Домашнее задание №2 (CUDA)
Умножение матриц на GPU средствами CUDA.

Постановка задачи

$$\boldsymbol{C = A \cdot B}$$

Элементы матриц A, B и C имеют тип double; матрицы имеют размер $N \times N$

Матрицы A, B и C хранятся в одномерных массивах в column-major порядке

Задания

1. Реализовать перемножение матриц с использованием глобальной памяти на CUDA.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include "cuda_runtime.h"
4   #include "device_launch_parameters.h"
5   #define BLOCK_SIZE 2
6
7   // Filling matrix with random double numbers from 0 to 9
8   void fill_matrix(double* H, int N)
9   {
10      int i;
11      for (i = 0; i < N * N; ++i)
12      {
13          H[i] = (rand() % 9);
14      }
15  }
16
```

```c
// Printing the matrix
void print_matrix(double* H, int N)
{
    int i, j;
    for (j = 0; j < N; ++j)
    {
        for (i = 0; i < N; ++i)
        {
            if (i != N - 1)
            {
                printf("%f\t", H[i * N + j]);
            }
            else
            {
                printf("%f\n", H[i * N + j]);
            }
        }
    }
    printf("\n");
}

// Sequential multiplication
void mult_pos(int N, double* A, double* B, double* C)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            for (int k = 0; k < N; k++)
            {
                C[j * N + i] += A[k * N + i] * B[j * N + k];
            }
        }
    }
}
```

```
52
53  // Comparing matrices
54  void compare_matrices(double* C, double* hostC, int N)
55  {
56      int i, flag;
57      double eps;
58      flag = 0;
59      eps = 0.001;
60      for (i = 0; i < N * N; ++i)
61      {
62          if (abs(C[i] - hostC[i]) >= eps)
63          {
64              flag = 1;
65              break;
66          }
67      }
```

```
68      if (flag == 0)
69      {
70          printf("Resuls of sequantial and parallel multiplications are equal \n");
71      }
72      else
73      {
74          printf("Resuls of sequantial and parallel multiplications are different \n");
75      }
76  }
77
```

```
78  //CUDA multiplication
79  __global__ void matrixmultiplication(double* A, double* B, double* C, int N)
80  {
81      //Computing position of elemets
82      int c = blockIdx.x * blockDim.x + threadIdx.x;
83      int r = blockIdx.y * blockDim.y + threadIdx.y;
84      double sum = 0.0;
85      //Computing elements for matrix C
86      if (r < N && c < N)
87      {
88          // each thread computes one element of the block sub-matrix
89          for (int i = 0; i < N; i++)
90          {
91              sum += B[r * N + i] * A[i * N + c];
92          }
93      }
94      C[r * N + c] = sum;
95  }
```

```
96
97  int main(int argc, char* argv[])
98  {
99      int N = 4;
100     double *hostA;
101     double *hostB;
102     double *hostC;
103     double *C;
104     float time;
105
106     //Allocating memory for hosts
107     hostA = (double*) calloc(N*N, sizeof(double));
108     hostB = (double*) calloc(N*N, sizeof(double));
109     hostC = (double*) calloc(N*N, sizeof(double));
110
111     C = (double*) calloc(N*N, sizeof(double));
112
```

```
113     //Filling matrices
114     fill_matrix(hostA, N);
115     fill_matrix(hostB, N);
116
117     //Printg matrices A and B
118     printf("hostA:\n");
119     print_matrix(hostA, N);
120     printf("hostB:\n");
121     print_matrix(hostB, N);
122
123     //Allocating memory for devices
124     double *deviceA;
125     cudaMalloc((void **)&deviceA, N*N * sizeof(double));
126     double *deviceB;
127     cudaMalloc((void **)&deviceB, N*N * sizeof(double));
128     double * deviceC;
129     cudaMalloc((void **)&deviceC, N*N * sizeof(double));
130
```

```
    //Defining dimensions of the block and the grid
    dim3 threadsPerBlock = dim3(BLOCK_SIZE, BLOCK_SIZE);
    int N_new = 0;
    if (N % BLOCK_SIZE == 0)
        N_new = int (N / BLOCK_SIZE);
    else
        N_new = int (N / BLOCK_SIZE) + 1;
    dim3 blocksPerGrid = dim3(N_new,N_new);

    //Defining and initializing variables for computing time
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

```

```
//Recording the computing time
    cudaEventRecord( start, 0);

    //Copying matrices from host to device
    cudaMemcpy(deviceA, hostA, N*N * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(deviceB, hostB, N*N * sizeof(double), cudaMemcpyHostToDevice);

    //Calling multiolication function
    matrixmultiplication<<<blocksPerGrid, threadsPerBlock>>>(deviceA, deviceB, deviceC, N);

    //Copying matrices from device to host
    cudaMemcpy(hostC, deviceC, N*N * sizeof(double), cudaMemcpyDeviceToHost);
    cudaEventRecord( stop, 0);
    cudaEventSynchronize(stop);

```

