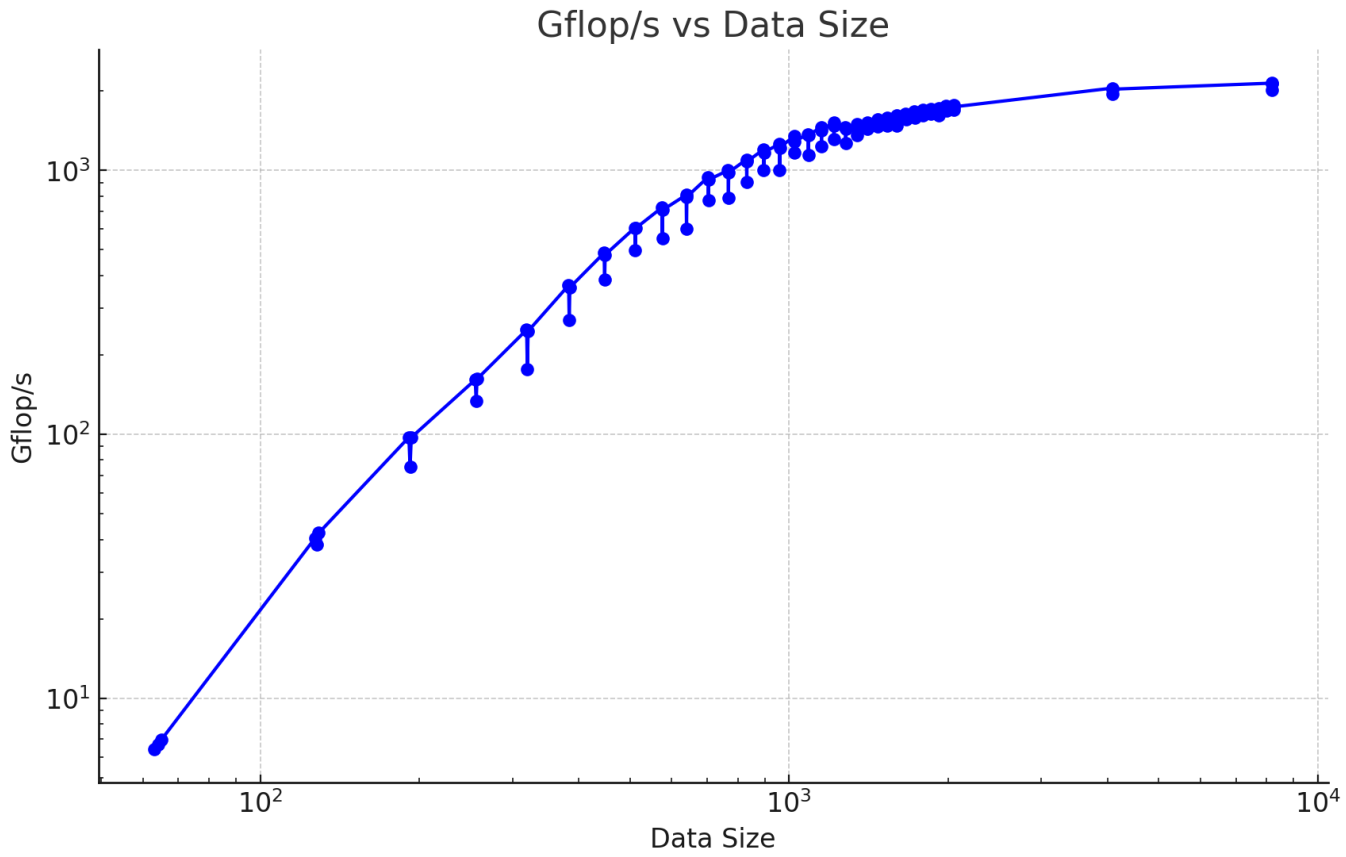
一些投机取巧

我们并不是一定要在softmax算子中找到一行的最大值才行，防止溢出某种程度下扫描一定量获得一个较大值其实就可以了，于是只扫描`n/2`，可以提升到：

`Average 1144.464168`

只扫描`n/3`，继续提升到 `Average 1148.440265`。

Gflops/s v.s. Data size图



使用性能工具分析

`srun --gres=gpu:1 nsys profile ./opt`

- GPU MemOps Summary (by Time)

时间占比	总时间 (ns)	次数	平均耗时 (ns)	中位数 (ns)	最小 (ns)	最大 (ns)	标准差 (ns)	操作类型
83.3%	734,292,243	510	1,439,788.7	364,545.5	1,952	35,666,281	4,570,946.3	CUDA memcpy HtoD
16.7%	147,607,157	102	1,447,129.0	469,921.5	2,144	25,650,664	4,371,576.9	CUDA memcpy DtoH

