小作业六：CUDA 优化（global memory，shared memory）

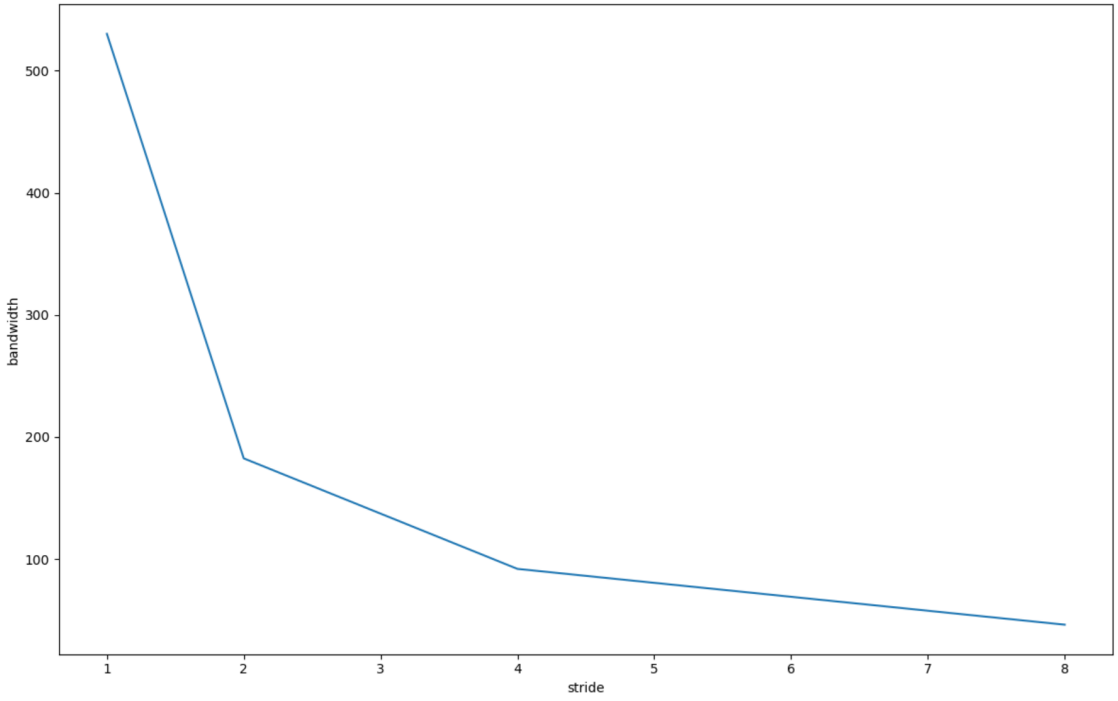
计04 何秉翔 2020010944

1. 测量结果

1.1 test_gmem.cu

STRIDE	Bandwidth
1	530.119
2	182.493
4	91.9947
8	46.2876

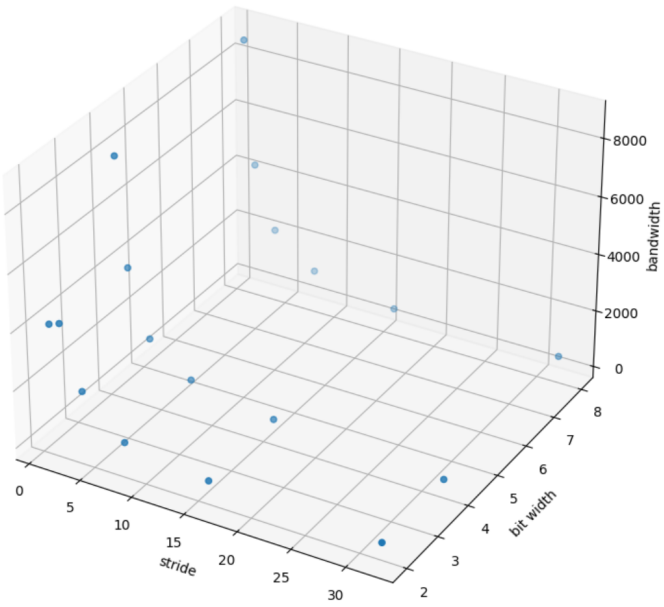
绘图如下：



1.2 test_smem.cu

STRIDE	BITWIDTH	Bandwidth
1	2	4149.82
1	4	8030.78
1	8	8657.22
2	2	4279.21
2	4	4305.62
2	8	4339.48
4	2	2154.81
4	4	2023.61
4	8	2173.55
8	2	828.301
8	4	1003.69
8	8	1087.67
16	2	426.853
16	4	505.617
16	8	544.071
32	2	215.257
32	4	251.199
32	8	544.071

