ФГБОУ ВО «Национальный исследовательский университет «МЭИ»

**Кафедра вычислительных машин, систем и сетей**

**Лабораторная работа №4**

по курсу «Программное обеспечение высокопроизводительных систем»

по теме «Размещение данных в глобальной, разделяемой и константной памяти»

Выполнили:

Певченко Сергей

Локотков Борис

Салий Екатерина

Бригада № 6

Группа: А-07м-18

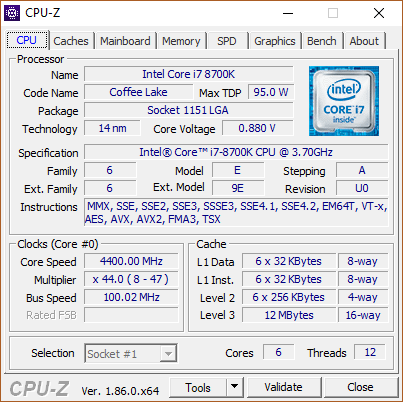
**Цель работы.**

Получение навыков в использовании графических процессоров в качестве сопроцессора для ускорения вычислений.

**Формула вычисления. Вариант задания №6.**

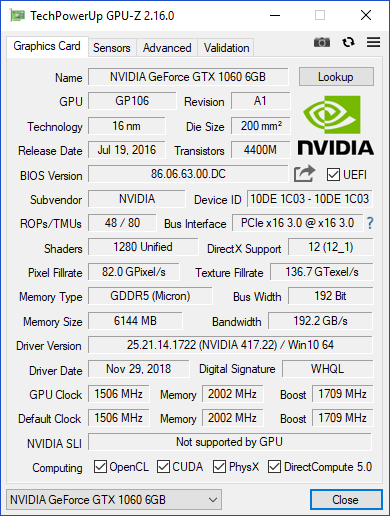
Характеристики ВС, на которой проводились вычисления:

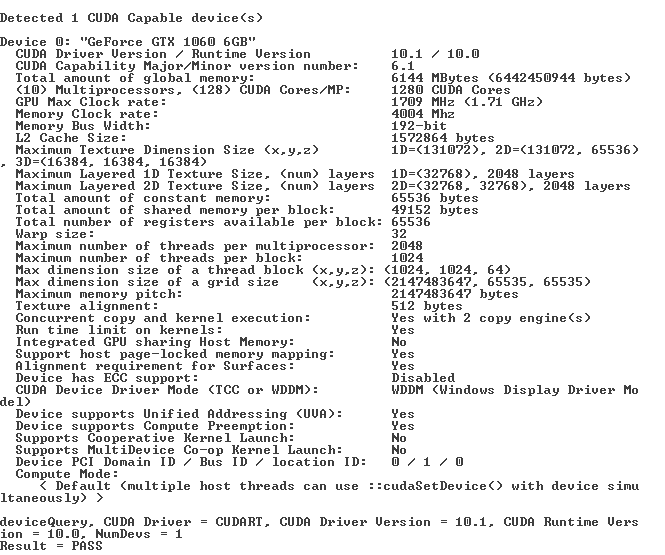
Процессор: Intel Core i7 8700K; Число ядер/потоков: 6/12; Тактовая частота: 4400 МГц;



Операционная система: Windows 10 Pro.

Видеокарта: Nvidia Geforce GTX 1060





3. Программа на CUDA, где все массивы в процессе вычислений хранятся в глобальной памяти.

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

#define M 381 //21\*7 + 234

#define N 512

\_\_global\_\_ void func\_Kernel(char \*A, char \*B, char \*C, double \*D, double \*X, int s)

{

int idx\_thread = blockIdx.x \* blockDim.x + threadIdx.x;

int i, j;

for (i = 0; i < N; i++)

{

for (j = 0; j < 2 \* M; j++)

{

X[idx\_thread\*s + i] = (double)A[idx\_thread\*s + i] \* X[idx\_thread\*s + i] \* (X[idx\_thread\*s + i] \* C[idx\_thread\*s + i] + B[idx\_thread\*s + i]) / D[idx\_thread\*s + i];

}

}

}

int main()

{

char A[N], B[N], C[N];

double X[N], D[N];

char \*dev\_a, \*dev\_b, \*dev\_c;

double \*dev\_d, \*dev\_x;

int steps, amIt;

int blocks[3] = { 1, 2, 4 };

int blocksize[4] = { 1,4,32,256 };

float srd = 0;

float elapsedTime;

int i, j;

while (true)

{

printf("Number of iterations: ");

scanf("%d", &amIt);

int numIt = amIt;

for (int numB = 0; numB < 3; numB++)

{

for (int numT = 0; numT < 4; numT++)

{

amIt = numIt;

while (amIt > 0)

{

cudaEvent\_t start, stop; //ids of events

cudaEventCreate(&start); //inits of events

cudaEventCreate(&stop);

for (i = 0; i < N; i++)

{

A[i] = i + 1;

