多核设计HW1

葉珺明 19335253

Coding：

- 矩阵相加函数：

```
__global__ void addmat(const float* a, const float* b, float* c, int height, int width){
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    int j = blockIdx.y*blockDim.y + threadIdx.y;
    if(i<height&&j<width){
        c[i*width+j] = a[i*width+j] + b[i*width+j];
    }
}
```

- 数据拷贝：

```
float *a, *b, *c;
cudaMalloc((void **)&a, sizeof(float)*height*width);
cudaMalloc((void **)&b, sizeof(float)*height*width);
cudaMalloc((void **)&c, sizeof(float)*height*width);

cudaMemcpy(a, input1, sizeof(float)*height*width, cudaMemcpyHostToDevice);
cudaMemcpy(b, input2, sizeof(float)*height*width, cudaMemcpyHostToDevice);

dim3 gridDim(n_grid, n_grid);
dim3 blockDim(n_block, n_block);
addmat<<<gridDim, blockDim>>>(a, b, c, height, width);
cudaMemcpy(c, result, sizeof(float)*height*width, cudaMemcpyDeviceToHost);


cudaFree(a);
cudaFree(b);
cudaFree(c);

```

Writing：

2.1 程序整体逻辑

- 主函数：

  已提供。

  通过文件流的方式读取存有做相加的两个矩阵内容的二进制文件，并将文件存入input1和input2中，result用于保存两个矩阵相加的结果；通过文件流的方式将result输出到二进制文件中。

- GPU并行部分：

  独立编程。

  - 核函数，$C[i,j]=A[i,j]+B[i,j]$，矩阵对应部分相加求和存入结果矩阵；其中$Block(x,y)$分别对应$blockIdx.x$，$blockIdx.y$，$blockDim$人为设定，决定$Block$中$Thread$的分布，$thread(x,y)$分别与$threadIdx.x$，$threadIdx.y$对应。
  - 将数据拷贝到设备内存中，按照矩阵的大小在设备内存中开辟空间，通过$cudaMemcpy$将数据从主机内存拷贝到设备内存中，再将计算得到的结果存入result中。释放已经开辟的空间。

2.2 影响因素

- 一维 VS 二维：

  一维矩阵读取的速度比二维的读取速度要快，二维矩阵元素的索引相对于一维多了乘法和加法的运算或内存地址访问指令

- 线程块：

  通过调整合适的block或grid能够影响索引时带来的乘法和加法计算量，有利于加速GPU的运行速度。

  如，从$Grid:2*2;Block:2*2$到$Grid:1*4;Block:2*2$能够减少每个线程一次乘法和整数加法的计算。

- 每个线程计算的元素数量：

  通过将矩阵划分成小的方块能够在一定程度上带来性能上的提升，需要考虑能够使用的线程数和处理器数量等，充分利用GPU资源。

- 矩阵大小

  - 对于小矩阵来说，如$8*8$矩阵相对于$2048*2048$矩阵来说更适合采用一维的Grid和Block来计算，并且每个线程都能够分别处理一个元素，对于小矩阵，有足够的资源来单独处理每个元素，一维的网格分布和每个元素由一个线程处理能够提高性能
  - 对于大矩阵来说，需要考虑计算资源，调整网格大小，减少索引的计算量的同时又不使得网格分布不合理，并且一个线程处理的多个元素，避免运行时由资源不足引起的等待问题。

2.3 OpenMP和CUDA比较

实验比较运行时间(s)：

METHOD\SIZE	512	1024	2048
OpenMP(8 threads)	0.180923	1.39904	16.5863
CUDA	0.000026	0.000035	0.000031

比较可以看出，CUDA的并行运行时间都比OpenMP的运行时间要短，但CUDA在$cudaMalloc$和$cudaMemcpy$上的开销很大，数据量越大，在设备内存和主机内存上的花费的拷贝时间越长。

OpenMP随着矩阵的规模增加，运行时间也会明显增加；CUDA上，随着矩阵规模增加，其在并行时间的增加并不明显，但在实验过程中拷贝数据的时间开销明显增加。