3.1.两个矩阵相乘

考虑到在 GPU 代码中需要使用线性数组来表示矩阵，所以均采用方阵进行计算。
CPU 代码:
使用矩阵乘法定义进行计算即可。

```
#include<iostream>
using namespace std;
#define SIZE 3

void matrix_multiply(int* a, int* b, int* c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                c[i * n + j] = a[i * n + k] * b[k * n + j];
            }
        }
    }
}

void matrixgen(int* a, int n)
{
    for (int i = 0; i < n * n; i++) {
        a[i] = (int)rand() / (double)RAND_MAX * 100;
    }
}

void print_matrix(int* a, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", a[i * n + j]);
        }
        printf("\n");
    }
}

int main() {

    int n = SIZE;
    int* a, * b, * c;
    a = (int*)malloc(sizeof(int) * n * n);
    b = (int*)malloc(sizeof(int) * n * n);
    c = (int*)malloc(sizeof(int) * n * n);
```

```
    matrixgen(a, n);
    matrixgen(b, n);
    print_matrix(a,n);
    printf("\n");
    print_matrix(b,n);
    printf("\n");
    for (int i = 0; i < n * n; i++)
        c[i] = 0;
    matrix_multiply(a, b, c, n);
    print_matrix(c, n);
    return 0;
}
```

计算结果:

```
0 56 19
80 58 47
35 89 82

74 17 85
71 51 30
1 9 36

19 171 684
47 423 1692
82 738 2952
```

GPU 代码:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <cuda_runtime.h>
#define THREAD_NUM 256
#define MATRIX_SIZE 7
//const int blocks_num = (MATRIX_SIZE * MATRIX_SIZE + THREAD_NUM - 1) / THREAD_NUM;
const int blocks_num = 5;
bool InitCUDA()
{
    int count;
    cudaGetDeviceCount(&count);
    if (count == 0)
    {
        fprintf(stderr, "There is no device.\n");
        return false;
    }
    int i;
```

```
    for (i = 0; i < count; i++)
    {
        cudaDeviceProp prop;
        cudaGetDeviceProperties(&prop, i);
        if (cudaGetDeviceProperties(&prop, i) == cudaSuccess)
        {
            if (prop.major >= 1) break;
        }
    }
    if (i == count)
    {
        fprintf(stderr, "There is no device supporting CUDA 1.x.\n");
        return false;
    }
    cudaSetDevice(i);
    return true;
}

void matgen(int* a, int n)
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
            a[i * n + j] = (int)(rand() / (double)RAND_MAX * 100);
    }
}

void print_matrix(int* a, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", a[i * n + j]);
        }
        printf("\n");
    }
}

__global__ static void matMultCUDA(const int* a, const int* b, int* c, int n)
{
    const int tid = threadIdx.x;
    const int bid = blockIdx.x;

    const int idx = bid * THREAD_NUM + tid;
```

```
    const int row = idx / n;
    const int column = idx % n;
    int i;
    if (row < n && column < n)
    {
        int t = 0;
        for (i = 0; i < n; i++)
            t += a[row * n + i] * b[i * n + column];
        c[row * n + column] = t;
    }
}

int main()
{
    if (!InitCUDA()) return 0;
    int* a, * b, * c;
    int n = MATRIX_SIZE;
    a = (int*)malloc(sizeof(int) * n * n);
    b = (int*)malloc(sizeof(int) * n * n);
    c = (int*)malloc(sizeof(int) * n * n);
    srand(0);
    matgen(a, n);
    matgen(b, n);
    print_matrix(a, n);
    printf("\n");
    print_matrix(b, n);
    printf("\n");
    int* cuda_a, * cuda_b, * cuda_c;
    cudaMalloc((void**)&cuda_a, sizeof(int) * n * n);
    cudaMalloc((void**)&cuda_b, sizeof(int) * n * n);
    cudaMalloc((void**)&cuda_c, sizeof(int) * n * n);
    cudaMemcpy(cuda_a, a, sizeof(int) * n * n, cudaMemcpyHostToDevice);
    cudaMemcpy(cuda_b, b, sizeof(int) * n * n, cudaMemcpyHostToDevice);
    matMultCUDA << < blocks_num, THREAD_NUM, 0 >> > (cuda_a, cuda_b, cuda_c, n);
    cudaMemcpy(c, cuda_c, sizeof(int) * n * n, cudaMemcpyDeviceToHost);
    print_matrix(c, n);
    cudaFree(cuda_a);
    cudaFree(cuda_b);
    cudaFree(cuda_c);
    return 0;
}
```

首先使用 1 个 block，尺寸为 7 的矩阵进行测试

