并行与分布式计算导论 作业 5
PDC 2025s Homework 5

软截止期限 2025 年 6 月 11 日 23:59
Soft DDL: 2025 Jun. 11 23:59 (GMT+8)
截止期限 2025 年 6 月 18 日 23:59
DDL: 2025 Jun. 18 23:59 (GMT+8)

1. 使用 CUDA 实现二维矩阵转置，利用共享内存（shared memory）优化性能。给定一个大小为 M×N 的二维矩阵，存储在一维数组中（行优先），输出也为行优先的一维数组：

```
#include <cuda_runtime.h>
#include <cstdio>

#define TILE_DIM 32
#define BLOCK_ROWS 8

// CUDA kernel: 利用 shared memory 进行矩阵转置
__global__ void matrixTranspose(const float* input, float* output, int
M, int N) {
    // ① 声明共享内存 tile，避免 bank 冲突
    __shared__ float tile[TILE_DIM][TILE_DIM + 1];

    // ② 计算当前线程负责的全局索引
    int xIndex = ____________________;  // 计算列索引
    int yIndex = ____________________;  // 计算行索引

    // ③ 从全局内存读取数据到共享内存 tile
    for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS) {
        int y = yIndex + i;
        if (xIndex < N && y < M) {
            tile[threadIdx.y + i][threadIdx.x] = ____________________;
        }
    }

    __syncthreads();

    // ④ 计算转置后写入的全局索引（行列反转）
    xIndex = ____________________;
    yIndex = ____________________;

    // ⑤ 从共享内存写回转置后的数据到全局内存
    for (int i = 0; i < TILE_DIM; i += BLOCK_ROWS) {
        int y = yIndex + i;
        if (xIndex < M && y < N) {
            output[y * M + xIndex] = ____________________;
        }
    }
}
```

```
int main() {
    const int M = 1024;
    const int N = 2048;
    size_t size = M * N * sizeof(float);

    // 1. 分配host 内存并初始化
    float* h_input = (float*)malloc(size);
    float* h_output = (float*)malloc(size);
    for (int i = 0; i < M * N; ++i) {
        h_input[i] = static_cast<float>(i);
    }

    // 2. 设备端内存分配
    float *d_input, *d_output;
    cudaMalloc(&d_input, size);
    cudaMalloc(&d_output, size);

// 3. 复制输入数据到设备
cudaMemcpy(d_input, h_input, size, cudaMemcpyHostToDevice);

// 4. 设计线程块和网格大小
dim3 blockDim(____________________, ____________________);
dim3 gridDim((N + TILE_DIM - 1) / TILE_DIM,
             (M + TILE_DIM - 1) / TILE_DIM);

// 5. 启动kernel
matrixTranspose<<<gridDim, blockDim>>>(d_input, d_output, M, N);

// 6. 复制结果回 host
cudaMemcpy(h_output, d_output, size, cudaMemcpyDeviceToHost);

// 8. 释放内存
free(h_input);
free(h_output);
cudaFree(d_input);
cudaFree(d_output);

    return 0;
}
```