```
    //Computing time
    cudaEventElapsedTime( &time, start, stop);
    printf("Time: %.2f \n",time);

    //Sequantial multiplication
    mult_pos(N, hostA, hostB, C);

    //Printg result of CUDA multiplication
    printf("hostC:\n");
    print_matrix(hostC, N);

    //Printg result of sequential multiplication
    printf("C:\n");
    print_matrix(C, N);

//Comparing results of CUDA multiplication and sequential multiplication
    compare_matrices(C, hostC, N);

```

```
178    //Deallocationg the memory
179        cudaFree(deviceA);
180        cudaFree(deviceB);
181        cudaFree(deviceC);
182        free(hostA);
183        free(hostB);
184        free(hostC);
185        cudaEventDestroy( start );
186        cudaEventDestroy( stop );
187        return 0;
188    }
```

```
hostA:
1.000000        5.000000        6.000000        5.000000
7.000000        7.000000        1.000000        7.000000
0.000000        1.000000        5.000000        5.000000
7.000000        3.000000        4.000000        4.000000

hostB:
6.000000        8.000000        8.000000        1.000000
0.000000        8.000000        8.000000        1.000000
7.000000        6.000000        8.000000        5.000000
1.000000        6.000000        4.000000        0.000000
```

Размер матрицы 128 x 128

Размер блока 2 x 2

```
Time: 0.21
```

Размер блока 4 x 4

```
Time: 0.21
```

Размер блока 8 x 8

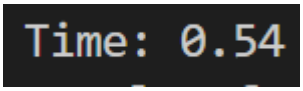
```
Time: 0.22
```

Размер блока 16 x 16

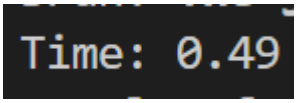
Размер блока 32 x 32

Размер матрицы 256 x 256

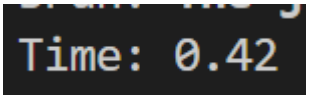
Размер блока 2 x 2



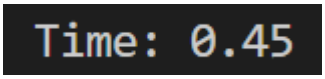
Размер блока 4 x 4



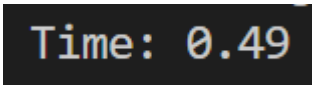
Размер блока 8 x 8



Размер блока 16 x 16

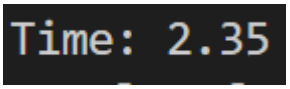


Размер блока 32 x 32

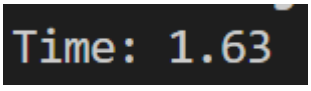


Размер матрицы 512 x 512

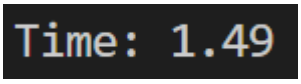
Размер блока 2 x 2



Размер блока 4 x 4



Размер блока 8 x 8



Размер блока 16 x 16

```
Time: 1.37
```

Размер блока 32 x 32

```
Time: 1.35
```

Размер матрицы 1024 x 1024

Размер блока 2 x 2

```
Time: 13.40
```

Размер блока 4 x 4

```
Time: 7.87
```

Размер блока 8 x 8

```
Time: 6.97
```

Размер блока 16 x 16

```
Time: 5.94
```

Размер блока 32 x 32

```
Time: 6.46
```

Размер матрицы 2048 x 2048

Размер блока 2 x 2

```
Time: 131.52
```

Размер блока 4 x 4

```
Time: 47.33
```

Размер блока 8 x 8

```
Time: 29.98
```

Размер блока 16 x 16

```
Time: 26.78
```

Размер блока 32 x 32

```
Time: 38.97
```

2. Ускорить передачу матрицы из CPU в GPU за счет использования Pinned памяти.

```
97  int main(int argc, char* argv[])
98  {
99      int N = 2048;
100     float time;
101     double *hostA;
102     double *hostB;
103     double *hostC;
104     double *C;
105
106     //Allocating memory for hosts
107     cudaMallocHost ((void**) &hostA, N*N * sizeof(double), cudaHostAllocDefault);
108     cudaMallocHost ((void**) &hostB, N*N * sizeof(double), cudaHostAllocDefault);
109     cudaMallocHost ((void**) &hostC, N*N * sizeof(double), cudaHostAllocDefault);
110
111     C = (double*) calloc(N*N, sizeof(double));
112
```

```
113     //Filling matrices
114     fill_matrix(hostA, N);
115     fill_matrix(hostB, N);
116
117     //Printg matrices A and B
118     //printf("hostA:\n");
119     //print_matrix(hostA, N);
120     //printf("hostB:\n");
121     //print_matrix(hostB, N);
122
```