```
0 23 64 7 27 36 25
98 31 93 17 85 3 0
62 48 27 80 9 44 64
97 66 79 56 4 88 6
29 98 7 1 2 56 51
64 72 38 38 2 90 20
86 54 96 62 37 94 23

52 23 89 37 41 74 35
37 21 7 58 67 43 82
3 66 25 74 85 17 75
43 40 14 50 46 48 8
83 88 54 50 9 36 71
0 12 8 88 9 31 62
65 48 35 79 42 1 21

5210 8995 4480 12913 8920 4526 11416
14308 17239 16116 17670 15574 14135 19304
13428 11808 10727 20454 14898 12703 13482
10853 12767 12984 24481 18770 16925 21046
8679 6523 5797 16382 11060 8386 14269
9206 9228 9210 20856 14094 13184 17440
13990 17416 14855 28457 20301 17563 24072
```

经测试，使用了 11844 个时钟周期。

当矩阵尺寸是 block 尺寸的整数倍，不妨令矩阵尺寸为 4，block 为 2，结果如下。

```
0 23 64 7
27 36 25 98
31 93 17 85
3 0 62 48

27 80 9 44
64 97 66 79
56 4 88 6
29 98 7 1

5259 3173 7199 2208
7275 15356 5505 4280
10206 19899 8508 8898
4945 5192 5819 552
```

使用了 33158 个时钟周期。

当矩阵尺寸不是 block 尺寸的整数倍时，令矩阵尺寸为 7，block 为 5，结果如下。

```
0 23 64 7 27 36 25
98 31 93 17 85 3 0
62 48 27 80 9 44 64
97 66 79 56 4 88 6
29 98 7 1 2 56 51
64 72 38 38 2 90 20
86 54 96 62 37 94 23

52 23 89 37 41 74 35
37 21 7 58 67 43 82
3 66 25 74 85 17 75
43 40 14 50 46 48 8
83 88 54 50 9 36 71
0 12 8 88 9 31 62
65 48 35 79 42 1 21

