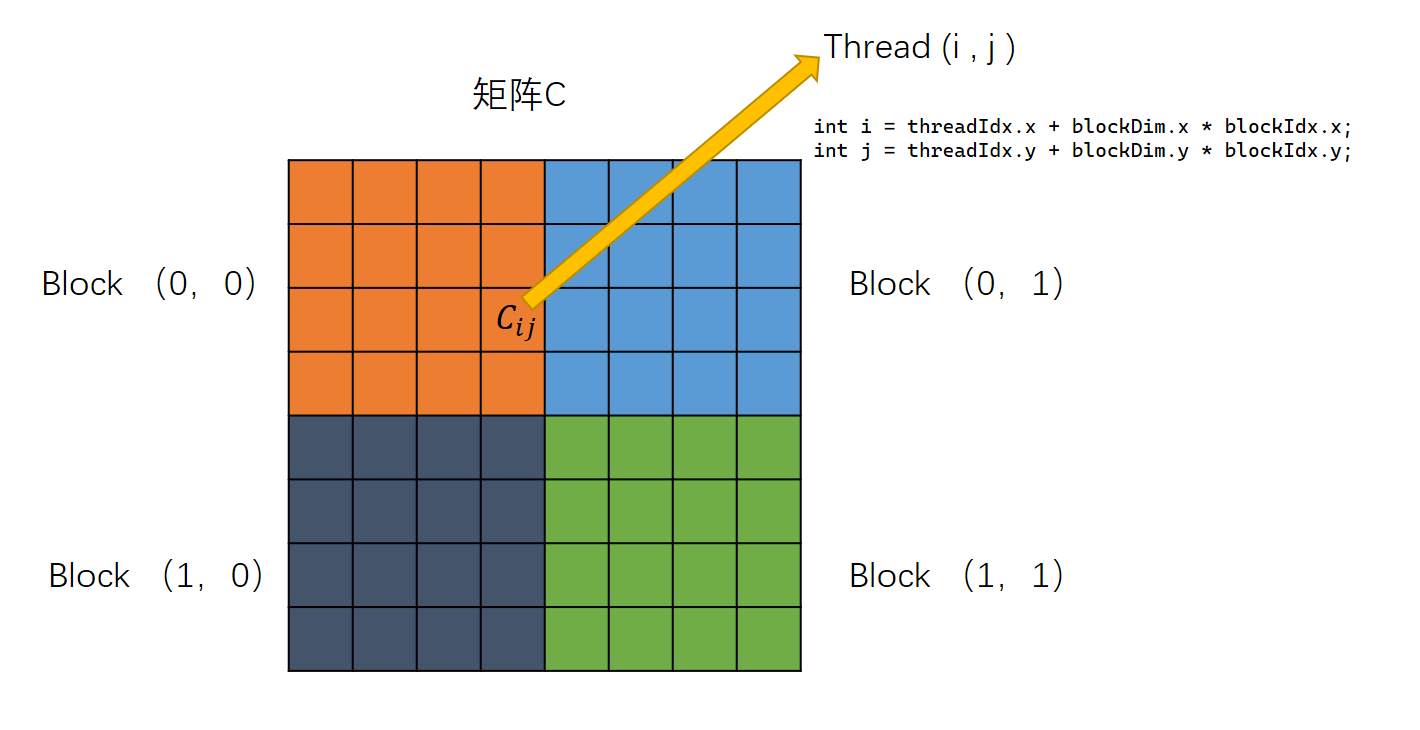
**中山大学数据科学与计算机院本科生实验报告**

**（2020学年秋季学期）**

课程名称：**高性能计算程序设计**  任课教师：**黄聃 批改人：**

|  |  |  |  |
| --- | --- | --- | --- |
| 年级+班级 | **2018级五班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **18340126** | 姓名 | **罗仁良** |
| Email | **luorliang@mail2.sysu.edu.cn** | 完成日期 | **2020年12月18日** |

1. **实验目的**
   1. 使用CUDA实现通用矩阵乘法
   2. 基于OpenMP+CUDA的多层次并行，实现通用矩阵乘法
   3. 调用CUBLAS计算矩阵相乘，比较分析不同方法的性能。
2. **实验过程和核心代码**
   1. **CUDA实现通用矩阵乘法**

实现的基本思路是，一个线程计算矩阵C的一个元素。在使用二维Blcok的情况下，比较容易得到矩阵元素与线程的对应关系。对应关系如图一所示。

**图一：线程索引与矩阵元素的对应关系**

* + 1. **Grid 和 Block 划分**

Grid和Blcok都是使用二维划分的形式，与二维矩阵相契合。

dim3 Block(blocksize, blocksize);

dim3 Grid((m+Block.x-1)/ Block.x, (k+Block.y-1)/ Block.y );

* + 1. **Kernel函数**

Kernel函数是一个线程完成的计算任务，也就是矩阵C一个元素的计算。需要的参数除了矩阵A、B、C之外，还需要传入矩阵A和矩阵B的列的维数。因为为了减少数据传输的次数，矩阵都是采用一维数组保存，需要矩阵的维度信息，实现对任意元素访问。

\_\_global\_\_ void matrix\_mul\_gpu(float \*A, float \* B, float \* C, int col\_a, int col\_b)

{

// 由线程索引确定计算Cij

int i = threadIdx.x + blockDim.x \* blockIdx.x;

int j = threadIdx.y + blockDim.y \* blockIdx.y;

int sum = 0;

for(int k=0;k<col\_a;k++)