B[i] = i + 2;

C[i] = i + 3;

D[i] = i + 4;

}

// Choose which GPU to run on, change this on a multi-GPU system.

cudaSetDevice(0);

// Allocate GPU buffers for five vectors

cudaMalloc((void\*\*)&dev\_a, N \* sizeof(char));

cudaMalloc((void\*\*)&dev\_b, N \* sizeof(char));

cudaMalloc((void\*\*)&dev\_c, N \* sizeof(char));

cudaMalloc((void\*\*)&dev\_d, N \* sizeof(double));

cudaMalloc((void\*\*)&dev\_x, N \* sizeof(double));

//blocks = 1; blocksize = 1;

steps = (int)N / (blocks[numB]\*blocksize[numT]);

cudaEventRecord(start, 0); //capture of event start

// Copy input vectors from host memory to GPU buffers.

cudaMemcpy(dev\_a, A, N \* sizeof(char), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b, B, N \* sizeof(char), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_c, C, N \* sizeof(char), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_d, D, N \* sizeof(double), cudaMemcpyHostToDevice);

cudaMemcpy(dev\_x, X, N \* sizeof(double), cudaMemcpyHostToDevice);

// Launch a kernel on the GPU with one thread for each element.

func\_Kernel << <blocks[numB], blocksize[numT] >> > (dev\_a, dev\_b, dev\_c, dev\_d, dev\_x, steps);

// cudaDeviceSynchronize waits for the kernel to finish

cudaDeviceSynchronize();

// Copy output vector from GPU buffer to host memory.

cudaMemcpy(X, dev\_x, N \* sizeof(double), cudaMemcpyDeviceToHost);

cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

cudaFree(dev\_d);

cudaFree(dev\_x);

cudaEventRecord(stop, 0); //capture of event stop

cudaEventSynchronize(stop); //waits for an event to complete

cudaEventElapsedTime(&elapsedTime, start, stop); //computes the elapsed time between events

srd += elapsedTime;

// cudaDeviceReset must be called before exiting in order for profiling and

// tracing tools such as Nsight and Visual Profiler to show complete traces.

cudaDeviceReset();

amIt--;

}

srd /= numIt;

printf("[blocks: %d, threads: %d]Average elapsed time of GPU computing = %f ms\n", blocks[numB], blocksize[numT], srd);

printf("============================================================\n");

srd = 0;

}

}

}

return 0;

}

5. Программа на CUDA, где все массивы в процессе вычислений на device хранятся в разделяемой памяти.

7. Программа на CUDA, где все массивы кроме X в процессе вычислений на

device хранятся в константной памяти, а массив X в разделяемой памяти.

10. Программы на CUDA пункта 5 и 7, но в них считается доступной для хранения массивов емкость разделяемой памяти по 16 Кб на блок, в котором размещаются и эффективно считаются данные их массивов.

*Таблица 1. Время выполнения программ пунктов 3, 5, 7 в мс*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Времена выполнения вычислений на GPU при N = 512 | | | | | | | | | | | | |
|  | Число блоков / число потоков в блоке | | | | | | | | | | | |
| 1/1 | 1/4 | 1/32 | 1/256 | 2/1 | 2/4 | 2/32 | 2/256 | 4/1 | 4/4 | 4/32 | 4/256 |
| Только глобальная память | 314.498 | 332.656 | 413.816 | 497.149 | 311.526 | 334.245 | 435.308 | 479.949 | 312.978 | 333.207 | 405.471 | 439.544 |
| Разделяемая память |  |  |  |  |  |  |  |  |  |  |  |  |
| Разделяемая и константная память |  |  |  |  |  |  |  |  |  |  |  |  |

*Таблица 2. Время выполнения программ пункта 10 в мс*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Времена выполнения вычислений на GPU при N = 65536 | | | | | | | | | | | | |
|  | Число блоков / число потоков в блоке | | | | | | | | | | | |
| 1/1 | 1/4 | 1/32 | 1/256 | 2/1 | 2/4 | 2/32 | 2/256 | 4/1 | 4/4 | 4/32 | 4/256 |
| Только глобальная память |  |  |  |  |  |  |  |  |  |  |  |  |
| Разделяемая память |  |  |  |  |  |  |  |  |  |  |  |  |
| Разделяемая и константная память |  |  |  |  |  |  |  |  |  |  |  |  |

*Таблица 3.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | число блоков / число потоков в блоке | | | | | | | | | | | |
| N | Память | 1/1 | 1/4 | 1/32 | 1/256 | 2/1 | 2/4 | 2/32 | 2/256 | 4/1 | 4/4 | 4/32 | 4/256 |
| 512 | Glob | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Shar |  |  |  |  |  |  |  |  |  |  |  |  |
| S&C |  |  |  |  |  |  |  |  |  |  |  |  |
| 65536 | Glob | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Shar |  |  |  |  |  |  |  |  |  |  |  |  |
| S&C |  |  |  |  |  |  |  |  |  |  |  |  |