5210 8995 4480 12913 8920 4526 11416
14308 17239 16116 17670 15574 14135 19304
13428 11808 10727 20454 14898 12703 13482
10853 12767 12984 24481 18770 16925 21046
8679 6523 5797 16382 11060 8386 14269
9206 9228 9210 20856 14094 13184 17440
13990 17416 14855 28457 20301 17563 24072
```

使用了 153888 个时钟周期。

通过比较矩阵尺寸为 7，block 为 1 和矩阵尺寸为 7，block 为 5 的结果，发现并不是 block 越多越好，可能是由于不同 block 之间的通信需要消耗了较多的时间。

使用 2048*2048 的矩阵进行测试，当 block 为 1 时消耗了 3661012 个时钟周期，block 为 2

时消耗了 3418222 个时钟周期。所以当计算量较大时，多个 block 才会发挥并行加速的作用。

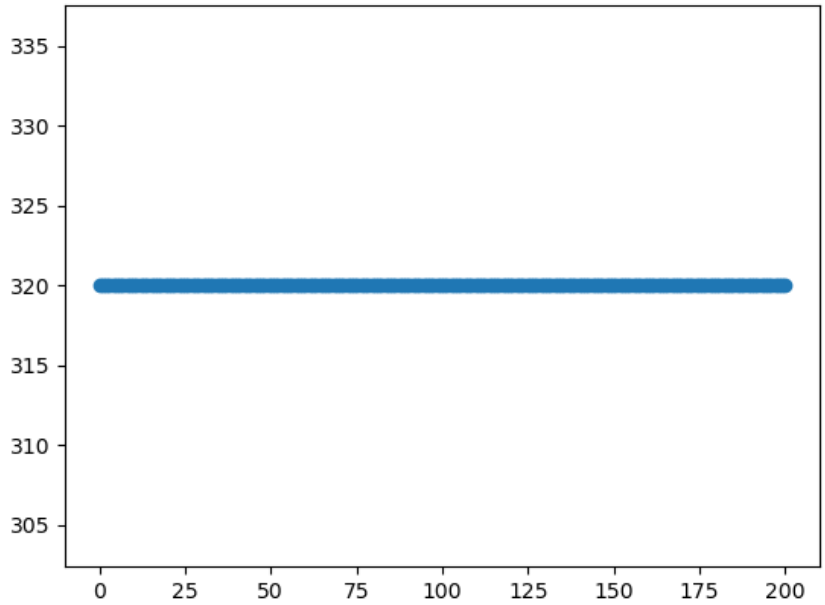
3.2.理解线程束的调度机制

加入计时功能，对 warp 的调度时间进行输出。

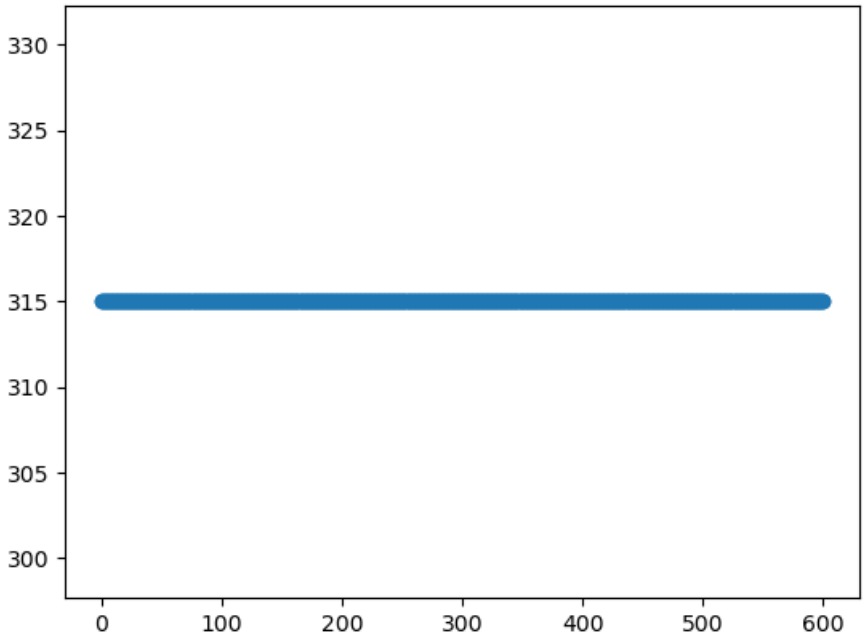
```
block:
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1       1       1       1       1       1

thread:
0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15
16      17      18      19      20      21      22      23      24      25      26      27      28      29      30      31
32      33      34      35      36      0       1       2       3       4       5       6       7       8       9       10
11      12      13      14      15      16      17      18      19      20      21      22      23      24      25      26

warp:
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0
1       1       1       1       1       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0

calc_thread:
0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15
16      17      18      19      20      21      22      23      24      25      26      27      28      29      30      31
32      33      34      35      36      37      38      39      40      41      42      43      44      45      46      47
48      49      50      51      52      53      54      55      56      57      58      59      60      61      62      63