- GPU MemOps Summary (by Size)

总大小 (MB)	次数	平均大小 (MB)	中位数 (MB)	最小 (MB)	最大 (MB)	标准差 (MB)	操作类型
7,844.661	510	15.382	5.021	0.016	268.501	45.568	CUDA memcpy HtoD
1,568.932	102	15.382	5.021	0.016	268.501	45.749	CUDA memcpy DtoH

HtoD 花费远多于 DtoH,说明程序在将大量数据从 CPU 拷贝到 GPU 上。

Kernel 名称	GridXYZ	BlockXYZ	调用次数	总时间 (ms)	平均耗时 (ms)	时间占比
`tiled_matmul_QKT` (512×512 grid)	512×512×1	16×16×1	6	3,308	551.4	8.9%
`tiled_matmul_attention` (512×512 grid)	512×512×1	16×16×1	6	2,909	484.9	7.8%
`tiled_matmul_QKT` (513×513 grid)	513×513×1	16×16×1	3	1,546	515.4	4.1%
`tiled_matmul_attention` (513×513 grid)	513×513×1	16×16×1	3	1,465	488.5	3.9%
`tiled_matmul_QKT` (256×256 grid)	256×256×1	16×16×1	6	413	68.9	1.1%
`tiled_matmul_attention` (256×256 grid)	256×256×1	16×16×1	6	363	60.6	1.0%

`tiled_matmul_QKT @ 512x512 Grid`，总耗时 3308ms，占总执行时间 8.9%,StdDev 比其他配置大，说明这个配置下波动较大，可能存在内存瓶颈或线程调度不稳定。`tiled_matmul_QKT (256×256 grid)` 配置波动小一些。

绘图代码

```
import matplotlib.pyplot as plt

sizes = [
    63, 64, 65, 127, 128, 129, 191, 192, 193, 255, 256, 257, 319, 320, 321,
    383, 384, 385, 447, 448, 449, 511, 512, 513, 575, 576, 577, 639, 640,
    641, 703, 704, 705, 767, 768, 769, 831, 832, 833, 895, 896, 897, 959,
    960, 961, 1023, 1024, 1025, 1087, 1088, 1089, 1151, 1152, 1153, 1215,
    1216, 1217, 1279, 1280, 1281, 1343, 1344, 1345, 1407, 1408, 1409, 1471,
    1472, 1473, 1535, 1536, 1537, 1599, 1600, 1601, 1663, 1664, 1665, 1727,
    1728, 1729, 1791, 1792, 1793, 1855, 1856, 1857, 1919, 1920, 1921, 1983,
    1984, 1985, 2047, 2048, 2049, 4095, 4096, 4097, 8191, 8192, 8193
]
```

```
gflops = [
    6.4, 6.67, 6.97, 40.5, 38.2, 42.3, 97.5, 75.2, 97.3, 161, 134, 162, 249,
    177, 246, 367, 271, 360, 487, 387, 479, 606, 498, 604, 723, 554, 708,
    807, 601, 794, 943, 772, 922, 1e+03, 789, 982, 1.1e+03, 901, 1.08e+03,
    1.2e+03, 1e+03, 1.17e+03, 1.26e+03, 1e+03, 1.22e+03, 1.35e+03, 1.17e+03,
    1.29e+03, 1.37e+03, 1.14e+03, 1.36e+03, 1.46e+03, 1.23e+03, 1.42e+03,
    1.52e+03, 1.31e+03, 1.48e+03, 1.46e+03, 1.27e+03, 1.44e+03, 1.5e+03,
    1.36e+03, 1.46e+03, 1.52e+03, 1.44e+03, 1.49e+03, 1.56e+03, 1.47e+03,
    1.53e+03, 1.58e+03, 1.48e+03, 1.55e+03, 1.61e+03, 1.48e+03, 1.58e+03,
    1.64e+03, 1.56e+03, 1.61e+03, 1.67e+03, 1.58e+03, 1.64e+03, 1.69e+03,
    1.62e+03, 1.66e+03, 1.71e+03, 1.64e+03, 1.69e+03, 1.72e+03, 1.62e+03,
    1.7e+03, 1.75e+03, 1.68e+03, 1.73e+03, 1.77e+03, 1.7e+03, 1.74e+03,
    2.05e+03, 1.95e+03, 2.03e+03, 2.14e+03, 2.02e+03, 2.13e+03
]

plt.figure(figsize=(10, 6))
plt.plot(sizes, gflops, marker='o', linestyle='-', color='b')

plt.xlabel('Data Size')
plt.ylabel('Gflop/s')
plt.title('Gflop/s vs Data Size')

plt.grid(True)
plt.xscale('log')
plt.yscale('log')
plt.show()
```