绘图如下：



2．回答问题

- 分析 `test_gmem.cu` 的性能变化来源：
  - 性能变化的主要来源是 GPU 的哪种机制引起的？

    主要是 GPU 的缓存机制。
  - 这种机制如何影响该程序的性能？

    当某个地址的数据被访问时，cache 会从内存中将该地址附近的一块数据一起载入到 cache 中，因此当访问模式具有较强的局部性时，能充分发挥缓存的作用，降低数据访问的时间。

    `test_gmem.cu` 中的可变参数 STRIDE 即改变了线程访问数据的模式，当 STRIDE 为 1 时，相邻线程访问的数据也相邻，因此相当于是连续内存访问，此时缓存被充分利用，带宽最大，性能最优。随着 STRIDE 增加，相邻线程访问的数据地址相隔越来越大，cache 的命中率逐渐降低，导致数据需要从下一级缓存乃至内存中取出，降低程序性能。
  - 是否有其他的硬件功能参与了该程序的执行过程，它们会如何影响该程序的执行效率？

    有其他的硬件功能参与，比如：

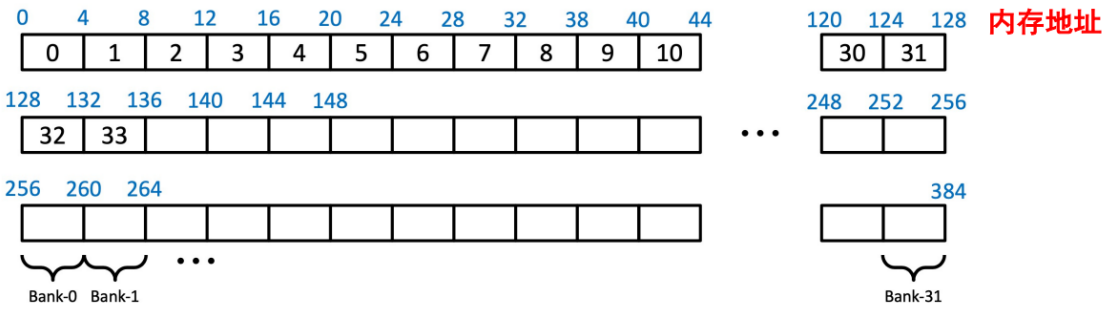
    - 寄存器：具有最低的延迟以及最高的带宽，但每个线程最多使用的寄存器有限，超过寄存器容量后会溢出到本地内存。
    - warp 调度器：调度器决定每个周期执行哪一个 warp 上的指令，对 warp 的调度顺序影响程序的执行效率。
    - SM：SM 上的资源是有限的，一个线程块会被分配到一个 SM 上执行，如果线程块较小，硬件资源足够，多个线程块可以同时执行，否则需要等一个线程块执行完才能开始下一个线程块的执行，线程块的执行时间和执行顺序都不确定，会影响程序的执行效率。
- 分析 `test_smem.cu` 的性能变化来源：
  - 固定 BITWIDTH 时，程序的性能变化来源于哪种硬件机制？

    主要来源于共享内存的硬件机制，当 BITWIDTH 固定时，共享内存的大小不变，bank 数以及每个 bank 的宽度不变，随着 STRIDE 的改变，对共享内存的访问模式出现变化，有可能有多个读请求同时访问一个 bank，此时不同的访问模式导致不同程度的访问冲突，影响程序的性能。
  - BITWIDTH 设置为 2,8 时性能变化的趋势相比于 BITWIDTH 设置为 4 时有什么不同之处，请解释。