completionTime:
231     231     231     231     231     231     231     231     231     231     231     231     231     231     231     231
231     231     231     231     231     231     231     231     231     231     231     231     231     231     231     231
231     231     231     231     231     232     232     232     232     232     232     232     232     232     232     232
232     232     232     232     232     232     232     232     232     232     232     232     232     232     232     232
```

分别修改 blockDim 和 gridDim，进行测试，记录完成的时间。
可以绘制出散点图。

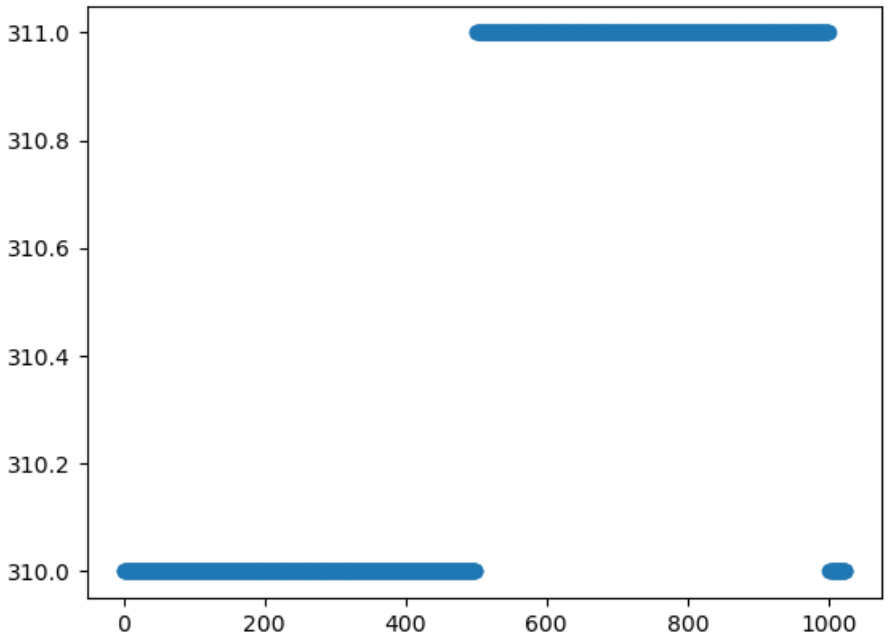
blockDim=1，gridDim=200：



blockDim=3，gridDim=200：



blockDim=5，gridDim=200:



代码

```
#include <stdio.h>
#include <string.h>

__global__
void what_is_my_id(int N, unsigned int* const block,
    unsigned int* const thread,
    unsigned int* const warp,
```

```
    unsigned int* const calc_thread,
    unsigned int* const completionTime)
{
    const unsigned int thread_idx = (blockIdx.x * blockDim.x) + threadIdx.x;
    if (thread_idx >= N) {
        return;
    }
    block[thread_idx] = blockIdx.x;
    thread[thread_idx] = threadIdx.x;
    warp[thread_idx] = threadIdx.x / warpSize;
    calc_thread[thread_idx] = thread_idx;
    completionTime[thread_idx] = clock() / 1620000;
}

void formatPrinter(const char* matName, int N, unsigned int* const mat) {
    printf("%s:", matName);
    for (int i = 0; i < N; i++) {
        if (i % 16 == 0) {
            printf("\n");
        }
        printf("%d\t", mat[i]);
    }
    printf("\n\n");
}

void showResult(int N, unsigned int* const block,
    unsigned int* const thread,
    unsigned int* const warp,
    unsigned int* const calc_thread,
    unsigned int* const completionTime)
{
    const char* colors[5] = { "block", "thread", "warp", "calc_thread",
"completionTime" };
    formatPrinter(colors[0], N, block);
    formatPrinter(colors[1], N, thread);
    formatPrinter(colors[2], N, warp);
    formatPrinter(colors[3], N, calc_thread);
    formatPrinter(colors[4], N, completionTime);
}

int main()
{
    int deviceId;
    int numberOfSMs;
```

```
    cudaGetDevice(&deviceId);
    cudaDeviceGetAttribute(&numberOfSMs, cudaDevAttrMultiProcessorCount, deviceId);

    const int N = 1 << 10;
    const int limN = (N / 32 + 1) * 32;
    size_t size = N * sizeof(int);

    unsigned int* block;
    unsigned int* thread;
    unsigned int* warp;
    unsigned int* calc_thread;
    unsigned int* completionTime;

    cudaMallocManaged(&block, size);
    cudaMallocManaged(&thread, size);
    cudaMallocManaged(&warp, size);
    cudaMallocManaged(&calc_thread, size);
    cudaMallocManaged(&completionTime, size);

    cudaError_t asyncErr;

    what_is_my_id << <3, 500 >> > (N, block, thread, warp, calc_thread,
completionTime);

    asyncErr = cudaDeviceSynchronize();
    if (asyncErr != cudaSuccess) printf("Error: %s\n", cudaGetErrorString(asyncErr));

    showResult(N, block, thread, warp, calc_thread, completionTime);

    cudaFree(block);
    cudaFree(thread);
    cudaFree(warp);
    cudaFree(calc_thread);
    cudaFree(completionTime);
}
```