{

sum += A[i\*col\_a+k]\*B[k\*col\_b+j];

}

C[i\*col\_b+j] = sum; // Cij = sum;

}

* + 1. **数据传输**

1. 把CPU主存上的数据传输到GPU上

cudaMemcpy(dA, A, sizeof(float) \* m \* n, cudaMemcpyHostToDevice);

cudaMemcpy(dB, B, sizeof(float) \* n \* k, cudaMemcpyHostToDevice);

1. 计算结果传回CPU的主存

cudaMemcpy(C, dC, sizeof(float) \* m \* k, cudaMemcpyDeviceToHost);

* + 1. **资源回收**

释放在CPU主存和GPU主存申请的资源。

    free(A);

    free(B);

    free(C);

    cudaFree(dA);

    cudaFree(dB);

    cudaFree(dC);

* 1. **OpenMP+CUDA多层次并行**

使用OpenMP把主进程切分多个并行的子线程，每个子线程调用使用CUDA实现的矩阵乘法函数，由此实现两个层次上的并行。一个是利用OepnMP的多线程的并行，另一个是使用CUDA实现的多线程并行。

* + 1. **线程任务划分**

把矩阵A按行分配，每个线程计算矩阵C的部分行。任务划分的时候尽量保证负载均衡。

  int q = m / num\_thread;

  int r = m % num\_thread;

    // first 矩阵A的起始行

    // count 需要计算的行数

    int first,count;

    count = q;

    if(rank < r) {

        count = q + 1;

        first = count \* rank;

    }

    else {

        first = count \* rank + r;

    }

* + 1. **数据传输**

根据任务划分的方式，矩阵B是每个子线程都共享的数据，所以把矩阵B传输到GPU上作为全局变量，减少矩阵B传输的次数。

    cudaMemcpy(dB,B,sizeof(float)\*n\*k,cudaMemcpyHostToDevice);

在每个线程中，线程根据任务划分，传输矩阵A中自己负责计算的那部分，减少数据传输的量。

    cudaMemcpy(dA,A+first\*n,sizeof(float)\*count\*n,cudaMemcpyHostToDevice);

* + 1. **主线程并行部分**

#pragma omp parallel num\_threads(num\_thread)

{ // 参数：

// CPU上的矩阵A数据

// GPU上的矩阵B数据

// 矩阵维度 m，n，k

// omp\_get\_thread\_num() 子线程的rank

// 并行线程数量 num\_thread

// block size blocksize

cudaMul(A,dB,C,m,n,k,omp\_get\_thread\_num(),num\_thread,blocksize);

}

* 1. **调用CUBLAS计算矩阵相乘**

使用CUBLAS的库进行矩阵乘法运算比较简单。有一个需要的注意的是，库函数中的矩阵大多以列为形式的储存。在计算的时候需要特别的转换。

    cublasHandle\_t handle;

    cublasCreate(&handle);

    cublasSgemm(handle,

                CUBLAS\_OP\_N,CUBLAS\_OP\_N,

                k,m,n,

                &alpha,

                dB,k,

                dA,n,

                &beta,

                 dC,k);

    cudaMemcpy(C, dC, sizeof(float) \* m \* k, cudaMemcpyDeviceToHost);

1. **实验结果**
   1. **任务1：CUDA实现矩阵乘法**

|  |  |  |
| --- | --- | --- |
| **矩阵规模** | **Blcok size** | **时间（ms）** |
| **512** | **32** | **2.968** |
| **1024** | **64** | **7.717** |
| **2048** | **128** | **47.553** |
| **4096** | **256** | **445.213** |
| **8192** | **512** | **2782.768** |

* 1. **任务2：OpenMP+CUDA实现矩阵乘法**

|  |  |  |
| --- | --- | --- |
| **矩阵规模** | **线程数量** | **时间（ms）** |
| **2048** | **1** | **23.613** |
| **2048** | **2** | **24.202** |
| **2048** | **4** | **75.833** |
| **2048** | **8** | **49.661** |

**注：Block size 为 128**

从结果看，不太符合预期。随着并行线程的增加，计算时间在增加。在增加到8个线程的时候，计算时间又下降了，结果比较奇怪。简单分析是多线程并行的时候传输数据的次数增加了，传输数据的开销，在程序运行中占的比列过大，降低了性能。

* 1. **任务3：调用CUBLAS 实现矩阵乘法**

|  |  |
| --- | --- |
| **矩阵规模** | **时间（ms）** |
| **512** | **225.983** |
| **1024** | **236.192** |
| **2048** | **260.726** |
| **4096** | **306.816** |
| **8192** | **588.315** |

数据规模较小时，性能较低。随着规模增大，相对性能在提升。猜测可能与库的具体实现相关，不同数据规模调用的线程数量，block规模的划分可能都不同，都会影响改程序的性能。

* 1. **性能对比**

比较任务1和任务3实现，可以看到在数据规模较小的时候，任务1的性能高很多，随着数据规模的提升，两者见的性能差距在缩小。当矩阵规模上升到8192时，调用库函数的性能就明显优于自己写的CUDA程序。

**优化方向：**任务1中的矩阵相乘，矩阵以行形式存储，在计算的时候cache的命中率较低，可更改为列形式存储，提高cache命中率。

1. **实验感想**

完成本次实验之后，熟悉了CUDA编程的基本方式。进一步加深了对GPU架构的理解，Grid、block、thread等之间的关系。同时，对GPU的内存分层模式有了更多的认识和理解。