    不同之处：

    - BITWIDTH 设置为 2 时，随着 STRIDE 的成倍增大，除了 STRIDE 从 1 增到 2 时性能基本不变，其余情况下带宽都依次减半，性能依次变为原来的一半。
    - BITWIDTH 设置为 4 时，随着 STRIDE 的成倍增大，所有情况下带宽都依次减半，性能依次变为原来的一半。
    - BITWIDTH 设置为 8 时，随着 STRIDE 的成倍增大，除了 STRIDE 从 16 增到 32 时性能基本不变，其余情况下带宽都依次减半，性能依次变为原来的一半。

    解释：这其实与 bank 的冲突情况有关，不同的 BITWIDTH 下，开辟的共享内存空间大小为 $BITWIDTH \times 32 \times 128B$，一共 32 个 bank，每个 bank 为 4 字节 32-bit，参考课件图如下：

    因此有：

    - BITWIDTH 为 2 时，共享内存空间大小为 $2 \times 32 \times 128B$，每个 bank 对应 64 块内存区域，即上图中有 64 行，`test_smem.cu` 程序一个 warp 内有 32 个线程，第 i 个线程访问第 $[2 \times i \times STRIDE, 2 \times i \times STRIDE + 1]$ 字节的数据，具体而言，对于所测 STRIDE 有：
      - STRIDE = 1，分别访问 `[0, 1]`、`[2, 3]`、...、`[62, 63]` 对应内存地址的数据，从上图我们可以看到，没有发生 bank conflict。
      - STRIDE = 2，分别访问 `[0, 1]`、`[4, 5]`、...、`[124, 125]` 对应内存地址的数据，从上图我们可以看到，没有发生 bank conflict，因此我们可以看到当 BITWIDTH = 2 时，STRIDE = 1 和 2 的带宽基

本一致。
    - 对于 STRIDE = 4、8、16、32 的情况，分别发生了 2-way、4-way、8-way、16-way 的 bank conflict。
- BITWIDTH 为 4 时，共享内存空间大小为 $4 \times 32 \times 128B$，每个 bank 对应 128 块内存区域，即上图中有 128 行，test_smem.cu 程序一个 warp 内有 32 个线程，第 i 个线程访问第 $[4 \times i \times STRIDE, 4 \times i \times STRIDE + 3]$ 字节的数据，具体而言，对于所测 STRIDE 有：
    - STRIDE = 1，分别访问 [0, 3]、[4, 7]、...、[124, 127] 对应内存地址的数据，从上图我们可以看到，没有发生 bank conflict。
    - STRIDE = 2，分别访问 [0, 3]、[8, 11]、...、[248, 251] 对应内存地址的数据，从上图我们可以看到，实际上发生了 2-way bank conflict。
    - 对于 STRIDE = 4、8、16、32 的情况，分别发生了 4-way、8-way、16-way、32-way 的 bank conflict。
    - 因此我们可以看到，随着 STRIDE 成倍增加，冲突程度成倍增加，导致性能逐渐变为原来的一半。
- BITWIDTH 为 8 时，共享内存空间大小为 $8 \times 32 \times 128B$，每个 bank 对应 256 块内存区域，即上图中有 256 行，test_smem.cu 程序一个 warp 内有 32 个线程，第 i 个线程访问第 $[8 \times i \times STRIDE, 8 \times i \times STRIDE + 7]$ 字节的数据，具体而言，对于所测 STRIDE 有：
    - STRIDE = 1，分别访问 [0, 7]、[8, 15]、...、[248, 255] 对应内存地址的数据，从上图我们可以看到，实际上发生了 2-way bank conflict。
    - STRIDE = 2，分别访问 [0, 7]、[16, 23]、...、[496, 503] 对应内存地址的数据，从上图我们可以看到，实际上发生了 4-way bank conflict。
    - 对于 STRIDE = 4、8 的情况，分别发生了 8-way、16-way 的 bank conflict。
    - STRIDE = 16 时，分别访问 [0, 7]、[128, 135]、... 对应内存地址的数据，已经产生最严重的 bank 冲突 32-way，即一个 warp 内每个线程都产生冲突。
    - STRIDE = 32 时，分别访问 [0, 7]、[256, 263]、... 对应内存地址的数据，仍然是 32-way bank conflict，因此我们可以看到当 BITWIDTH = 8 时，STRIDE = 16 和 32 的带宽基本一致。