中山大学计算机学院本科生实验报告

课程名称：并行程序设计与算法

实验	CUDA矩阵转置	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛
Email	dengxy66@mail2.sysu.edu.cn	完成日期	2024.5.20

1、实验目的

1.CUDA Hello World

本实验为CUDA入门练习，由多个线程并行输出“Hello World！”。

输入：3个整数n,m,k，其取值范围为[1,32]

问题描述：创建n个线程块，每个线程块的维度为m×k，每个线程均输出线程块编号、二维块内线程编号及Hello World！（如，“Hello World from Thread (1, 2) in Block 10!”）。主线程输出“Hello World from the host!”。

要求：完成上述内容，观察输出，并回答线程输出顺序是否有规律?

2.CUDA矩阵转置

使用CUDA对矩阵进行并行转置。

输入：整数n，其取值范围均为[512, 2048]

问题描述：随机生成n×n的矩阵A，对其进行转置得到A^T，转置矩阵中第i行j列上的元素为原矩阵中j行i列元素，即$A^T_{ij} = A_{ji}$。

输出：矩阵A及其转置矩阵A^T，及计算所消耗的时间t。

要求：使用CUDA实现并行矩阵转置，分析不同线程块大小，矩阵规模，访存方式，任务/数据划分方式，对程序性能的影响。

2、实验过程和核心代码

执行命令

```
nvcc -o main main.cu
./main
```

1.CUDA Hello World

kernel函数

```
__global__ void helloFromGPU(int m, int k) {
    int blockId = blockIdx.x;
    int threadIdX = threadIdx.x;
    int threadIdY = threadIdx.y;
    printf("Hello World from Thread (%d, %d) in Block %d!\n", threadIdX, threadIdY,
blockId);
}
```

启动kernel

```
dim3 threadsPerBlock(m, k);
helloFromGPU<<<n, threadsPerBlock>>>(m, k);
```

等待线程结束

```
cudaDeviceSynchronize();
```

2.CUDA矩阵转置

kernel函数

```
__global__ void transpose(int *A, int *AT, int n) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < n && col < n) {
        AT[row * n + col] = A[col * n + row];
    }
}
```

分配内存

```
int *d_A, *d_AT;
cudaMalloc((void**)&d_A, n * n * sizeof(int));
cudaMalloc((void**)&d_AT, n * n * sizeof(int));
```

启动内核

```
clock_t start = clock();
transpose<<<gridSize, blockSize>>>(d_A, d_AT, n);
cudaDeviceSynchronize();
clock_t end = clock();
double time_taken = double(end - start) / CLOCKS_PER_SEC;
cudaMemcpy(h_AT, d_AT, n * n * sizeof(int), cudaMemcpyDeviceToHost);
```

3、实验结果

1.CUDA Hello World

n=3,m=3,k=3

```
Hello World from Thread (0, 0) in Block 0!
Hello World from Thread (1, 0) in Block 0!
Hello World from Thread (2, 0) in Block 0!
Hello World from Thread (0, 1) in Block 0!
Hello World from Thread (1, 1) in Block 0!
Hello World from Thread (2, 1) in Block 0!
Hello World from Thread (0, 2) in Block 0!
Hello World from Thread (1, 2) in Block 0!
Hello World from Thread (2, 2) in Block 0!
Hello World from Thread (0, 0) in Block 1!
Hello World from Thread (1, 0) in Block 1!
Hello World from Thread (2, 0) in Block 1!
Hello World from Thread (0, 1) in Block 1!
Hello World from Thread (1, 1) in Block 1!
Hello World from Thread (2, 1) in Block 1!
Hello World from Thread (0, 2) in Block 1!
Hello World from Thread (1, 2) in Block 1!
Hello World from Thread (2, 2) in Block 1!
Hello World from Thread (0, 0) in Block 2!
Hello World from Thread (1, 0) in Block 2!
Hello World from Thread (2, 0) in Block 2!
Hello World from Thread (0, 1) in Block 2!
Hello World from Thread (1, 1) in Block 2!
Hello World from Thread (2, 1) in Block 2!
Hello World from Thread (0, 2) in Block 2!
Hello World from Thread (1, 2) in Block 2!
Hello World from Thread (2, 2) in Block 2!
Hello World from the host!
```

线程的输出顺序具有一定的规律

2.CUDA矩阵转置

线程块大小	矩阵规模		
	512	1024	2048
1	0.000483	0.00179	0.007002
2	0.000169	0.000479	0.001778
4	8.1e-05	0.000161	0.000497
8	6.7e-05	8e-05	0.000177
16	5.6e-05	7e-05	0.00013

实验结果分析：

1.随着线程块大小的增加，转置时间通常减少，但矩阵规模越小时，影响效果更显著。

2.线程块大小对于不同矩阵规模的影响不同，矩阵规模较小时，转置时间的降低幅度更大。

线程块为4：

访存方式/矩阵规模	**512**	**1024**	**2048**
连续访存	4.7e-05	0.00012	0.00045
非连续访存	3.4e-05	0.000116	0.000443

实验结果分析：

1.连续访存通常比非连续访存具有更好的性能，说明在访存连续的情况下，内存访问模式更加友好，有助于提高性能。

2.性能差异随着矩阵规模的增加而放大，在较小的矩阵规模下，连续访存和非连续访存的性能差异相对较小，而在较大的矩阵规模下，连续访存和非连续访存的性能差异变得更加显著。

3.连续访存对于性能的提升有一定限度，虽然连续访存通常比非连续访存具有更好的性能，但在矩阵规模大的情况下，连续访存的转置时间仍然相对较长，说明在大规模矩阵的情况下，其他因素（比如：内存带宽）可能成为提升性能的瓶颈。

4、实验感想

1.通过这个实验，进一步加深了对CUDA编程的理解和认识，并掌握了一些基本的性能优化技巧。未来可以尝试更多的实验和优化策略，以进一步提升并行计算的性能。

2.访存模式对性能有显著影响，利用了内存的空间局部性，减少了缓存未命中率和内存延迟，在较大规模的矩阵上表现得尤为明显。

3.较大的线程块大小可以更好地利用GPU的计算资源和内存带宽，减少内存访问延迟，从而提高性能。然而，过大的线程块也可能导致线程间的同步和调度开销增加，需要找到一个平衡点来提升性能。

4.性能优化是一个复杂的过程，需要综合考虑多个因素，包括硬件架构、算法实现、数据划分和内存访问模式等。不同的GPU硬件和应用场景可能会有不同的优化策略，因此需要针对具体情况进行调整和测试。

5.CUDA编程可以显著提升计算密集型任务的性能，但需要对GPU硬件的深入理解、对并行算法的设计和实现等。