```
123     //Allocating memory for devices
124     double *deviceA;
125     double *deviceB;
126     double * deviceC;
127     cudaMalloc((void **)&deviceA, N*N * sizeof(double));
128     cudaMalloc((void **)&deviceB, N*N * sizeof(double));
129     cudaMalloc((void **)&deviceC, N*N * sizeof(double));
130
131     //Defining dimensions of the block and the grid
132     dim3 threadsPerBlock = dim3(BLOCK_SIZE, BLOCK_SIZE);
133     int N_new = 0;
134     if (N % BLOCK_SIZE == 0)
135         N_new = int (N / BLOCK_SIZE);
136     else
137         N_new = int (N / BLOCK_SIZE) + 1;
138     dim3 blocksPerGrid = dim3(N_new,N_new);
139
```

```
    //Defining and initializing variables for computing time
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

//Recording the computing time
    cudaEventRecord( start, 0);

    //Copying matrices from host to device
    cudaMemcpy(deviceA, hostA, N*N * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(deviceB, hostB, N*N * sizeof(double), cudaMemcpyHostToDevice);

    //Calling multiplication function
    matrixmultiplication<<<blocksPerGrid, threadsPerBlock>>>(deviceA, deviceB, deviceC, N);

```

```
     //Copying matrices from device to host
     cudaMemcpy(hostC, deviceC, N*N * sizeof(double), cudaMemcpyDeviceToHost);
     cudaEventRecord( stop, 0);
     cudaEventSynchronize( stop );

     //Computing time
     cudaEventElapsedTime( &time, start, stop);
     printf("Time: %.2f \n",time);

     //Sequantial multiplication
     mult_pos(N, hostA, hostB, C);

     //Printg result of CUDA multiplication
     //printf("hostC:\n");
     //print_matrix(hostC, N);
```

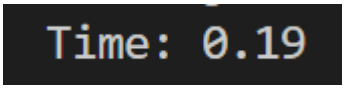
```
    //Printg result of sequential multiplication
    //printf("C:\n");
    //print_matrix(C, N);

////Comparing results of CUDA multiplication and sequential multiplication
    compare_matrices(C, hostC, N);

//Deallocationg the memory
    cudaFree(deviceA);
    cudaFree(deviceB);
    cudaFree(deviceC);
    cudaFreeHost(hostA);
    cudaFreeHost(hostB);
    cudaFreeHost(hostC);
    cudaEventDestroy( start );
```

```
     cudaEventDestroy( start );
     cudaEventDestroy( stop );
     return 0;
}
```

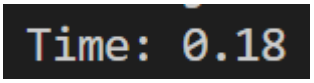
Размер матрицы 128 x 128

Размер блока 2 x 2

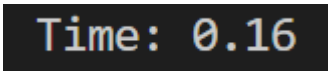
Размер блока 4 x 4



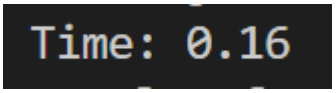
Размер блока 8 x 8



Размер блока 16 x 16

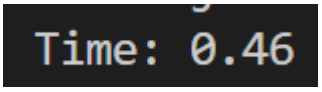


Размер блока 32 x 32

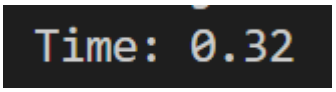


Размер матрицы 256 x 256

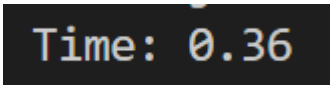
Размер блока 2 x 2



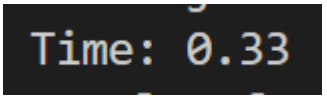
Размер блока 4 x 4



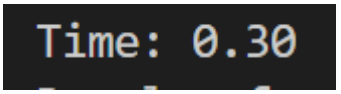
Размер блока 8 x 8



Размер блока 16 x 16

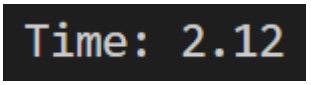


Размер блока 32 x 32



Размер матрицы 512 x 512

Размер блока 2 x 2



Размер блока 4 x 4

```
Time: 1.25
```

Размер блока 8 x 8

```
Time: 1.10
```

Размер блока 16 x 16

```
Time: 0.93
```

Размер блока 32 x 32

```
Time: 0.98
```

Размер матрицы 1024 x 1024

Размер блока 2 x 2

```
Time: 16.62
```

Размер блока 4 x 4

```
Time: 6.32
```

Размер блока 8 x 8

```
Time: 4.15
```

Размер блока 16 x 16

```
Time: 3.53
```

Размер блока 32 x 32

```
Time: 3.59
```

Размер матрицы 2048 x 2048

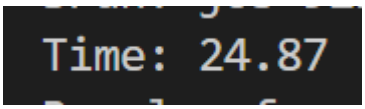
Размер блока 2 x 2

```
Time: 126.24
```

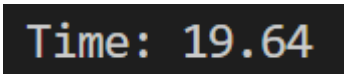
Размер блока 4 x 4

```
Time: 44.93
```

Размер блока 8 x 8



Размер блока 16 x 16



Размер блока 32 x 32