高效能巨量資料與人工智慧系統 - HW4

You are asked to **implement conjugate transpose** of an $n \times n$ square matrix in CUDA and **answer some questions** individually in this assignment.

1. Introduction

The conjugate transpose of an $m \times n$ matrix $\mathbf{A}$ is an $n \times m$ matrix obtained by transposing $\mathbf{A}$ and applying complex conjugation to each entry (the complex conjugate of $a + bi$ being $a - bi$, for real numbers a and b).

Reference: Wikipedia

2. Requirements

Part 1

- Implement a naive CUDA code using CUDA global memory only.
- Improve previous code by using CUDA shared memory.
  Hint: consider how to ensure memory coalescing with shared memory.
- Improve previous code by avoiding as many bank conflicts as possible.
- Analyze and compare the results in the report.
  - Provide details of your experiment environment.
  - Vary the matrix size and compare the result of different implementations. Show what you observed and explain the result.
    Hint 1: consider how large the global memory cache line is.
    Hint 2: it's recommended to measure the execution time starting from the second run of a newly compiled executable, since CUDA generates instruction cache during the first run.

> **Important**
>
> You will need to submit all of your implementations,
> so remember to preserve your old codes.

> **Tip**

You can profile your code with Nsight Compute to observe the behavior of memory coalescing and bank conflicts.
The following metrics should be helpful:

- l1tex__average_t_sectors_per_request_pipe_lsu_mem_global_op_ld.ratio
- l1tex__average_t_sectors_per_request_pipe_lsu_mem_global_op_st.ratio
- l1tex__t_sectors_pipe_lsu_mem_global_op_ld.sum
- l1tex__t_sectors_pipe_lsu_mem_global_op_st.sum
- l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.sum
- l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.sum

Part 2

Answer the following questions in the report:

1. What is "warp" in GPU? How a 2-D thread block divided into warps?
   Hint: consider which dimension increases faster.
2. What is the difference between regular CUDA memory (allocated with `cudaMalloc`) and CUDA shared memory (allocated with `__shared__` qualifier)?
3. What is memory coalescing and when does it happen? Did you apply any technique to ensure memory coalescing? If so, explain how you realized it.
4. What is bank conflict and when does it happen? Did you avoid bank conflict in your code? If so, explain how you realized it.

3. Execution

A template code will be provided for you to get started.

We will run your code like this:

```
nvcc conj-transpose.cu -o conj-transpose
./conj-transpose <N> <times>
```

Be careful! We will test your code with an arbitrary $N \in [1, 4096]$.

For your reference, the followings are execution time of TA's implementations with default arguments:

Implementation	Execution Time on RTX 3070 (us)	Execution Time on RTX 4090 (us)
conj-transpose.cu	65	33
conj-transpose-shmem.cu	46	11
conj-transpose-shmem-bc-avoid.cu	43	8

You are not required to implement the fastest possible code,
but rather to ensure that the memory access patterns are coalesced and that bank conflicts are minimized.
As long as your optimized code produces correct results, meets the requirements, and shows lower execution time compared to the previous version,
you will get full points for the correctness portion.

4. Submission

- Deadline: 12/18 (Thurs) 23:59
- Submit a zip file named `<Student ID>_HW4.zip` with the following files:
  - `conj-transpose.cu`: implementation using global memory only
  - `conj-transpose-shmem.cu`: implementation using shared memory
  - `conj-transpose-shmem-bc-avoid.cu`: implementation using shared memory and avoiding bank conflicts
  - `<Student ID>_HW4_report.pdf`
- Please make sure your source codes can be compiled and executed without any problem.
  If some modifications are required to compile and run your code, there will be a small points deduction.

5. Grading Policy

- Correctness: 30%
  - implementation using global memory only: 10%
  - implementation using shared memory: 10%
  - implementation using shared memory and avoiding bank conflicts: 10%
- Report: 30%
- Question Answering: 40%
  - Q1: 5% + 5%
  - Q2: 10%
  - Q3: 5% + 5%

    - If you do not write memory-coalescing code, you will lose the latter 5%.
- Q4: 5% + 5%
    - If you do not write bank-conflict-avoiding code, you will lose the latter 5%

> ⚠ **Late Submission Penalty:**
>
> - Your score will be multiplied by 70% if you submit your homework after the deadline.

6. References

- CUDA C++ Programming Guide